HAL
open science

BeleniosVS: Secrecy and Verifiability against a Corrupted Voting Device

Véronique Cortier, Alicia Filipiak, Joseph Lallemand

To cite this version:

Véronique Cortier, Alicia Filipiak, Joseph Lallemand. BeleniosVS: Secrecy and Verifiability against a Corrupted Voting Device. CSF 2019 - 32nd IEEE Computer Security Foundations Symposium, Jun 2019, Hoboken, United States. hal-02268399

HAL Id: hal-02268399

https://inria.hal.science/hal-02268399v1

Submitted on 20 Aug 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

BeleniosVS: Secrecy and Verifiability against a Corrupted Voting Device

Véronique Cortier
CNRS, Loria, France
veronique.cortier@loria.fr

Alicia Filipiak
Orange Labs, France
alicia.filipiak@orange.com

Joseph Lallemand
Inria, Loria, France
joseph.lallemand@loria.fr

***Abstract*—Electronic voting systems aim at two conflicting properties, namely privacy and verifiability, while trying to minimise the trust assumptions on the various voting components. Most existing voting systems either assume trust in the voting device or in the voting server.**

We propose a novel remote voting scheme BeleniosVS that achieves both privacy and verifiability against a dishonest voting server as well as a dishonest voting device. In particular, a voter does not leak her vote to her voting device and she can check that her ballot on the bulletin board does correspond to her intended vote. More specifically, we assume two elections authorities: the voting server and a registrar that acts only during the setup. Then BeleniosVS guarantees both privacy and verifiability against a dishonest voting device, provided that not both election authorities are corrupted. Additionally, our scheme guarantees receipt-freeness against an external adversary.

We provide a formal proof of privacy, receipt-freeness, and verifiability using the tool ProVerif, covering a hundred cases of threat scenarios. Proving verifiability required to develop a set of sufficient conditions, that can be handled by ProVerif. This contribution is of independent interest.

1. Introduction

Internet voting offers a convenient way for voting, getting rid of physical voting booths. Voters may vote using their own device, from any place. Internet voting is now used on a regular basis in several countries such as Estonia [1], Australia [2], or Switzerland [3], at least on trial. On the other hand, other countries like Germany or Norway have stopped or even banned electronic voting due to the fear that votes may be manipulated or that privacy may be lost. This fear is supported by several attacks on real systems [4]–[7]. Besides national elections, Internet voting is used in many elections of lower stake, such as professional elections (*e.g.* unions), administrative boards, or elections in the academic world, from the election of the president of the university [8] to the selection of the invited speakers at the next conference. In these cases, Internet voting is often seen as an alternative means for remote elections.

Internet voting as well as traditional paper-based voting aim at two main security properties.

- *Privacy*: no one should know how I voted. This property guarantees that a voter may vote in real independence. Depending on the context of the election, privacy may be not sufficient. Instead, the system should guarantee that a voter may not *prove* how she voted, to prevent vote-buying or coercion. This property is often called *receipt-freeness* or *coercion-resistance*.
- *Verifiability*: the system should offer a means for voters to check that the result of the election corresponds to the intent of the voters, without having to trust the voting server nor the voting devices. Typically, verifiability encompasses several more atomic properties: the ballot cast by a voter should match her intent; the ballot box should contain all the ballots cast by voters, provided they are legitimate voters; finally, the result should correspond to the ballots in the ballot box.

Related work. Many voting systems have been proposed to address at least some of these requirements. There are two main types of electronic voting systems. The first one is on-site voting: voters go to traditional polling stations and use on-site computers, possibly using voting sheets that contain cryptographic material. Examples of such systems include Scantegrity [9], Prêt-à-Voter [10], and Star-Vote [11]. In this paper, we will focus on the second family, Internet voting, where voters use their own device, from remote places. Civitas [12] is probably the only system that achieves both verifiability and coercion-resistance, assuming a rather demanding setup (voters need a public key infrastructure - PKI). In Civitas, voters have to trust their voting device both for privacy and verifiability. In particular, Civitas does not have the *cast-as-intended property*: voters have no guarantee that their voting device encrypts the right vote, when it is corrupted. Helios [13] is a simple protocol (both private and verifiable, but not coercion-resistant) that includes a "cast or audit" mechanism. Voters may interact with their voting device, requesting the device to provide the randomness used for encryption, in order to check (thanks to a third party) that it always encrypts the desired vote. However, these interactions may be cumbersome and difficult to understand. Select [14] achieves cast-as-intended in a simpler way: voters are given a tracking number. After the tally, voters may check that their tracking number appears next to their vote. Selene [15] builds upon the same idea but further guarantees receipt-freeness thanks to a rather complex cryptographic

machinery: despite the tracking number, voters cannot prove how they voted. Designed for a deployment in Switzerland, the Neuchâtel protocol [16] and CHVote [3] provide cast-as-intended thanks to return codes. Voters vote with their voting device and then receive a return code. They should check that the code corresponds to the desired candidate using the voting sheet they received during the setup phase. [16] does not achieve end-to-end verifiability: voters have to trust the voting server to be convinced that the result corresponds to the received ballots and that no ballots were added. CHVote [3] does not place its trust in a single voting server but requires instead several independent servers, among which one should be trusted.

Noticeably, all these systems assume the voting device to be honest *w.r.t.* privacy. Indeed, since voters enter directly their choice in their device, the device may leak how they voted to a third party. Since voting devices are typically smartphones, tablets, or personal computers, it is really hard to guarantee that such a vote leak does not occur, especially since the voting device is connected to Internet during the voting phase. An option is to communicate through an insecure device (the voter's smartphone) and discharge the sensitive operations to a trusted device, as proposed in the Alethea system [17]. This requires a rather costly infrastructure, justified in Alethea by the fact that only a (random) fraction of the voters do vote. Moreover, this solution still assumes a trusted voting device.

Our goal is to design a voting system that achieves verifiability *and* privacy *w.r.t.* a corrupted voting server and a corrupted voting device. This requires in particular that our system should be cast-as-intended and private *w.r.t.* the voting device. More precisely, our protocol will make use of two voting devices (*e.g.* a smartphone and a computer), any of which could be corrupted (but not both at the same time). One will be used to actually cast the vote, and the other one for auditing purposes. To our knowledge, the only system that achieves verifiability and privacy in this corruption scenario is the D-Demos system [18], a successor of Demos [19]. In D-Demos, voters vote using vote codes (in the spirit of code voting [20]). Each voting sheet has two (similar) parts. A voter selects one part at random and sends the vote code corresponding to the candidate of her choice. The second part is used for auditing, to make sure that the vote codes are correctly associated with the candidates. Note that a dishonest voting sheet generator may try to cheat, *e.g.* by having all vote codes encoding the same candidate on the first half of the sheet. The voter has probability 1/2 to catch this dishonest behaviour (if she choses to audit the first half). The initial Demos system relies on a centralised election authority that maintains all secrets and knows the voters' votes, and is therefore a single point of failure *w.r.t.* privacy. Instead, D-Demos explains how to thresholdise the election authority (and the ballot box), introducing a set of vote collectors. Hence in practice, the system requires several independent servers with full availability during the voting phase. This means independent softwares, independent machines, owned by independent companies, which may be too costly for many election contexts.

Our contribution. We propose a new voting system, BeleniosVS, that achieves both privacy and verifiability, against a dishonest voting server and a dishonest voting client. "VS" stands for *voting sheet*: as for D-Demos, voters receive a voting sheet with vote codes used for voting. A vote code simply corresponds to the encrypted vote. It can be encoded *e.g.* through a QR code, scanned by the voter. Interestingly, the sheet used for voting is the same that is audited. This way, the voter is ensured that her vote has been cast-as-intended (in contrast with the 1/2 probability of being fooled without detection in the D-Demos system). This audit process is optional and can be delegated to a third party. The reason why the voting sheet may be learned without compromising voter's privacy is that both the voting device and the voting server re-randomise the ballot before publishing it on the ballot box. Provided that the voting device, the voting server, and the voting sheet issuer (a.k.a. registrar) are not all corrupted - and that the auditing device is either honest or not used at all - then privacy is guaranteed. End-to-end verifiability is then ensured in a rather standard way: the published ballots are either tallied homomorphically or run through a mixnet, and then decrypted, with zero-knowledge proofs of correct mixing and decryption (the decryption key is shared among several talliers in a standard fashion). Our system is based on BeleniosRF [21], that has verifiability and privacy (and receipt-freeness) against a dishonest voting server. However, BeleniosRF assumes the voting device to be trusted for both properties. As for BeleniosRF, we strongly rely on a re-randomisation functionality of both the ciphertext, the signature, and the zero-knowledge proof used to form a ballot. This functionality, based on pairing, is due to Blazy *et al.* [22]. Interestingly, BeleniosVS inherits receipt-freeness from BeleniosRF: a voter cannot prove to an adversary how she voted, even if she provides him with her voting sheet and the randomness used by her voting device. Receipt-freeness is guaranteed by the fact that the voting server re-randomises ballots before publishing them. Note however that this does not prevent vote buying: if a voter provides all her credentials then an adversary may vote in her place. Protecting against vote buying would require a stronger setting, such as the existence of a PKI for voters like in Civitas, for instance.

We formally prove privacy, receipt-freeness, and verifiability in a symbolic model, using the ProVerif tool [23]. ProVerif is a state-of-the art automatic prover for security protocols. It has already been used to analyse hundreds of protocols of the literature, including TLS [24], voting protocols [25], and avionic protocols [26]. We chose an automatic tool to conduct our security analysis in order to analyse multiple corruption scenarios. Indeed, the BeleniosVS protocol involves several components, namely the voting device, the auditing device, the voting service, and the voting sheet issuer. Any of these components may be corrupted. In addition, we also consider the (realistic) case where a voter simply loses her voting sheet and/or her password. We analyse all possible combinations of corruption scenarios, yielding 96 cases.

Actually, the ProVerif tool cannot automatically prove end-to-end verifiability. Indeed, verifiability roughly says that

the result should correspond to the honest voters' intent plus at most k dishonest votes, where k is the number of dishonest voters. Such a property requires to count the number of votes for each candidate, which is beyond the scope of ProVerif. Instead, we devise a set of sufficient conditions that can be proved with ProVerif. Interestingly, these properties correspond to the intuitive notions of recorded-as-intended (that is cast-as-intended and recorded-as-cast), eligibility, and universal verifiability, often used to described end-to-end verifiability. This result extends the approach of [27] to the case of dishonest voters, preventing ballot stuffing. It is of independent interest and could be used to analyse other voting protocols, especially with ProVerif.

Discussion. As for Belenios and BeleniosRF, BeleniosVS assumes one of the election authorities to be honest (here either the voting server or the voting sheet issuer, also called registrar). This assumption is common in voting schemes that aim at both verifiability and receipt-freeness/coercion resistance. For instance, in Civitas, colluding authorities at the setup can learn all valid credentials and use them to cast votes thus resulting in an undetectable manipulation of the election. Selene also assumes one of the election authorities to be honest for verifiability. One interesting feature of BeleniosVS is that we do not assume a PKI (as in D-Demos but with a much lighter infrastructure) since a PKI for all voters is difficult to achieve in practice (except in some countries like Estonia) and often hides further implicit trust assumptions. Interestingly, BeleniosVS is easy to adapt in case voters have a PKI, with stronger guarantees.

2. Overview of BeleniosVS

2.1. Preliminary notions

Our protocol is inspired by the BeleniosRF protocol [21] and uses the same cryptographic primitive, namely *signatures on randomisable ciphertexts* [22]. This primitive allows a message to be encrypted, and then signed. The ciphertext is linked with the signing key, so that it can only be signed with this specific key. The ciphertext can then be re-randomised by anyone and the signature can be adapted to still be valid for the randomised ciphertext, without knowing the underlying signing key.

We introduce here some notations, refering the reader to [22] for the full definition of the primitives. If k denotes a private (decryption) key, then $\mathsf{pk}(k)$ is the associated public (encryption) key. Similarly, if k' denotes a signing key, $\mathsf{spk}(k')$ is the associated verification key. Let us denote $\mathsf{aenc}(m, \mathsf{pk}(k), \mathsf{spk}(k'), r)$ the encryption of message m under $\mathsf{pk}(k)$, randomised with random value r, and intended to be signed with k'. Similarly, let us also denote $\mathsf{sign}(c, \mathsf{pk}(k), k', t)$ the signature of a ciphertext c with key k', randomised with random value t, where c is intended to be a ciphertext produced with $\mathsf{pk}(k)$. Note that the encryption actually combines an El Gamal ciphertext with a Groth-Sahai proof that protects against malleability (still allowing re-randomisation). The actual algorithms are not relevant for the presentation of BeleniosVS and are therefore left abstract here.

The *signatures on randomisable ciphertexts* primitive provides an algorithm $\mathsf{rand}(c, s, \mathsf{pk}(k), \mathsf{spk}(k'), r', t')$ that randomises a ciphertext c and a signature s, associated with the public keys $\mathsf{pk}(k)$ and $\mathsf{spk}(k')$, using two new random values r', t'. This algorithm satisfies the following relation:

$$\begin{aligned} & \mathsf{rand}(c, \mathsf{sign}(c, \mathsf{pk}(k), k', t), \mathsf{pk}(k), \mathsf{spk}(k'), r', t') \\ = \; & \langle c', \mathsf{sign}(c', \mathsf{pk}(k), k', t + t') \rangle \\ \text{where} \; & c = \mathsf{aenc}(m, \mathsf{pk}(k), \mathsf{spk}(k'), r) \\ & c' = \mathsf{aenc}(m, \mathsf{pk}(k), \mathsf{spk}(k'), r + r'). \end{aligned}$$

This property is at the core of BeleniosRF and BeleniosVS. Ballots will be re-randomised both by the voting device and the voting server.

- Randomising the ballots ensures *receipt-freeness*, as Alice cannot reconstruct her ballot once it has been randomised with some random value she does not know. Thus she cannot prove what vote it encrypts.
- Even if an attacker knows the link between an encrypted vote and the plaintext vote, this link is lost after re-randomisation. This is a key element to ensure *vote privacy* when the voting devices are corrupted.

2.2. The BeleniosVS protocol

The BeleniosVS protocol involves seven main entities. The *election administrator* publishes the parameters of the election: the list of eligible voters, and of candidates. The *registrar* generates voting sheets, which contain encrypted and signed ballots for all candidates, and distributes them to the *voters*. The *voters* then use their *voting device* to scan the ballot (a code) corresponding to their candidate. Their voting device authenticates to the *voting server* using a password entered by the voter, and sends the ballot to the server. Optionally, before casting their vote, the voters may use an *auditing device* to check that the voting sheet they received is correctly constructed. The voting server publishes all the ballots on a bulletin board. Once the voting phase is over, the *tallying authority* computes and publishes the result of the election. The bulletin board is public and can thus be audited by anyone, for example to check that the server has only accepted ballots signed with keys of eligible voters, and has not accepted two ballots signed by the same key.

The core voting process of BeleniosVS is summarised in Figure 1 and we describe it in more details in the following sections.

The tallying authority first generates a pair of asymmetric encryption keys $(\mathsf{sk_e}, \mathsf{pk_e})$ for the election. The public key is then communicated to the election administrator, who publishes it, together with the lists of eligible voters and of possible votes. Once the voting phase is closed, the tallying authority retrieves the public bulletin board. It then runs the list of corresponding ciphertexts through a mixnet, decrypts each element using the election secret key $\mathsf{sk_e}$, and generates zero-knowledge proofs that the mixing and decryption are

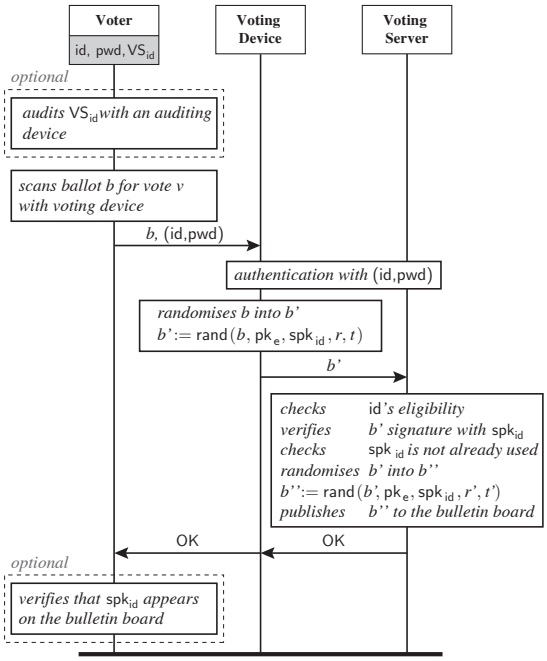

Figure 1: Overview of the BeleniosVS voting process, where $\mathsf{VS}_{\mathsf{id}}$ is defined in Figure 2.

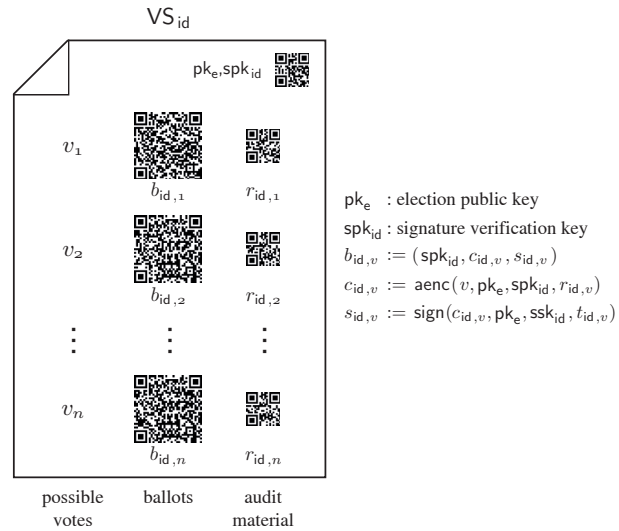

Figure 2: Structure of a voting sheet $\mathsf{VS}_{\mathsf{id}}$ for voter id.

correct. Finally it outputs the list of the decrypted votes and the proofs of correct decryption. Alternatively, the ciphertexts may be homomorphically added before decrypting their sum (also with a proof of correct decryption). As usual, the decryption key can be thresholdised among several tallying authorities [28].

The registrar generates a voting sheet for each voter identity id as follows. For each voter id, the registrar generates a pair of signature and verification keys $(\mathsf{ssk}_{\mathsf{id}}, \mathsf{spk}_{\mathsf{id}})$. Then, for each candidate v, the registrar encrypts v with the public election key pk_{e}, and a random number $r_{\mathsf{id},v}$, yielding the ciphertext $c_{\mathsf{id},v} = \mathsf{aenc}(v, \mathsf{pk}_{\mathsf{e}}, \mathsf{spk}_{\mathsf{id}}, r_{\mathsf{id},v})$. The ciphertext also embeds a verification key $\mathsf{spk}_{\mathsf{id}}$, associated with voter id. The registrar signs the ciphertext with the signing key $\mathsf{ssk}_{\mathsf{id}}$, and a random number $t_{\mathsf{id},v}$, yielding the signature $s_{\mathsf{id},v} = \mathsf{sign}(c_{\mathsf{id},v}, \mathsf{pk}_{\mathsf{e}}, \mathsf{ssk}_{\mathsf{id}}, t_{\mathsf{id},v})$. So each voting sheet contains one line per candidate v, with the corresponding ballot $b_{\mathsf{id},v} = (\mathsf{spk}_{\mathsf{id}}, c_{\mathsf{id},v}, s_{\mathsf{id},v})$ composed of the verification key, the ciphertext and the signature, as well as the randomness $r_{\mathsf{id},v}$ used for the encryption. This random value $r_{\mathsf{id},v}$ will allow the voter to audit the sheet. The structure of a voting sheet is depicted in Figure 2. The voting sheet is then sent to the voter.

The registrar, when honest, is assumed to delete all her secret data (signing keys and randomness) as it is no longer of use to her. Note that the security of BeleniosVS could increase if we could assume voters to have a signing key. This way, even a dishonest registrar colluding with the voting server could not add ballots. However, this would require that either voters are engaged in the setup phase or that they already have a signing key, for example in their electronic ID card. We chose to propose a setup that is more realistic given the current state of affairs.

The voter Alice receives a voting sheet from the registrar. Optionally, before casting her vote, Alice may scan her voting sheet with her auditing device to check that it is properly constructed. She may also ask a third party (a friend or a colleague) to do so. To cast a vote for candidate v, she uses her voting device to scan the ballot $b_{Alice,v}$ corresponding to v (that is, displayed next to it). The rest of the voting sheet is physically hidden from the voting device. In particular, the voting device cannot see the vote v itself, the randomness $r_{Alice,v}$, nor any other ballot. Alice also enters her password. Once her ballot has been properly cast, she receives a confirmation from her voting device.

Finally, Alice may verify that her ballot has been correctly cast. She does so by checking that the public bulletin board indeed contains a ballot signed with her key. For usability reasons, verification keys are appended to the ballots. Therefore Alice simply needs to check that her verification key spk_{Alice} (on her voting sheet) appears on the bulletin board. External auditors can guarantee that each ballot includes a valid signature corresponding to the key next to it. This check can be done at any time. It assumes that Alice has access at least once to the true bulletin board, thus using an honest device (be it her voting or auditing device, or a third device).

The voting device reads the ballot $b_{Alice,v}$ selected by Alice. It randomises the encryption and signature, yielding a randomised ballot $b'_{Alice,v} = \mathsf{rand}(b_{Alice,v}, \mathsf{pk}_{\mathsf{e}}, \mathsf{spk}_{Alice}, r, t)$ (for some random nonces r, t). It then uses Alice's password to authenticate to the voting server, and sends $b'_{Alice,v}$ to the server. The voting device waits for a confirmation from the server that the ballot has been received, and displays this confirmation to Alice.

The voting server generates and sends passwords (privately to voters). During the voting phase, it receives ballots from voters. When it receives a ballot $b'_{Alice,v}$ from Alice (authenticated with her password), it checks that Alice is an eligible voter, and that $b'_{Alice,v}$ is well-formed, *i.e.* that it is composed of a ciphertext and a signature with Alice's key. The server also ensures that no ballot signed by Alice's key has already been cast. If these tests succeed, it randomises the ballot again, yielding a ballot $b''_{Alice,v} = \mathsf{rand}(b'_{Alice,v}, \mathsf{pk_e}, \mathsf{spk}_{Alice}, r', t')$ (for some random nonces r', t'), which is published on the bulletin board. The voting server finally provides the voting device with a confirmation that the ballot was accepted.

The auditing device. If Alice chooses to audit her voting sheet, she scans the entire sheet on her auditing device. The device checks that each line of the voting sheet is of the form $(v, \mathsf{spk}_{Alice}, c, s, r)$, with $c = \mathsf{aenc}(v, \mathsf{pk_e}, \mathsf{spk}_{Alice}, r)$ and such that $\mathsf{verify}(c, s, \mathsf{spk}_{Alice}) = \mathtt{true}$. Note that the data $\mathsf{pk_e}$ and spk_{Alice} are also available on the voting sheet. However, the login and password are not given to the auditing device and therefore it does not have enough material to cast a vote. It cannot learn Alice's vote either since her ballot will be re-randomised before its publication.

2.3. Practical considerations

In terms of computational power, the main bottleneck is the voting device of the voter. Compared to BeleniosRF, the voting device does not need anymore to encrypt and sign a ballot but instead has to re-randomise the encryption (that is an El Gamal ciphertext and a Groth-Sahai proof) and the signature. In terms of atomic operations (exponentiation, multiplication, pairing), the cost is similar. Therefore, the benchmarks from [21] remain valid. The experiments were conducted with a BN curve on a 254-bit prime and show for example about 7s for a 2016 phone with SD 9810 to form a ballot with a 5-bits payload (which is enough to handle most elections in a mixnet mode). The computation time therefore remains reasonable and will improve over time.

To cast a vote, a voter simply needs to scan one QR code. It is important however that a malicious voting device does not "see" the rest of the voting sheet. This can be achieved by folding the paper or providing a mask to the voter. Auditing a voting sheet requires more work. The direct approach consists in scanning all the codes one by one, which can be quite cumbersome. Instead, we believe that voters could directly scan the entire voting sheet (most cameras now have a good definition) and the auditing device could recognise all the QR codes by itself. However, we acknowledge that we need to carefully adjust the required size of a QR code, for reasonable security parameters and study the associated usability level [29]. At this stage, BeleniosVS can probably be used for a small number of candidates only.

The rest of the paper is devoted to the formal analysis of BeleniosVS.

$M, N, U ::=$	terms
$x \mid n \mid f(M_1, \ldots, M_k)$	where $x \in \mathcal{X}$, $n \in \mathcal{N}$, and $f \in \mathcal{C}$
$D ::=$	expressions
$M \mid h(D_1, \ldots, D_k) \mid \mathsf{fail}$	where $h \in \mathcal{C} \cup \mathcal{D}$
$\phi ::=$	formulas
$D = D' \mid \mathtt{true} \mid \mathtt{false} \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi$	
$P, Q ::=$	processes
0	nil
$\mathsf{out}(N, M); P$	output
$\mathsf{in}(N, x : T); P$	input
$P \mid Q$	parallel composition
$!P$	replication
$\mathsf{new}\ a : T; P$	restriction
$\mathsf{let}\ x : T = D\ \mathsf{in}\ P\ \mathsf{else}\ Q$	assignment
$\mathsf{insert}\ \mathsf{Tbl}(M); P$	write in table
$\mathsf{get}\ \mathsf{Tbl}(x : T)\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ P\ \mathsf{else}\ Q$	read from table
$\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional
$\mathsf{event}(M); P$	event

Figure 3: Syntax of the core language of ProVerif.

3. Formal model

Protocols are typically analysed either in symbolic or cryptographic models. We choose here symbolic models, that offer a higher level of automation, at the cost of a more coarse-grained abstraction. In this section, we briefly describe the core syntax and semantics of the ProVerif tool, as defined in [23]. We next explain how to express vote secrecy, receipt-freeness, and verifiability.

3.1. Syntax

As usual, messages sent over the network are modelled as terms and protocols as processes. The corresponding syntax is displayed in Figure 3. In what follows, $\{\!|a, a, b|\!\}$ denotes the multiset composed of two instances of a and one of b. If M, M' are multisets and v an element, $M(v)$ is the number of instances of v in M, $M \uplus M'$ is the union of M and M', and we write $M \subseteq_{\mathsf{m}} M'$ if M is a sub-multiset of M'.

Terms and expressions. Terms are built over a set $\mathcal{X}$ of variables, a set $\mathcal{N}$ of names, a set $\mathcal{T}$ of types, and a set Tbl of table names. Symbols for functions are either *constructors* (in set $\mathcal{C}$) or *destructors* (in set $\mathcal{D}$). Expressions are built using both constructors and destructors and represent cryptographic computations, while terms contain only constructors and represent actual messages. Function symbols are given with their types: $g(T_1, \ldots, T_n) : T$ means that the function g takes n arguments as input, of types respectively $T_1, \ldots, T_n$, and returns a result of type T. By default in ProVerif, types include channel for channel names, and

bitstring for any messages (also written any). A substitution $\sigma = \{U_1/x_1, \ldots, U_n/x_n\}$ associates variables with terms. The application of a substitution σ to a term U is denoted $U\sigma$ and is defined as usual. In what follows, we consider only well typed substitutions.

Each destructor d corresponds to a cryptographic operation (*e.g.* decryption), represented by a rewrite rule of the form $d(U_1, \ldots, U_n) \to U$, where $U_1, \ldots, U_n, U$ are terms. The evaluation of an expression D is defined as expected: we write $D \Downarrow U$ if, applying the rewrite rules, D reduces to a term U. In case no rewrite rule can be applied, and D still contains destructors, we say the evaluation fails, and write $D \Downarrow \mathsf{fail}$.

The evaluation $[\![\phi]\!]$ of a formula ϕ is defined by $[\![D = D']\!] = \top$ if D and D' evaluate to the same term, $[\![D = D']\!] = \bot$ otherwise, and is then extended to $\mathtt{true}, \mathtt{false}, \wedge, \vee, \neg$ as expected.

Example 1. *To model concatenation, addition, and signatures on randomisable ciphertexts (Section 2.1), we consider types* rand, skey, pkey, sskey, spkey, *respectively for random nonces, private and public encryption keys, and signature and verification keys. The constructor and destructor symbols are displayed in Figure 4 with their associated rewrite rules.*

We may write $\langle m_1, m_2, \ldots, m_k\rangle$ *for* $\langle m_1, \langle m_2, \langle \ldots, m_k\rangle\rangle\rangle$. *We may also write* $\mathsf{verify}(\langle c, s\rangle, \mathsf{spk}(k))$ *for* $\mathsf{verify}(c, s, \mathsf{spk}(k))$.

Processes. Protocols are formally modelled as *processes* defined in Figure 3. Process $\mathsf{out}(N, M); P$ outputs message M on channel N and then proceeds with P. Process $\mathsf{in}(N, x : T); P$ inputs on channel N a message stored in variable x. Process $P \mid Q$ represents the parallel execution of P and Q, while $!P$ stands for process P that can be run an arbitrary number of times (in parallel). $\mathsf{new}\, a : T; P$ generates a fresh name a. Process $\mathsf{let}\ x : T = D\ \mathsf{in}\ P\ \mathsf{else}\ Q$ evaluates D, stores the result in x, and behaves like P unless the evaluation fails, in which case it behaves like Q. Process $\mathsf{event}(M); P$ introduces an *event* that is used to specify security properties as explained in Section 3.3.1. The events are invisible to the adversary and simply reflect that an agent has reached a specific state with some specific values. $\mathsf{insert}\ \mathsf{Tbl}(M);\ P$ inserts an entry for message M in table Tbl. $\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q$ behaves like P if ϕ evaluates to $\top$, and like Q otherwise. $\mathsf{get}\ \mathsf{Tbl}(x : T)\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ P\ \mathsf{else}\ Q$ reads (non-deterministically) an entry of type T from table Tbl such that the condition ϕ evaluates to $\top$, and behaves like P; or, if there is no such entry in Tbl, behaves like Q.

We denote by $\mathsf{fn}(P)$ (resp. $\mathsf{fv}(P)$) the free names (resp. free variables) of a process P. A process is *closed* if it has no free variables. Following ProVerif's style, we may write $\mathsf{in}(c, {=}x).P$ instead of $\mathsf{in}(c, y : T).\mathsf{if}\ x = y\ \mathsf{then}\ P$; and $\mathsf{in}(c, \langle x : T, y : T'\rangle).P$ instead of $\mathsf{in}(c, z : \mathsf{any}).\mathsf{let}\ x : T = \mathsf{proj1}(z)\ \mathsf{in}\ \mathsf{let}\ y : T' = \mathsf{proj2}(z)\ \mathsf{in}\ P$. We also write $\mathsf{get}\ \mathsf{Tbl}(x : T)\ \mathsf{in}\ P\ \mathsf{for}\ \mathsf{get}\ \mathsf{Tbl}(x : T)\ \mathsf{suchthat}\ \mathtt{true}\ \mathsf{in}\ P\ \mathsf{else}\ 0$.

Example 2. *In BeleniosVS, a voter* id *willing to vote for A first receives a voting sheet from the registrar, then scans the ballot corresponding to A on her voting device, and waits for confirmation that her ballot was received. She finally verifies that a ballot signed with her key appears on the public bulletin board. We model this role by the process* $\mathsf{Voter}(\mathsf{id}, A, c_{vs}, c_{vd})$ *depicted in Figure 5.*

The communication with the registrar and the voting device is modelled using private channels c_{vs} and c_{vd}. The voter process triggers an event Voter *that records the fact that the voter received a voting sheet with a verification key* $\mathsf{spk}_{\mathsf{id}}$. *It then triggers a* Voted *event, to record the voter's intended vote. Finally, the process triggers the* Verified *event at the end of the vote procedure, to express the fact that the voter is satisfied that her vote has been counted.*

3.2. Semantics

A *configuration* $E, \mathcal{P}, \mathcal{S}$ is composed of a multiset $\mathcal{P}$ of processes, that represents the currently active processes, sets $E = (\mathcal{N}_{\mathsf{pub}}, \mathcal{N}_{\mathsf{priv}})$ that model respectively the public and private names used so far, and a mapping $\mathcal{S}$ from table names to sets of messages, representing the contents of the tables. The behaviour of processes described earlier is formalised as a reduction relation $\to$ between configurations, defined as expected (see Appendix B and [23]). A *trace* is a sequence of reductions between configurations $E_0, \mathcal{P}_0, \mathcal{S}_0 \to \cdots \to E_n, \mathcal{P}_n, \mathcal{S}_n$. In particular, we say that a trace $t = E_0, \mathcal{P}_0, \mathcal{S}_0 \to^* E', \mathcal{P}', \mathcal{S}'$ *executes* an event M, denoted $M \in t$, if t contains a reduction $E, \mathcal{P}, \mathcal{S} \cup \{\mathsf{event}(M); P\} \to E, \mathcal{P} \cup \{P\}, \mathcal{S}$ for some $E, \mathcal{P}, \mathcal{S}, P$. By a slight abuse of notations, we say that t is a trace of $\mathcal{P}$ if t is a trace that starts from $(\mathsf{fn}(\mathcal{P}), \emptyset), \mathcal{P}, \emptyset$.

3.3. Properties

Security properties are typically modelled through two main families of properties, namely correspondence and equivalence properties. Correspondence properties are used to express that in any execution of a protocol, an event (*e.g.* the publication of the result of the election) is always preceded by other events satisfying some relation (*e.g.* honest voters did vote consistently *w.r.t.* the outcome). This is particularly useful to state *e.g.* authentication properties. Equivalence states that two processes cannot be distinguished by an attacker. It is often used to specify privacy properties like anonymity or, here, ballot privacy.

3.3.1. Correspondence. A property such as authentication typically states that whenever Alice reaches some state, possibly with some value for a key, then it must be the case that Bob initiated a session with Alice, with the same values. ProVerif allows to specify such properties through correspondence assertions.

Definition 1 (Correspondence [23])**.** *A closed process P_0 satisfies the correspondence*

$$\bigwedge_{i=1}^{n} \mathsf{event}(M_i) \wedge \bigwedge_{i=1}^{n'} \phi_i \rightsquigarrow \bigvee_{i=1}^{m} \left(\bigwedge_{j=1}^{l_i} \mathsf{event}(M_{i,j}) \wedge \bigwedge_{j=1}^{l'_i} \phi_{i,j} \right)$$

$$
\begin{array}{ll}
\mathcal{C}_{\mathsf{rand}} = & \\
\quad \{ & \mathtt{true} : \mathsf{any}, \\
& \langle \mathsf{any}, \mathsf{any} \rangle : \mathsf{any}, \\
& \mathsf{rand}{+}\mathsf{rand} : \mathsf{rand}, \\
& \mathsf{pk}(\mathsf{skey}) : \mathsf{pkey}, \\
& \mathsf{aenc}(\mathsf{any}, \mathsf{pkey}, \mathsf{spkey}, \mathsf{rand}) : \mathsf{any}, \\
& \mathsf{spk}(\mathsf{sskey}) : \mathsf{spkey}, \\
& \mathsf{sign}(\mathsf{any}, \mathsf{pkey}, \mathsf{sskey}, \mathsf{rand}) : \mathsf{any} \quad \}
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{D}_{\mathsf{rand}} = & \\
\quad \{ & \mathsf{proj1}(\mathsf{any}) : \mathsf{any}, \mathsf{proj2}(\mathsf{any}) : \mathsf{any}, \\
& \mathsf{adec}(\mathsf{any}, \mathsf{skey}) : \mathsf{any}, \\
& \mathsf{verify}(\mathsf{any}, \mathsf{any}, \mathsf{spkey}) : \mathsf{any}, \\
& \mathsf{rand}(\mathsf{any}, \mathsf{any}, \mathsf{pkey}, \mathsf{spkey}, \mathsf{rand}, \mathsf{rand}) : \mathsf{any} \quad \}
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{R}_{\mathsf{rand}} = & \\
\quad \{ & \mathsf{proj1}(\langle x, y \rangle) \to x, \\
& \mathsf{proj2}(\langle x, y \rangle) \to y, \\
& \mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y), z, r), y) \to x, \\
& \mathsf{verify}(c, \mathsf{sign}(c, \mathsf{pk}(y), z, s), \mathsf{spk}(z)) \to \mathtt{true}, \\
& \mathsf{rand}(c, \mathsf{sign}(c, \mathsf{pk}(y), z, s), \mathsf{pk}(y), \mathsf{spk}(z), r', s') \to \\
& \quad \langle \mathsf{aenc}(x, \mathsf{pk}(y), \mathsf{spk}(z), r{+}r'), \mathsf{sign}(\mathsf{aenc}(x, \mathsf{pk}(y), \mathsf{spk}(z), r{+}r'), \mathsf{pk}(y), z, s{+}s') \rangle \\
& \quad \text{with } c = \mathsf{aenc}(x, \mathsf{pk}(y), \mathsf{spk}(z), r) \quad \}
\end{array}
$$

Figure 4: Equational theory for re-randomisable encryption and signature.

$$
\begin{array}{l}
\mathsf{Voter}(\mathsf{id}, A, c_{vs}, c_{vd}) = \\
\quad \mathsf{in}(c_{vs}, vs : \mathsf{any}); \\
\quad \mathsf{let}\ \langle \mathsf{spk}_{\mathsf{id}} : \mathsf{spkey}, \\
\qquad = A, \langle c_A : \mathsf{any}, s_A : \mathsf{any}, r_A : \mathsf{rand} \rangle, \\
\qquad = B, \langle c_B : \mathsf{any}, s_B : \mathsf{any}, r_B : \mathsf{rand} \rangle \rangle \\
\quad = vs\ \mathsf{in} \\
\quad \mathsf{event}\ \mathsf{Voter}(\mathsf{id}, \mathsf{spk}_{\mathsf{id}}, \mathsf{H}); \\
\quad \mathsf{let}\ b_v = \langle c_A, s_A \rangle\ \mathsf{in} \\
\quad \mathsf{event}\ \mathsf{Voted}(\mathsf{id}, A); \\
\quad \mathsf{out}(c_{vd}, \langle \mathsf{id}, \mathsf{spk}_{\mathsf{id}}, b_v \rangle); \\
\quad \mathsf{in}(c_{vd}, = \mathsf{ok}); \\
\quad \mathsf{get}\ \mathsf{BB}(= \mathsf{spk}_{\mathsf{id}}, b')\ \mathsf{in} \\
\quad \mathsf{event}\ \mathsf{Verified}(\mathsf{id}, A).
\end{array}
$$

Figure 5: Process modelling an honest voter id voting for A

where the M_i and the $M_{i,j}$ do not contain names, if for any closed process Q (representing an attacker) such that $\mathsf{fn}(Q) \subseteq \mathsf{fn}(P_0)$, for any trace tr of $P_0 \mid Q$, for any substitution σ, if for all i, tr executes the event $M_i\sigma$ and for all i, $[\![\phi_i\sigma]\!] = \top$, then there exists i and σ' extending σ such that tr executes event $M_{i,j}\sigma'$, and $[\![\phi_{i,j}\sigma']\!] = \top$ for all j.

For simplicity, we may *e.g.* write $M \rightsquigarrow M_1 \wedge M_2$ instead of $\mathsf{event}(M) \rightsquigarrow \mathsf{event}(M_1) \wedge \mathsf{event}(M_2)$.

Examples can be found in Section 3.6.

3.3.2. Equivalence. Intuitively, two processes are in equivalence if an adversary cannot distinguish between them. More precisely, whenever P may emit on some channel c (interacting with a process R modelling the adversary), then Q can also emit on c. We write $\mathcal{C} \downarrow_N$ when a configuration $\mathcal{C} = E, \mathcal{P}$ with $E = (\mathcal{N}_{\mathsf{pub}}, \mathcal{N}_{\mathsf{priv}})$ can output on some channel N, *i.e.* if there exists $\mathsf{out}(N, M); P \in \mathcal{P}$ such that $\mathsf{fn}(N) \subseteq \mathcal{N}_{\mathsf{pub}}$. An adversarial context $C[_]$ is a process of the form $\mathsf{new}\, n : \mathsf{bitstring}; _ \mid Q$ where $\mathsf{fv}(Q) = \emptyset$ and _ is a "hole". The application of the context C to a configuration $\mathcal{C} = (\mathcal{N}_{\mathsf{pub}}, \mathcal{N}_{\mathsf{priv}}), \mathcal{P}$ is defined as follows, assuming $\mathcal{N}_{\mathsf{priv}} \cap \mathsf{fn}(Q) = \emptyset$.

$$
\begin{array}{rrcl}
 & C[\mathcal{C}] & = & (\mathcal{N}'_{\mathsf{pub}}, \mathcal{N}'_{\mathsf{priv}}), \mathcal{P} \cup \{Q\} \\
\text{with} & \mathcal{N}'_{\mathsf{pub}} & = & (\mathcal{N}_{\mathsf{pub}} \cup \mathsf{fn}(Q)) \backslash \{n\} \\
\text{and} & \mathcal{N}'_{\mathsf{priv}} & = & \mathcal{N}_{\mathsf{priv}} \cup \{n\}
\end{array}
$$

We are now ready to define equivalence.

Definition 2 (Observational equivalence [23])**.** Observational equivalence*, denoted by $\approx$, is the largest symmetric relation between configurations such that $\mathcal{C} \approx \mathcal{C}'$ implies:*

- *if $\mathcal{C} \downarrow_N$ then $\exists \mathcal{C}'_1.\ \mathcal{C}' \to^* \mathcal{C}'_1 \ \wedge\ \mathcal{C}'_1 \downarrow_N$;*
- *if $\mathcal{C} \to \mathcal{C}_1$, then $\exists \mathcal{C}'_1.\ \mathcal{C}' \to^* \mathcal{C}'_1 \ \wedge\ \mathcal{C}_1 \approx \mathcal{C}'_1$;*
- *$C[\mathcal{C}] \approx C[\mathcal{C}']$, for any adversarial context $C[_]$.*

3.4. Privacy

The privacy property for a voting protocol formalises the fact that an attacker should be unable to know which voter voted for which candidate. This is expressed as an equivalence property [30]: an attacker should not be able to distinguish between a scenario where Alice votes for candidate 0 and Bob for 1, and a scenario where Alice votes for 1 and Bob for 0. Formally, the privacy property is written

$$\mathcal{C}[V_A(0)][V_B(1)] \ \approx\ \mathcal{C}[V_A(1)][V_B(0)]$$

where $V_{\mathsf{id}}(v)$ is the process modelling voter id voting v, and $\mathcal{C}$ is a context modelling the rest of the voting protocol.

3.5. Receipt-freeness

Receipt-freeness models the guarantee that a voter Alice is not able to prove to an attacker for which candidate she voted. Intuitively, this prevents her from selling her vote, for

instance, as she cannot provide any guarantee that her actual vote corresponds to the one she sold.

Delaune *et al.* [30] propose a formal definition of receipt-freeness as an equivalence property, similarly to privacy. The fact that Alice tries to prove who she voted for is modelled as a modified version ${V_A}^{\mathsf{ch}}$ of her voting process V_A, in which she reveals all her knowledge to the attacker. That is, she outputs to the attacker on a reserved channel ch any message she receives (even on private channels), and any fresh nonce she generates. The formal transformation is provided in Appendix C.

Conversely, we define the transformation mapping a process P to $P^{\setminus\mathsf{ch}} = \mathsf{new\,ch} : \mathsf{channel}.(P \mid !\mathsf{in}(\mathsf{ch}, x : \mathsf{any}))$, which is similar to P, except that, intuitively, the outputs on channel ch are ignored.

Then, a voting protocol is *receipt-free* if there exists a process V' such that $V'^{\setminus\mathsf{ch}} \approx V_A(1)$ and

$$\mathcal{C}[(V_A(0))^{\mathsf{ch}}][V_B(1)] \;\approx\; \mathcal{C}[V'][V_B(0)]$$

where, as before, $V_{\mathsf{id}}(v)$ is the process modelling voter id voting for v, and $\mathcal{C}$ is a context with two "holes" representing the rest of the protocol.

Intuitively, V' represents how Alice should behave when she pretends that she votes for 0 while she is actually voting for 1. In particular, with $V' = (V_A(1))^{\mathsf{ch}}$, if

$$\mathcal{C}[(V_A(0))^{\mathsf{ch}}][V_B(1)] \;\approx\; \mathcal{C}[(V_A(1))^{\mathsf{ch}}][V_B(0)].$$

then the protocol is receipt-free even if Alice provides the attacker with all her data. In that case, the voter does not need to fake any receipt.

3.6. Verifiability

Verifiability models the fact that the voters should be able to verify that their vote is correctly taken into account. More precisely, the result of the election should contain

- the votes from the voters who successfully performed the verifications specified by the protocol;
- some of the votes from the voters who did not make any verification (such votes may typically be dropped by the attacker);
- and at most k dishonest votes where k is the number of dishonest voters, that is, under the control of the attacker.

In practice, it is likely that many voters will drop the verification steps, while the attacker only fully controls a few - dishonest - voters. Therefore, it is important to control what can happen to votes from voters that do not, we require here that they may be dropped but not modified, which prevents ballot stuffing. For example, Helios does not satisfy this definition against a dishonest voting server since the voting server may add any votes for absentee voters. This is not the case for schemes like Belenios, Civitas, or Selene, to cite a few.

We respectively denote $\mathcal{A}$, $\mathcal{V}$, and $\mathcal{C}$ the sets of *voter identities*, *admissible votes*, and *credentials*. $\mathcal{A}$, $\mathcal{V}$ and $\mathcal{C}$ are sets of messages.

Following previous approaches [31], [32], we formally define verifiability using events, that record the progress of the protocol for each voter.

- $\mathsf{Voter}(\mathsf{id}, \mathsf{cred}, l)$: voter id is registered with credential (*i.e.* signing key) cred. The label l (either H or D) records whether the voter is honest or dishonest. Typically, a voter will be labelled as dishonest if her corresponding voting material is provided to the adversary, and labelled as honest otherwise.
- $\mathsf{Voted}(\mathsf{id}, v)$: voter id has cast a vote for v.
- $\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b)$: ballot b has been recorded on the bulletin board by the voting server. According to the voting server, b is associated with voter id with credential cred.
- $\mathsf{Verified}(\mathsf{id}, v)$: voter id has voted for v and has performed all required checks. She should be guaranteed that v will be properly counted.

We denote ρ the *counting function*, *i.e.* the function that, given a multiset of votes, computes the result of the election. Typical examples would be the winner function, that outputs the candidate with the majority of votes, or the multiset function, that simply outputs the multiset of votes itself.

We also assume a predicate valid on terms, *i.e.* a mapping from terms to $\{\top, \bot\}$. $\mathsf{valid}(b)$ is intended to represent the public checks that may be defined by the protocol to audit a ballot b published on the bulletin board, and ensure it is well-formed. In case the protocol provides no such mechanism, $\mathsf{valid}(b) = \top$ for any b.

We introduce some additional vocabulary. Given a trace t, we denote by $\mathsf{HV}(t)$ (resp. $\mathsf{DV}(t)$) the set of honest (resp. dishonest) voters that appear in the trace, that is, the set of $\mathsf{id}'s$ for which an event of the form $\mathsf{Voter}(\mathsf{id}, *, \mathsf{H})$ (resp. $\mathsf{Voter}(\mathsf{id}, *, \mathsf{D})$) occurs in the trace t. The set of voters is simply $\mathsf{Voters}(t) = \mathsf{HV}(t) \cup \mathsf{DV}(t)$. Then $\mathsf{CHV}(t)$ is the subset of honest voters who have verified their vote, *i.e.* for which there is also an event of the form $\mathsf{Verified}(\mathsf{id}, *)$ in t. The multiset of votes corresponding to voters who verified is

$$\mathcal{V}_{\mathsf{CHV}}(t) = \{\!\!\{v \in \mathcal{V} \mid \exists \mathsf{id} \in \mathsf{CHV}(t).\ \mathsf{Voted}(\mathsf{id}, v) \in t\}\!\!\}$$

The multiset of votes $\mathcal{V}_{\overline{\mathsf{CHV}}}(t)$ corresponding to voters who did not verify, that is voters in $\mathsf{HV}(t) \setminus \mathsf{CHV}(t)$, is defined similarly. The multiset of ballots ready to be tallied, that is, ballots b occurring in an event $\mathsf{Going\text{-}to\text{-}tally}$ and such that $\mathsf{valid}(b)$ holds, is denoted by $\mathsf{BB}(t)$. Finally, the result of the election can be computed from $\mathsf{BB}(t)$:

$$\mathsf{result}(t) = \rho(\{\!\!\{\mathsf{open}(b), b \in \mathsf{BB}(t)\}\!\!\}).$$

where open is a function that retrieves the vote contained in a ballot. This function is typically the decryption function but its exact definition depends on the protocol. It corresponds to the Extract function considered in *e.g.* [33].

Then a voting system, modelled by a configuration $\mathcal{C}$ is *verifiable* if, for any trace t of $\mathcal{C}$, the result of the election accounts for all the votes from honest voters who verified their votes, a subset of the votes of the other honest voters,

and at most as many additional votes as there are dishonest voters. Formally:

$$\exists V \subseteq_{\mathsf{m}} \mathcal{V}_{\overline{\mathsf{CHV}}}(t).\ \exists V_c \subseteq_{\mathsf{m}} \mathcal{V}.\ |V_c| \leq |\mathsf{DV}(t)| \ \wedge\ \mathsf{result}(t) = \rho(\mathcal{V}_{\mathsf{CHV}}(t) \uplus V \uplus V_c)$$

The multiset $\mathcal{V}_{\overline{\mathsf{CHV}}}(t)$ is the multiset of votes from honest voters who verified, while V is some subset of votes from the other honest voters and V_c represents dishonest votes.

This property guarantees that the votes of the voters who performed the verifications are indeed counted. Votes from the other honest voters may be discarded, but not changed. It also ensures that there is no ballot stuffing: the attacker may only add as many votes as there are registered dishonest voters, and these votes must be for admissible candidates.

4. Sufficient conditions for verifiability

Verifiability as defined in the previous 3.6 cannot be directly expressed in ProVerif. Indeed, it requires counting the votes to compare the size of multisets, which is typically out of reach for ProVerif.

To circumvent this issue, we identify sufficient conditions for verifiability, expressed as correspondence properties, which ProVerif is able to prove. We show that these conditions indeed entail verifiability. This contribution is of independent interest and could be reused as a proof technique for analysing verifiability of other voting protocols. This approach was initiated in [25] in the context of typing systems (which cannot count easily either). Our result extends [25] to a stronger verifiability notion (presented in Section 3.6) that accounts for ballot stuffing and for voters who may not perform all the verification tests.

4.1. Assumptions

We need to make a few assumptions on the voting processes that we consider. Namely, we assume that:

1) a voter cannot be labelled as both honest and dishonest;
2) if a voter checks that she has voted for v then she indeed intended to vote for v (and not for another candidate);
3) no revote: voters may attempt to vote several times but only the first vote will be counted.

Formally, we assume the following properties hold for any trace t.

1) $\forall \mathsf{id} \in \mathsf{HV}(t).\ \mathsf{id} \notin \mathsf{DV}(t)$;
2) $\forall \mathsf{id} \in \mathsf{HV}(t).\ \mathsf{Verified}(\mathsf{id}, v) \in t \Rightarrow \mathsf{Voted}(\mathsf{id}, v) \in t$;
3) $\forall \mathsf{id} \in \mathsf{HV}(t).\ |\{\!|\mathsf{Voted}(\mathsf{id}, v) \in t | v \in \mathcal{V}|\!\}| \leq 1$.

The first two assumptions are trivially satisfied by any process modelling a voting scheme. The last one is a simplifying assumption, that avoids to spell out a revote policy. We believe that our results can be easily extended to the case of revoting.

In addition, we assume that no duplicate ballots appear on the bulletin board:

$$\forall b.\ |\{\!|\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \mid \mathsf{id} \in \mathcal{A}, \mathsf{cred} \in \mathcal{C}|\!\}| \leq 1$$

This assumption is satisfied by most voting systems and can be easily enforced by an external audit.

4.2. Sufficient conditions for verifiability

We characterise verifiability through three well-known properties: recorded-as-intended, eligibility, and individual verifiability. These properties are standard desired properties in the context of voting. Intuitively, recorded-as-intended means that any ballot recorded on the bulletin board in the name of a honest voter corresponds to her intended vote. Eligibility states that any ballot present on the bulletin board is cast in the name of a registered voter and corresponds to a valid vote. Individual verifiability guarantees that once a honest voter Alice has successfully performed the verification step specified by the protocol, there indeed exists a ballot containing her vote on the bulletin board.

We formally define these three properties in the rest of this section. They all require to refer to voters. A voter may be identified either with her identity or with her credential (*e.g.* her signature verification key). In case some election authorities are corrupted, one of these two identifying data (if not both) may be unreliable (that is, controlled by the attacker). Therefore, we consider two variants of recorded-as-intended, eligibility, and individual verifiability; one variant based on identities and the other one based on credentials.

4.2.1. Identity based verifiability. Recorded-as-intended. For any ballot b registered in the name of an honest voter id, voter id must have voted for v, where v is the content of b.

$$\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \ \wedge\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}', \mathsf{H}) \rightsquigarrow \mathsf{Voted}(\mathsf{id}, v) \ \wedge\ v = \mathsf{open}(b).$$

Eligibility. Any ballot b registered in the name of a voter corresponds to a valid voter and a valid vote.

$$\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \ \wedge\ \mathsf{valid}(b) \rightsquigarrow \mathsf{Voter}(\mathsf{id}, \mathsf{cred}', l) \ \wedge\ \mathsf{open}(b) \in \mathcal{V}.$$

Individual verifiability. When a voter id successfully verifies that her vote v is counted, then there is a valid ballot b, registered for id, that contains v.

$$\mathsf{Verified}(\mathsf{id}, v) \rightsquigarrow \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b).$$

These three properties imply verifiability as soon as no two distinct ballots have been registered in the name of the same id, *i.e.*

$$\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \ \wedge \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}', b') \rightsquigarrow b = b'.$$

This property, called *consistent identities*, may be enforced by auditing when the correspondance between identities and ballots is public. In BeleniosVS, this property holds as soon as the voting server is honest.

Theorem 1. *Consider a process P that satisfies the assumptions stated in the beginning of section 4. If P guarantees recorded-as-intended, eligibility, individual verifiability (on identities), and consistent identities, then P is verifiable.*

Proof. We give here the main ideas of the proof of this theorem. The fully detailed proofs can be found in Appendix E.

Consider a trace t of P. By definition, each ballot $b \in \mathsf{BB}(t)$ is registered in the name of some voter id, *i.e.* t contains an event $\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b)$ for some cred. By the eligibility property, id is a registered voter that participates in the election, *i.e.* t also contains an event $\mathsf{Voter}(\mathsf{id}, \mathsf{cred}', l)$ for some cred', l. In addition, by assumption (Section 4.1), $\mathsf{BB}(t)$ contains no duplicate ballots. Together with the consistent identities property, this implies that each ballot $b \in \mathsf{BB}(t)$ occurs exactly once in $\mathsf{BB}(t)$, and is registered with a different id.

Let then S_{CHV}, $S_{\overline{\mathsf{CHV}}}$, and S_{DV} respectively denote the multisets of the votes contained in the ballots from $\mathsf{BB}(t)$ registered with ids in $\mathsf{CHV}(t)$, $\mathsf{HV}(t) \backslash \mathsf{CHV}(t)$, and $\mathsf{DV}(t)$. Together they form a partition of the votes to be tallied:

$$\mathsf{result}(t) = \rho(S_{\mathsf{CHV}} \uplus S_{\overline{\mathsf{CHV}}} \uplus S_{\mathsf{DV}}).$$

Since $\mathsf{BB}(t)$ contains no duplicates, and all ballots are registered with different ids, $|S_{\mathsf{DV}}| \leq |\mathsf{DV}(t)|$. In addition, by the eligibility property, all ballots on the board contain valid votes: thus $S_{\mathsf{DV}} \subseteq \mathcal{V}$.

Now consider the ballots in $\mathsf{BB}(t)$ that are registered in the name of voters who do not check. The recorded-as-intended property states that such a ballot b, registered for id, contains the intended vote of id. This means intuitively that the ballots in $\mathsf{BB}(t)$ registered for ids in $\mathsf{HV}(t) \backslash \mathsf{CHV}(t)$ are a subset of the ballots these voters intended to cast. Following this idea, we show formally that $S_{\overline{\mathsf{CHV}}}$ is a submultiset of the intended votes $\mathcal{V}_{\overline{\mathsf{CHV}}}(t)$ of the voters who do not check.

Similarly, we use the recorded-as-intended property to show that the ballots registered for voters who check their votes form a subset of the ballots these voters intended to cast. Formally, $S_{\mathsf{CHV}} \subseteq_{\mathsf{m}} \mathcal{V}_{\mathsf{CHV}}(t)$. Finally, the individual verifiability property states that for any voter $\mathsf{id} \in \mathsf{CHV}(t)$, $\mathsf{BB}(t)$ does in fact contain a ballot registered for id. This means that all the ballots of the voters who check have been registered for the right id. Formally, we prove that $\mathcal{V}_{\mathsf{CHV}}(t) \subseteq_{\mathsf{m}} S_{\mathsf{CHV}}$. Thus, $S_{\mathsf{CHV}} = \mathcal{V}_{\mathsf{CHV}}(t)$

Therefore, combining the previous observations,

$$\mathsf{result}(t) = \rho(\mathcal{V}_{\mathsf{CHV}}(t) \uplus S_{\overline{\mathsf{CHV}}} \uplus S_{\mathsf{DV}})$$

with $S_{\overline{\mathsf{CHV}}} \subseteq_{\mathsf{m}} \mathcal{V}_{\overline{\mathsf{CHV}}}(t)$, $S_{\mathsf{DV}} \subseteq \mathcal{V}$, and $|S_{\mathsf{DV}}| \leq |\mathsf{DV}(t)|$, which proves verifiability. □

4.2.2. Credential based verifiability.

5. Sufficient conditions for verifiability: credential-based case

We state here the properties forming a sufficient condition for verifiability based on the voter's credential.

Recorded-as-intended (on credentials).

$$\begin{array}{l}\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \wedge \\ \mathsf{Voter}(\mathsf{id}', \mathsf{cred}, \mathsf{H}) \rightsquigarrow \\ \mathsf{Voted}(\mathsf{id}', v) \ \wedge \ v = \mathsf{open}(b).\end{array}$$

Eligibility (on credentials).

$$\begin{array}{l}\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \ \wedge \ \mathsf{valid}(b) \rightsquigarrow \\ \mathsf{Voter}(\mathsf{id}', \mathsf{cred}, l) \ \wedge \ \mathsf{open}(b) \in \mathcal{V}.\end{array}$$

Individual verifiability (on credentials).

$$\begin{array}{l}\mathsf{Verified}(\mathsf{id}, v) \rightsquigarrow \\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H}) \wedge \\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \ \wedge \ \mathsf{valid}(b) \ \wedge \ v = \mathsf{open}(b).\end{array}$$

They are analogous to the identity case except for individual verifiability that further links the voter with her credential.

As for the identity case, these three properties imply verifiability as soon as no two distinct ballots have been registered in the name of the same cred, *i.e.*

$$\begin{array}{l}\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \wedge \\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b') \rightsquigarrow \\ b = b'.\end{array}$$

Moreover the credentials distribution must be consistent: no two identities should be associated with the same credential, and conversely.

$$\begin{array}{l}\mathsf{Voter}(\mathsf{id}, \mathsf{cred}, l) \ \wedge \ \mathsf{Voter}(\mathsf{id}', \mathsf{cred}', l') \rightsquigarrow \\ (\mathsf{id} = \mathsf{id}' \ \wedge \ \mathsf{cred} = \mathsf{cred}') \ \vee \ (\mathsf{id} \neq \mathsf{id}' \ \wedge \ \mathsf{cred} \neq \mathsf{cred}').\end{array}$$

These two properties are called *consistent credentials*. The first one may typically be enforced by external auditing (this is the case in our protocol BeleniosVS). The second property typically holds as soon as the authority in charge of distributing the credentials is honest.

Theorem 2. *Consider a process P that satisfies the assumptions stated in the beginning of section 4. If P guarantees recorded-as-intended, eligibility, individual verifiability (on credentials), and consistent credentials, then P is verifiable.*

The proof for this theorem is similar to the one for Theorem 1, and can be found in appendix.

6. Analysis

Using the ProVerif tool, we prove that BeleniosVS is both verifiable and private, and even receipt-free against an external adversary.

6.1. Formal model

We present our ProVerif model of the BeleniosVS protocol and we discuss our modelling choices. The corresponding ProVerif files can be found at [34]. At the end of this section, we discuss alternative choices of tools for conducting the security analysis.

Equational theory. The constructors, destructors, and rewrite rules representing randomisable ciphertexts and signatures have been presented in Example 1. However, ProVerif is unable to handle the rule for randomisation for two reasons. First, ProVerif cannot handle associative and commutative (AC) operators. Second, the re-randomisation rule itself (simplified) is as follows:

$$\mathsf{rand}(\mathsf{aenc}(m, pk, r), r') \rightarrow \mathsf{aenc}(m, pk, r{+}r')$$

Even if $+$ is not defined as AC, ProVerif runs into termination issues, building ever growing terms. Therefore, we over-approximate the re-randomisation functionality. Intuitively, encryption and signature now use two random nonces (r, r') instead of one. Instead of modelling the addition operation, we model the randomisation by a nonce r'' as replacing the second nonce r' with r''. This yields the following (simplified) rule.

$$\mathsf{rand}(\mathsf{aenc}(m, pk, r, r'), r") \rightarrow \mathsf{aenc}(m, pk, r, r")$$

More precisely, we consider the set $\mathcal{C}'_{\mathsf{rand}}$ of constructors, which is obtained from $\mathcal{C}_{\mathsf{rand}}$ (from Example 1) by removing the $+$ constructor, and modifying the aenc and sign constructors so that they take an additional argument, of type rand. The destructors from $\mathcal{D}_{\mathsf{rand}}$ are unchanged. We also consider the set of rewrite rules $\mathcal{R}'_{\mathsf{rand}}$, which is obtained from $\mathcal{R}_{\mathsf{rand}}$ by adapting the rules to account for the additional argument and replacing the rule for randomising with the following rule:

$$\begin{aligned}
&\mathsf{rand}(\mathsf{aenc}(x, \mathsf{pk}(y), \mathsf{spk}(z), r, r'),\\
&\quad \mathsf{sign}(\mathsf{aenc}(x, \mathsf{pk}(y), \mathsf{spk}(z), r, r'), \mathsf{pk}(y), z, s, s'),\\
&\quad \mathsf{pk}(y), \mathsf{spk}(z), r'', s'')\\
&\rightarrow \langle \mathsf{aenc}(x, \mathsf{pk}(y), \mathsf{spk}(z), r, r''),\\
&\quad \mathsf{sign}(\mathsf{aenc}(x, \mathsf{pk}(y), \mathsf{spk}(z), r, r''), \mathsf{pk}(y), z, s, s'')\rangle.
\end{aligned}$$

This model of randomisation is an overapproximation of the actual behaviour of the primitives, that intuitively gives more power to the attacker. Indeed an attacker who knows r' can now recompute the original ciphertext $\mathsf{aenc}(x, \mathsf{pk}(y), \mathsf{spk}(z), r, r')$ from the randomised ciphertext $\mathsf{aenc}(x, \mathsf{pk}(y), \mathsf{spk}(z), r, r'')$, and similarly for the signatures, which is not possible with the actual primitive.

Modelling the protocol. We model the different entities of BeleniosVS as individual ProVerif processes running in parallel, communicating through channels that can be public or private. The public bulletin board is modelled as a table BB, containing entries of the form (spk, b). Such an entry denotes that the voting server has recorded a ballot b in the name of a voter with verification key spk.

Voter process. An honest voter voting for candidate v is modelled by the process $\mathsf{Voter}(\mathsf{id}, v, c_{vs}, c_{vd})$ presented in Example 2. This process corresponds to the case of a voter who does not audit the voting sheet.

The optional audit phase is modelled by adding tests verifying the signatures and ciphertexts on the voting sheet, before the voter scans a ballot with the voting device.

Voting device process. The voting device is modelled by the process displayed in Figure 6. This process uses a public channel, and shares private channels c_{vd} and $c_s(\mathsf{id})$ with resp. the voter and the voting server. The private channel c_{vd} models the fact that the voter can securely enter her data to her voting device. The private channel $c_s(\mathsf{id})$ models an authenticated channel between the voting device and the voting server. Any emission on $c_s(\mathsf{id})$ is immediately preceded by an emission on a public channel. Of course, these channels may become public when the corresponding entities are considered to be corrupted.

$$\begin{aligned}
&\mathsf{VD}(c_{pub}, c_{vd}, c_s(\mathsf{id})) =\\
&\quad \mathsf{in}(c_{vd}, \langle \mathsf{id} : \mathsf{any}, \mathsf{spk_{id}} : \mathsf{spkey}, b : \mathsf{any}\rangle);\\
&\quad \mathsf{new}\, r : \mathsf{rand};\\
&\quad \mathsf{new}\, t : \mathsf{rand};\\
&\quad \mathsf{let}\ b' : \mathsf{any} = \mathsf{rand}(b, \mathsf{pk_e}, \mathsf{spk_{id}}, r, t)\ \mathsf{in}\\
&\quad \mathsf{out}(c_{pub}, \langle \mathsf{id}, \mathsf{spk_{id}}, b'\rangle);\\
&\quad \mathsf{out}(c_s(\mathsf{id}), \langle \mathsf{id}, \mathsf{spk_{id}}, b', \rangle);\\
&\quad \mathsf{in}(c_s(\mathsf{id}), = \mathsf{ok});\\
&\quad \mathsf{out}(c_{vd}, \mathsf{ok}).
\end{aligned}$$

Figure 6: Process modelling an honest voting device.

$$\begin{aligned}
&\mathsf{S}(c_{\mathsf{pub}}, c) =\\
&\quad \mathsf{in}(c, \langle \mathsf{id} : \mathsf{any}, \mathsf{spk_{id}} : \mathsf{spkey}, b' : \mathsf{any}\rangle);\\
&\quad \mathsf{if}\ (c = c_s(\mathsf{id}))\\
&\qquad \&\&\ \mathsf{verify}(b', \mathsf{spk_{id}}) = \mathtt{true}\\
&\quad \mathsf{then}\\
&\quad \mathsf{new}\, r : \mathsf{rand};\\
&\quad \mathsf{new}\, t : \mathsf{rand};\\
&\quad \mathsf{let}\ b'' : \mathsf{any} = \mathsf{rand}(b', \mathsf{pk_e}, \mathsf{spk_{id}}, r, t)\ \mathsf{in}\\
&\quad \mathsf{event}(\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{spk_{id}}, b''));\\
&\quad \mathsf{insert}\ \mathsf{BB}(\mathsf{spk_{id}}, b'');\\
&\quad \mathsf{out}(c_{pub}, \langle \mathsf{spk_{id}}, b''\rangle);\\
&\quad \mathsf{out}(c, \mathsf{ok}).
\end{aligned}$$

Figure 7: Process modelling the voting server.

The voting device first receives the voter's id, verification key $\mathsf{spk_{id}}$ and ballot b, then randomises the ballot yielding b', and sends id, $\mathsf{spk_{id}}$, and b' to the voting server.

Voting server process. The process $\mathsf{S}(c_{\mathsf{pub}}, c)$, depicted in Figure 7, models one instance of the voting server, handling the ballot of one voter id. This process shares the private channel $c_s(\mathsf{id})$ with the voting device. It first receives the voter's identity id, her verification key and her ballot from the voting device. It then checks that the voter's claimed identity corresponds to the identity of the voter owning the private channel, *i.e.* that $c = c_s(\mathsf{id})$. This models the fact that the voter is correctly authenticated, that is, the server only accepts a ballot cast under identity id on the private channel associated with id, *i.e.* $c_s(\mathsf{id})$.

For clarity, we omit from the presentation that the voting server also verifies that the received verification key is indeed the key corresponding to id.

The server then checks that the ballot is correctly signed with the received key. Finally, if these checks succeed, the server randomises the ballot before adding it, together with the voter's verification key, to the board BB; and sends confirmation to the voting device.

Tallying process. We consider a simple model of the tallying phase. We represent it as a process that retrieves the ballots signed by honest voters from the board BB, shuffles them and decrypts them in a non-deterministic order, and outputs the plaintexts. To simplify the analysis (*w.r.t.* ProVerif), the mixnet is only modelled for the ballots from honest

voters: for the dishonest ballots, the tallying process simply retrieves them from the board, decrypts them and outputs the corresponding plaintexts without mixing them. Note that this simplification may only yield more attacks since ProVerif now has to prove equivalence even if the order of dishonest ballots cannot be changed. The resulting process Tally can be found in [34].

Registrar process The instance of the registrar that generates the voting sheet for voter id is modelled by a process $\mathsf{Registrar}(\mathsf{id}, c_{vs})$, fully specified in [34], where c_{vs} is a private channel shared with the voter process $\mathsf{Voter}(\mathsf{id}, v, c_{vs}, c_{vd})$. This process generates a fresh signing key $\mathsf{ssk}_{\mathsf{id}}$ for voter id, uses it to generate a voting sheet, and sends this voting sheet to the voter on channel c_{vs}. It also sends to the voting server the list verification keys, with their association with each voter. To simplify the analysis, we simply publish this list. However, for everlasting privacy (not studied in this paper), this list should be kept private.

Complete election process. The model of the whole protocol is obtained by putting in parallel unbounded numbers of honest voters for each candidate, and an unbounded number of instances of the registrar and voting server processes, to be used by dishonest voters played by the attacker. Each of the honest voter processes has its own channels and instances of the voting device, registrar, and voting server processes.

Choice of ProVerif. Our main motivation for using ProVerif stems from the fact that ProVerif is a mature push-button tool that offers a lot of flexibility in terms of equational theories and modelling choices for protocols. Another tool that offers a lot of flexibility is Tamarin [35] that provides a native support for the AC theory. Given the complexity of our protocol, it is likely however that Tamarin would have required some manual guidance through ad-hoc lemmas. Another difficulty of our model is again the equation $\mathsf{rand}(\mathsf{aenc}(m, pk, r), r') \rightarrow \mathsf{aenc}(m, pk, r{+}r')$ that may not be well supported by the tool.

6.2. Threat model

Several entities taking part in the protocol may be dishonest, namely the dishonest registrars, the voting server, voting devices, auditing devices, and voters. The goal of our security analysis is to consider any combination of honesty/dishonesty status for each entity and identify in which cases our protocol is secure. We also consider the case where honest voters inadvertently lose their voting credentials (voting sheet and/or password). In addition, we consider the case where the election secret key sk_{e} is compromised.

Our analysis shows that our protocol is robust to the corruption of any component provided that sufficiently many entities remain honest. In other words, there is no single point of failure.

Dishonest participants. The attacker fully controls dishonest parties, which is simply modelled by letting the attacker know their corresponding secrets and communication channels. The only exception is the dishonest voting server, that requires some more care. A dishonest voting server is modelled as a process that accepts any message from the attacker, and adds it to the bulletin board. Since the board is public and can be audited by anyone, we assume that a dishonest voting server may only display messages signed by eligible signature keys on the board. Indeed, invalidly signed ballots on the board would be detected, and the election would be called off.

In addition, we also consider voters who do not audit their sheets: the voter simply always believe the voting sheet is well-formed.

Leaked data. Independently from the honesty or dishonesty status of the protocol participants, we consider that voters may inadvertently lose or reveal their voting sheet or their password, letting the attacker learn them. This is modelled as the voter process outputting the leaked data on a public channel. Finally, we also consider that the attacker may learn the election secret key sk_{e}, which is modelled by outputting this key on a public channel at the beginning of the election process. Typically, when the attacker learns the election secret key, privacy is entirely lost, but verifiability may still be achievable.

6.3. Results

We analyse verifiability, privacy, and receipt-freeness, under all possible scenarios, in terms of corruption status of the entities and data, as listed in Section 6.2. We however restrict the analysis to the cases where the voters' voting devices are either all honest or all dishonest, either all voters reveal their voting sheet or none does, and similarly for the auditing devices and passwords. This yields 96 corruption scenarios. In addition, we also consider the case where all authorities in the election, *i.e.* the registrar and the voting server, are honest but some of the voters reveal their voting sheet and/or audit it with a dishonest auditing device, while other voters reveal their password to the attacker and/or use a dishonest voting device.

Verifiability. We study the verifiability of our protocol with an unbounded number of honest and dishonest voters voting for each candidate. To establish verifiability, we use the properties stated in Section 4, relying either on Theorem 1 or its variant corresponding to a dishonest voting server.

Privacy and Receipt-freeness. To prove receipt-freeness, we make the standard assumption that the attacker does not see the actual ballot sent to the server by the voter. As explained in Section 3.5, the protocol is receipt-free if an attacker cannot distinguish between a voter that provides him with all her secrets from a voter that provides fake data. This requires that the attacker does not monitor when a voter actually votes.

A notable modelling issue in the study of privacy and receipt-freeness is that, in some cases, the overapproximation made in the equational theory described in Section 6.1 was too coarse. It caused ProVerif to find false attacks, due to the additional power given to the attacker. In these cases, we adapted the model by replacing both random values r and r' with fresh nonces, instead of only r' (when applying

the rewrite rule). This adaptation was only used in cases where the randomisation is performed by an honest entity, and models the fact that the randomisation is actually perfect and cannot be tampered with by the attacker.

Results of the analysis We analyse verifiability, privacy, and receipt-freeness for all 96 corruption scenarios mentioned previously, with the help of the tool ProVerif. Thanks to the automation of the analysis, we were able to analyse cases that are often left out. For example, we consider scenarios where voters leak their password or their voting sheet. In practice, such scenarios are more likely than a full compromission of the voting client or the voting server. We depict in Figure 8 all the scenarios (22 in total) for which at least one of the property holds, as well as the analysis time in ProVerif, when ProVerif can prove security. For trace properties (*i.e.* verifiability), ProVerif always find a true attack when too many parties are corrupted. For equivalence properties (*i.e.* privacy and receipt-freeness), ProVerif typically outputs "cannot be proved" when too many parties are corrupted. But in all cases marked with a ✗, we found a real attack. We also provide the "minimal" scenarios (11 in total) for which the properties do not hold. Of course, the properties still do not hold when more parties are corrupted. We ran ProVerif on one single core of an Intel Xeon E5 2687W v3 @3.10GHz CPU.

In a nutshell, the protocol is secure in the following cases:

- as soon as the registrar is honest, the protocol is both verifiable and private, provided that the voting sheet is not leaked (by the voter or the auditing device);
- as soon as the voting server is honest and the password is not leaked (which requires the voting device to be honest), then the protocol is both verifiable, private, and receipt-free, provided that the voting sheets are well-formed (ensured *e.g.* by honest auditing).

For privacy and receipt-freeness, we also need to assume the election key to be secret. For verifiability properties, the election key can be given to the attacker.

Real attacks exist in the other cases. In particular, our protocol is not receipt-free against a corrupted voting server. Indeed, a voter could simply provide the server with the randomness used to build her ballot from her voting sheet.

Limitations. In order to conduct our analysis with ProVerif, we had to make some modelling choices that we list and explain here.

Number of candidates. Since the voting sheet is pre-printed prior to the voting phase, we had to set the number of candidates for the election. Specifically, we consider elections with two candidates (A and B). We believe the results still hold for an arbitrary number of candidates but this is not covered by our analysis.

Tally. As in most symbolic analyses, our model for the tally is very abstract: we assume an (honest) tally that simply outputs the decryption of the ballots, in some random order. Intuitively, verifiability holds without having to trust the tally, thanks to the proofs of correct decryption. However, modelling faithfully an homomorphic tally or a mixnet would require to model zero-knowledge proofs that involve an arbitrary number of ballots (as many as the number of voters), which is currently out of reach of ProVerif.

7. Lessons learned

We report here a few noteworthy points that surfaced during our analysis of the BeleniosVS protocol. More specifically, we identified several points in the design of the protocol that are crucial to its security, although they might seem unimportant at first glance.

Role of the auditing device. As intended, auditing the voting sheets allows the voters to check that the ballots were correctly generated, and thus protects them from an attack from a dishonest registrar. Indeed, a dishonest registrar could hand out to a voter Alice a wrongly constructed sheet, where all ballots encode votes for the same candidate for instance. This would of course break verifiability for Alice, as even if she checks that a ballot signed by her key is present on the bulletin board, this ballot could encode a different vote than the one she intended to cast.

It seems natural that the auditing device must check each ballot, and not only the one Alice intends to cast. Indeed, in case the attacker controls the auditing device, Alice's vote would otherwise not remain secret. However, it also appears in our analysis that it is crucial for the security of the protocol that the auditing device checks the entire voting sheet, *including the signatures of the ballots*. It might at first seem sufficient to check that the ciphertexts correspond to the right votes, without verifying the signatures. This check indeed prevents the aforementioned attack. In addition it might seem preferable not to let the auditing device learn the content of the entire voting sheet, so as to give it less power in case it is dishonest. However, ignoring the signature subjects the voters to a different attack from a dishonest registrar, this time against privacy. Indeed, a dishonest registrar could provide Alice with a voting sheet that is entirely valid, except for the signature of the ballot for candidate A. If Alice's auditing device does not verify the signatures, this invalid ballot would not be noticed. The voting server would then accept Alice's ballot only if its signature was valid, *i.e.* if it was not the maliciously constructed ballot for A. By observing whether Alice is able to vote or not, the attacker would then deduce whether she intended to vote for A or not. Having the auditing device also verify the signatures prevents this attack.

Voting server's confirmation to the voter. The confirmation message sent by the voting server to the voter (through the voting device) may also look like an unimportant detail. As it turns out, it is in fact also essential to protect the voters from a dishonest registrar.

Indeed, a dishonest registrar could give to Alice and Bob two voting sheets that are valid but are generated using the same signing key. In that case, the presence on the public board of a ballot signed with Alice's key gives her no guarantee that her vote is counted: this ballot could actually

Dishonest parties and leaked data	Registrar		◆	◆																				◆	◆	◆	◆	◆												
	Voting server				◆	◆	◆	◆	◆	◆	◆	◆												◆					◆	◆									
	Voting device					◆		◆		◆		◆	◆	◆	◆	◆									◆						◆	◆						
	Auditing device								-	-	-	-			-	-	◆	◆		-		-	-			◆	-		◆		◆		◆						
	Voting sheet			◆														◆	◆	◆										◆		◆			◆				
	Password						◆	◆			◆	◆		◆		◆					◆	◆						◆									◆	◆	
Properties	Privacy (assumes secret election key)	✓ 7.9	✓ 18	✓ 18	✓ 7.0	✓ 6.5	✓ 7.0	✓ 6.5	✓ 7.0	✓ 6.5	✓ 7.0	✓ 6.5	✓ 12	✓ 12	✓ 12	✓ 12	✓ 22	✓ 22	✓ 22	✓ 22	✓ 12	✓ 12	✓ 7.9	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	Receipt-freeness (assumes secret election key)	✓ 24	✓ 18	✓ 24	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓ 39	✓ 39	✓ 24	✓ 24	✗	✗	✓ 24	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	Verifiability (even with leaked election key)	✓ 0.5	✓ 0.4	✓ 0.5	✓ 0.3	✓ 0.2	✓ 0.3	✓ 0.2	✓ 0.2	✓ 0.1	✓ 0.2	✓ 0.1	✓ 0.4	✓ 0.4	✓ 0.3	✓ 0.3	✓ 0.6	✓ 0.6	✓ 0.8	✓ 0.6	✓ 0.5	✓ 0.5	✓ 0.4	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

Figure 8: Results of the analysis for each corruption scenario
(◆: dishonest entity or leaked data, -: no audit, ✓/✗: the property holds/does not hold, time in seconds).
For privacy and receipt freeness, the election key is assumed to be secret.

have been cast by Bob, while Alice's ballot would have been blocked by the attacker. In that situation, only the (honest) server's confirmation message to Alice guarantees her that her ballot has been received and published on the board. Moreover, an honest server will check that no two voters are assigned the same signature key.

Association between voters and voting sheets. Finally, it is also important that the registrar transmits to the voting server the association it establishes between the voters and the verification keys corresponding to the signing keys used to generate their voting sheets. It might seem sufficient that the voting server records the list of the verification keys corresponding to each ballot as it receives them, in order to ensure that no key is used twice. However, this would cause an attack against verifiability in the scenario where all authorities are honest, but the voting sheets of some voters, and the passwords of other voters, have been leaked to the attacker. Indeed, consider an election with no dishonest voter, and two honest voters that do not verify their votes: Alice, whose voting sheet has been leaked, and Bob, and whose password has been leaked. The attacker could then intercept and discard Alice's ballot on its way to the voting server, pick a ballot for another candidate C from her sheet, and submit it using Bob's password. If the server does not know which key is supposed to belong to Bob, it has no choice but to accept this ballot, even though it is signed by Alice's key. The bulletin board, and hence the result of the election, would then contain a vote for candidate C. Since neither Alice nor Bob voted for this candidate, and no dishonest voter is involved, this would break the verifiability property. On the other hand, if the voting server knows the association between voters and keys, it can prevent the ballot from Alice's voting sheet from being cast using Bob's password.

8. Conclusion

We have proposed a new voting scheme that achieves both privacy and verifiability against a dishonest voting server and a dishonest voting device. This assumes a honest Registrar, in charge of generating and distributing voting sheets to the voters, The behavior of the Registrar can easily be audited by the voters. This audit is optional and can be delegated to third parties, without compromising privacy. Our scheme further guarantees receipt-freeness thanks to the re-randomisation done by the voting server.

A careful reader may have noticed that privacy and verifiability hold in exactly the same threat models. Some cases are counter intuitive. Consider for example the case where the password and the voting sheet of Alice are leaked. This immediately breaks verifiability since an attacker may cast a ballot in name of Alice, for a candidate of her choice. Yet, there is no direct privacy attack since Alice's voting device is honest. But ballot privacy, as defined in Section 3.4, no longer holds. Indeed, an attacker may learn how Bob voted by replacing Alice's vote by any fix value of her choice. Bob's vote can then be inferred from the result. This is another illustration of a recent observation [36] that shows that current definitions of ballot privacy (be it in computational or in symbolic models) actually imply verifiability. To show that our protocol does satisfy some level of privacy in the scenario where verifiability is lost, we would first need to design new notions of ballot privacy.

Voting schemes do not only need to be verifiable, but also to account for dispute resolution. For example, in BeleniosVS, if a voter complains that her ballot has not been registered while she claimed to have cast a ballot, then she can convince an auditor that indeed no ballot is registered with her signing key on the board. Then the voter may simply vote again (possible after the emission of a new voting sheet). As future work, we plan to conduct a more systematic exploration of dispute resolution in BeleniosVS, possibly enhancing the protocol with additional procedures.

Acknowledgements

We are very grateful to Sylvain Ruhault for his fruitful discussion on early versions of this protocol. We would like to acknowledge the work of Mathieu Turuani and Vincent Cheval whose insights on the ProVerif models were very helpful. We also would like to thank the anonymous reviewers for their insightful suggestions.

This work has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research (grant agreement No 645865-SPOOC).

References

[1] S. Heiberg, P. Laud, and J. Willemson, "The application of I-Voting for Estonian parliamentary elections of 2011," in *VoteID 2011*, ser. LNCS, vol. 7187. Springer, 2012, pp. 208–223.

[2] "iVote system - NSW Electoral Commission," http://www.elections.nsw.gov.au/voting/ivote.

[3] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis, "CHVote system specification," Cryptology ePrint Archive, Report 2017/325, 2017, https://eprint.iacr.org/2017/325.

[4] V. Cortier and B. Smyth, "Attacking and fixing Helios: An analysis of ballot secrecy," *Journal of Computer Security*, vol. 21, no. 1, pp. 89–148, 2013.

[5] R. Küsters, T. Truderung, and A. Vogt, "Clash Attacks on the Verifiability of E-Voting Systems," in *33rd IEEE Symposium on Security and Privacy (S&P'12)*, 2012, pp. 395–409.

[6] S. Wolchok, E. Wustrow, D. Isabel, and J. A. Halderman, "Attacking the Washington, D.C. Internet voting system," in *Financial Cryptography and Data Security (FC'12)*, 2012.

[7] 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 Conference on Computer and Communications Security (CCS'14)*, 2014, pp. 703–715.

[8] O. de Marneffe, O. Pereira, and J. Quisquater, "Electing a university president using open-audit voting: Analysis of real-world use of Helios," in *2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '09*, 2009.

[9] A. T. Sherman, R. A. Fink, R. Carback, and D. Chaum, "Scantegrity III: automatic trustworthy receipts, highlighting over/under votes, and full voter verifiability," in *2011 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '11*, 2011.

[10] P. Ryan, "Prêt à Voter with Paillier encryption," *Mathematical and Computer Modelling*, vol. 48, no. 9–10, pp. 1646–1662, 2008.

[11] S. Bell, J. Benaloh, M. D. Byrne, D. Debeauvoir, B. Eakin, P. Kortum, N. McBurnett, O. Pereira, P. B. Stark, D. S. Wallach, G. Fisher, J. Montoya, M. Parker, and M. Winn, "STAR-Vote: A secure, transparent, auditable, and reliable voting system," in *Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE'13)*, 2013.

[12] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a secure voting system," in *IEEE Symposium on Security and Privacy (S&P'08)*, 2008, pp. 354–368.

[13] B. Adida, "Helios: Web-based open-audit voting," in *17th USENIX Security Symposium (Usenix'08)*, 2008, pp. 335–348.

[14] R. Küsters, J. Müller, E. Scapin, and T. Truderung, "sElect: A lightweight verifiable remote voting system," in *IEEE Computer Security Foundations Symposium (CSF'16)*, 2016, pp. 341–354.

[15] P. Y. A. Ryan, P. B. Roenne, and V. Iovino, "Selene: Voting with transparent verifiability and coercion-mitigation," in *1st Workshop on Secure Voting Systems (VOTING'16)*, 2016, pp. 176–192.

[16] D. Galindo, S. Guasch, and J. Puiggali, "2015 Neuchâtel's cast-as-intended verification mechanism," in *5th International Conference on E-Voting and Identity, (VoteID'15)*, 2015, pp. 3–18.

[17] D. A. Basin, S. Radomirovic, and L. Schmid, "Alethea: A provably secure random sample voting protocol," in *31st IEEE Computer Security Foundations Symposium (CSF'18)*, 2018, pp. 283–297.

[18] N. Chondros, B. Zhang, T. Zacharias, P. Diamantopoulos, S. Maneas, C. Patsonakis, A. Delis, A. Kiayias, and M. Roussopoulos, "D-DEMOS: A distributed, end-to-end verifiable, internet voting system," in *36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016*, 2016, pp. 711–720.

[19] A. Kiayias, T. Zacharias, and B. Zhang, "DEMOS-2: Scalable E2E verifiable elections without random oracles," in *ACM Conference on Computer and Communications Security (CCS'15)*, 2015.

[20] D. Chaum, "Surevote: Technical overview," in *Workshop on Trustworthy Elections (WOTE '01)*, 2001.

[21] P. Chaidos, V. Cortier, G. Fuchsbauer, and D. Galindo, "BeleniosRF: A non-interactive receipt-free electronic voting scheme," in *23rd ACM Conference on Computer and Communications Security (CCS'16)*. ACM, 2016, pp. 1614–1625.

[22] O. Blazy, G. Fuchsbauer, D. Pointcheval, and D. Vergnaud, "Signatures on randomizable ciphertexts," in *14th International Conference on Practice and Theory in Public Key Cryptography (PKC 2011)*, 2011, pp. 403–422.

[23] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and ProVerif," *Foundations and Trends in Privacy and Security*, vol. 1, no. 1–2, pp. 1–135, 2016.

[24] K. Bhargavan, R. Corin, C. Fournet, and E. Zalinescu, "Cryptographically Verified Implementations for TLS," in *15th ACM Conference on Computer and Communications Security (CCS'08)*, 2008, pp. 459–468.

[25] V. Cortier and C. Wiedling, "A formal analysis of the Norwegian E-voting protocol," *Journal of Computer Security*, vol. 25, no. 15777, pp. 21–57, 2017.

[26] B. Blanchet, "Symbolic and computational mechanized verification of the ARINC823 avionic protocols," in *30th IEEE Computer Security Foundations Symposium (CSF'17)*, 2017, pp. 68–82.

[27] V. Cortier, F. Eigner, S. Kremer, M. Maffei, and C. Wiedling, "Type-based verification of electronic voting protocols," in *4th Conference on Principles of Security and Trust (POST'15)*, ser. LNCS, vol. 9036. Springer, 2015, pp. 303–323.

[28] V. Cortier, D. Galindo, S. Glondu, and M. Izabachene, "Distributed ElGamal à la Pedersen - application to Helios," in *Workshop on Privacy in the Electronic Society (WPES 2013)*, 2013.

[29] R. Focardi, F. L. Luccio, and H. A. M. Wahsheh, "Usable cryptographic qr codes," in *IEEE International Conference on Industrial Technology (ICIT 2018)*, 2018.

[30] S. Delaune, S. Kremer, and M. D. Ryan, "Verifying privacy-type properties of electronic voting protocols: A taster," in *Towards Trustworthy Elections – New Directions in Electronic Voting*, ser. LNCS, vol. 6000. Springer, 2010, pp. 289–309.

[31] V. Cortier, D. Galindo, S. Glondu, and M. Izabachene, "Election verifiability for helios under weaker trust assumptions," in *19th European Symposium on Research in Computer Security (ESORICS'14)*, ser. LNCS, vol. 8713. Springer, 2014, pp. 327–344.

[32] V. Cortier, D. Galindo, and M. Turuani, "A formal analysis of the Neuchâtel e-voting protocol," in *3rd IEEE European Symposium on Security and Privacy (EuroSP'18)*, 2018.

[33] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi, "A comprehensive analysis of game-based ballot privacy definitions," in *36th IEEE Symposium on Security and Privacy (S&P'15)*, 2015, pp. 499–516.

[34] [Online]. Available: https://members.loria.fr/JLallemand/beleniosvs/

[35] B. Schmidt, S. Meier, C. Cremers, and D. Basin, "Automated analysis of Diffie-Hellman protocols and advanced security properties," in *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, S. Chong, Ed. IEEE, 2012, pp. 78–94.

[36] V. Cortier and J. Lallemand, "Voting: You can't have privacy without individual verifiability," in *25th ACM Conference on Computer and Communications Security (CCS'18)*. ACM, 2018.

Appendix A.
Overview of the BeleniosVS protocol

Figure 9 displays a more detailed overview of the protocol than Figure 1.

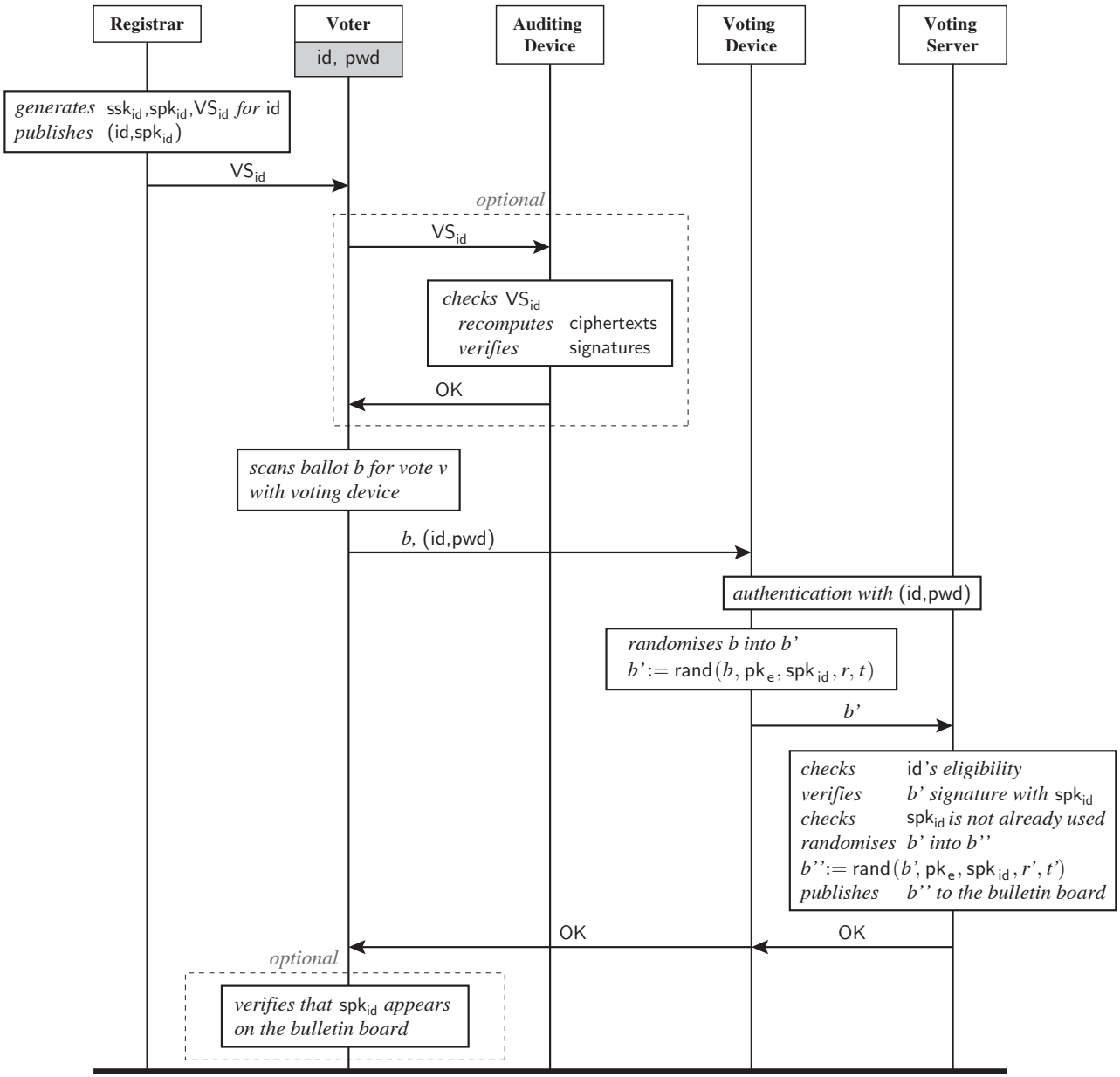

Figure 9: Overview of the BeleniosVS voting process, where $\mathsf{VS_{id}}$ is defined in Figure 2.

Appendix B. ProVerif semantics

$$
\begin{aligned}
& E, \mathcal{P} \cup \{0\}, \mathcal{S} && \rightarrow && E, \mathcal{P}, \mathcal{S} \\
& E, \mathcal{P} \cup \{P \mid Q\}, \mathcal{S} && \rightarrow && E, \mathcal{P} \cup \{P, Q\}, \mathcal{S} \\
& E, \mathcal{P} \cup \{!P\}, \mathcal{S} && \rightarrow && E, \mathcal{P} \cup \{P, !P\}, \mathcal{S} \\
& (\mathcal{N}_{\mathsf{pub}}, \mathcal{N}_{\mathsf{priv}}), \mathcal{P} \cup \{\mathsf{new}\, a : T; P\}, \mathcal{S} && \rightarrow && (\mathcal{N}_{\mathsf{pub}}, \mathcal{N}_{\mathsf{priv}} \cup \{a'\}), \mathcal{P} \cup \{P[a'/a]\}, \mathcal{S} \\
& && && \text{where } a' \notin \mathcal{N}_{\mathsf{pub}} \cup \mathcal{N}_{\mathsf{priv}} \\
& E, \mathcal{P} \cup \{\mathsf{out}(N, M); Q, \mathsf{in}(N, x); P\}, \mathcal{S} && \rightarrow && E, \mathcal{P} \cup \{Q, P[M/x]\}, \mathcal{S} \\
& E, \mathcal{P} \cup \{\mathsf{let}\ x = D\ \mathsf{in}\ P\}, \mathcal{S} && \rightarrow && E, \mathcal{P} \cup \{P[M/x]\}, \mathcal{S} \\
& && && \text{if } D \Downarrow M \text{ and } M \neq \mathsf{fail} \\
& E, \mathcal{P} \cup \{\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\}, \mathcal{S} && \rightarrow && E, \mathcal{P} \cup \{P\}, \mathcal{S} \\
& && && \text{if } [\![\phi]\!] = \top \\
& E, \mathcal{P} \cup \{\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\}, \mathcal{S} && \rightarrow && E, \mathcal{P} \cup \{Q\}, \mathcal{S} \\
& && && \text{if } [\![\phi]\!] = \bot \\
& E, \mathcal{P} \cup \{\mathsf{event}(M); P\}, \mathcal{S} && \rightarrow && E, \mathcal{P} \cup \{P\}, \mathcal{S} \\
& E, \mathcal{P} \cup \{\mathsf{insert}\ \mathsf{Tbl}(M); P\}, \mathcal{S} && \rightarrow && E, \mathcal{P} \cup \{P\}, \mathcal{S}[\mathsf{Tbl} \mapsto \mathcal{S}(\mathsf{Tbl}) \cup \{M\}] \\
& E, \mathcal{P} \cup \{\mathsf{get}\ \mathsf{Tbl}(x)\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ P\ \mathsf{else}\ Q\}, \mathcal{S} && \rightarrow && E, \mathcal{P} \cup \{P[M/x]\}, \mathcal{S} \\
& && && \text{for any } M \in \mathcal{S}(\mathsf{Tbl}) \text{ such that } [\![\phi[M/x]]\!] = \top \\
& E, \mathcal{P} \cup \{\mathsf{get}\ \mathsf{Tbl}(x)\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ P\ \mathsf{else}\ Q\}, \mathcal{S} && \rightarrow && E, \mathcal{P} \cup \{Q\}, \mathcal{S} \\
& && && \text{if } \forall M \in \mathcal{S}(\mathsf{Tbl}).\ [\![\phi[M/x]]\!] = \bot
\end{aligned}
$$

Figure 10: Transitions between configurations.

Figure 10 displays the semantics of ProVerif, defined as a transition relation $\rightarrow$ between configurations.

Appendix C. Receipt-freeness

The transformation mapping a process P to P^{ch} which reveals its knowledge to the attacker (where ch is a name not occurring in P) is formally defined in Figure 11.

Appendix D. Details on the model

The optional audit phase that can be performed by the voter is modelled by adding tests verifying the signatures and ciphertexts on the voting sheet, before the voter scans a ballot with the voting device. We model this by inlining the checks made by the (honest) auditing device in the voter process. The resulting process $\mathsf{VoterAudit}(\mathsf{id}, v, c_{vs}, c_{vd})$, modelling a voter who audits the voting sheet, is depicted in Figure 12.

$$
\begin{aligned}
0^{\mathsf{ch}} &= 0 \\
(\mathsf{out}(N, M); P)^{\mathsf{ch}} &= \mathsf{out}(N, M); P^{\mathsf{ch}} \\
(\mathsf{in}(N, x : T); P)^{\mathsf{ch}} &= \mathsf{in}(N, x : T); \mathsf{out}(\mathsf{ch}, x); P^{\mathsf{ch}} \\
(P \mid Q)^{\mathsf{ch}} &= P^{\mathsf{ch}} \mid Q^{\mathsf{ch}} \\
(!P)^{\mathsf{ch}} &= !P^{\mathsf{ch}} \\
(\mathsf{new}\, a : T; P)^{\mathsf{ch}} &= \mathsf{new}\, a : T; \mathsf{out}(\mathsf{ch}, a); P^{\mathsf{ch}} \\
(\mathsf{let}\ x : T = D\ \mathsf{in}\ P\ \mathsf{else}\ Q)^{\mathsf{ch}} & \\
&= \mathsf{let}\ x : T = D\ \mathsf{in}\ P^{\mathsf{ch}}\ \mathsf{else}\ Q^{\mathsf{ch}} \\
(\mathsf{if}\ \phi\ \mathsf{then}\ P\ \mathsf{else}\ Q)^{\mathsf{ch}} &= \mathsf{if}\ \phi\ \mathsf{then}\ P^{\mathsf{ch}}\ \mathsf{else}\ Q^{\mathsf{ch}} \\
(\mathsf{event}(M); P)^{\mathsf{ch}} &= \mathsf{event}(M); P^{\mathsf{ch}} \\
(\mathsf{insert}\ \mathsf{Tbl}(M); P)^{\mathsf{ch}} &= \mathsf{insert}\ \mathsf{Tbl}(M); P^{\mathsf{ch}} \\
(\mathsf{get}\ \mathsf{Tbl}(x : T)\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ P\ \mathsf{else}\ Q)^{\mathsf{ch}} & \\
&= \mathsf{get}\ \mathsf{Tbl}(x : T)\ \mathsf{suchthat}\ \phi \\
&\quad \mathsf{in}\ (\mathsf{out}(\mathsf{ch}, x); P^{\mathsf{ch}})\ \mathsf{else}\ Q^{\mathsf{ch}}
\end{aligned}
$$

Figure 11: Receipt-freeness transformation.

$$
\begin{aligned}
& \mathsf{VoterAudit}(\mathsf{id}, A, c_{vs}, c_{vd}) = \\
& \quad [\ldots] \\
& \quad \mathsf{event}\ \mathsf{Voter}(\mathsf{id}, \mathsf{spk}_{\mathsf{id}}, \mathsf{H}); \\
& \quad \mathsf{if}\ (\quad c_A = \mathsf{aenc}(A, \mathsf{pk}_{\mathsf{e}}, \mathsf{spk}_{\mathsf{id}}, r_A, r_A) \\
& \qquad \&\&\ c_B = \mathsf{aenc}(B, \mathsf{pk}_{\mathsf{e}}, \mathsf{spk}_{\mathsf{id}}, r_B, r_B) \\
& \qquad \&\&\ \mathsf{verify}(c_A, s_A, \mathsf{spk}_{\mathsf{id}}) = \mathtt{true} \\
& \qquad \&\&\ \mathsf{verify}(c_B, s_B, \mathsf{spk}_{\mathsf{id}}) = \mathtt{true}) \\
& \quad \mathsf{then} \\
& \quad \mathsf{let}\ b_v = \langle c_A, s_A \rangle\ \mathsf{in} \\
& \quad [\ldots]
\end{aligned}
$$

Figure 12: Process modelling an honest voter id voting for A, performing the optional audit phase (parts of the process that are identical to Figure 5 are omitted).

$$
\begin{array}{l}
\mathsf{new}\ \mathsf{sk_e};\\
(\\
\quad !\,(\ \mathsf{new}\ \mathsf{id} : \mathsf{any}; \mathsf{out}(c_{\mathsf{pub}}, \mathsf{id});\\
\qquad \mathsf{new}\ c_{vs} : \mathsf{channel}; \mathsf{new}\ c_{vd} : \mathsf{channel};\\
\qquad (\ \mathsf{Registrar}(\mathsf{id}, c_{vs})\\
\qquad\ \ |\ \mathsf{VoterAudit}_A(\mathsf{id}, c_{vs}, c_{vd})\\
\qquad\ \ |\ \mathsf{VD}(c_{pub}, c_{vd}, c_s(\mathsf{id}))\\
\qquad\ \ |\ \mathsf{VS}(c_{pub}, c_s(\mathsf{id}))))\\
\qquad |\\
\quad !\,(\ \mathsf{new}\ \mathsf{id} : \mathsf{any}; \mathsf{out}(c_{\mathsf{pub}}, \mathsf{id});\\
\qquad \mathsf{new}\ c_{vs} : \mathsf{channel}; \mathsf{new}\ c_{vd} : \mathsf{channel};\\
\qquad (\ \mathsf{Registrar}(\mathsf{id}, c_{vs})\\
\qquad\ \ |\ \mathsf{VoterAudit}_B(\mathsf{id}, c_{vs}, c_{vd})\\
\qquad\ \ |\ \mathsf{VD}(c_{pub}, c_{vd}, c_s(\mathsf{id}))\\
\qquad\ \ |\ \mathsf{VS}(c_{pub}, c_s(\mathsf{id}))))\\
\qquad |\\
\quad !\,(\ \mathsf{new}\ \mathsf{id} : \mathsf{any}; \mathsf{out}(c_{\mathsf{pub}}, \langle \mathsf{id}, c_s(\mathsf{id})\rangle);\\
\qquad (\ \mathsf{Registrar}(\mathsf{id}, c_{pub})\\
\qquad\ \ |\ \mathsf{VS}(c_{\mathsf{pub}}, c_s(\mathsf{id}))))\\
\qquad |\\
\quad \mathsf{Tally}\\
)
\end{array}
$$

Figure 13: Process modelling the BeleniosVS protocol.

The model of the whole protocol can be found in Figure 13. It is obtained by putting in parallel unbounded numbers of honest voters for each candidate, and an unbounded number of instances of the registrar and voting server processes, to be used by dishonest voters played by the attacker. Each of the honest voter processes has its own channels and instances of the voting device, registrar, and voting server processes.

Appendix E.
Proofs of the sufficient conditions for verifiability

E.1. Id based verifiability: Proof of Theorem 1

Consider a process P that satisfies the assumptions stated in section 4.1. Assume P guarantees recorded-as-intended, eligibility, individual verifiability (on identities), and consistent identities. We then prove that P is verifiable.

Let t be a trace of P.

Recall that, as defined in section 3.6, the bulletin board $\mathsf{BB}(t)$ is

$$\mathsf{BB}(t) = \{\!|b \in \mathcal{B}. \mid \exists \mathsf{id}, \mathsf{cred}.\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b)|\!\},$$

and $\mathsf{HV}(t)$ and $\mathsf{DV}(t)$ denote the sets of honest (resp. dishonest) voters in t, *i.e.* the sets of voters id such that $\mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H})$ (resp. $\mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{D})$) occurs in t for some credential cred. The set of voters in t is then $\mathsf{Voters}(t) = \mathsf{CHV}(t) \cup \mathsf{DV}(t)$.

Then $\mathsf{CHV}(t)$ is the subset of honest voters who have verified their vote, *i.e.* for which there is also an event of the form $\mathsf{Verified}(\mathsf{id}, *)$ in t. Let then $\overline{\mathsf{CHV}}(t) = \mathsf{HV}(t) \backslash \mathsf{CHV}(t)$ be the set of honest voters that do not successfully verify their votes.

By assumption, $\mathsf{HV}(t)$ and $\mathsf{DV}(t)$ are disjoint. Hence $\mathsf{CHV}(t) \uplus \overline{\mathsf{CHV}}(t) \uplus \mathsf{DV}(t)$ is a partition of $\mathsf{Voters}(t)$.

By the eligibility property (on identities), for any $\mathsf{id}, \mathsf{cred}, b$, if $\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t$ and $\mathsf{valid}(b)$ then $\mathsf{id} \in \mathsf{Voters}(t)$.

We may thus partition $\mathsf{BB}(t)$ into the ballots from voters in $\mathsf{CHV}(t)$, $\overline{\mathsf{CHV}}(t)$, and $\mathsf{DV}(t)$. For $\mathsf{V} \in \{\mathsf{CHV}, \overline{\mathsf{CHV}}, \mathsf{DV}\}$, let

$$\mathsf{BB}_{\mathsf{V}} = \{\!|b \in \mathsf{BB}(t). \mid \exists \mathsf{id} \in \mathsf{V}(t), \mathsf{cred}.\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b)|\!\},$$

and

$$S_{\mathsf{V}} = \{\!|\mathsf{open}(b), b \in \mathsf{BB}_{\mathsf{V}}|\!\}.$$

We have

$$\mathsf{BB}(t) = \mathsf{BB}_{\mathsf{CHV}} \uplus \mathsf{BB}_{\overline{\mathsf{CHV}}} \uplus \mathsf{BB}_{\mathsf{DV}}.$$

By the partial tally assumption, we then have

$$\mathsf{result}(t) = \rho(S_{\mathsf{CHV}}) * \rho(S_{\overline{\mathsf{CHV}}}) * \rho(S_{\mathsf{DV}})$$

As defined in section 3.6,

$$\mathcal{V}_{\mathsf{CHV}}(t) = \{\!|v | \exists \mathsf{id} \in \mathsf{CHV}(t).\ \mathsf{Voted}(\mathsf{id}, v) \in t|\!\}.$$

Since, by assumption, t contains at most one Voted event for each honest voter, for all v we have

$$\mathcal{V}_{\mathsf{CHV}}(t)(v) = |\{\mathsf{id} \in \mathsf{CHV}(t) | \mathsf{Voted}(\mathsf{id}, v) \in t\}|,$$

and thus

$$\begin{aligned}
\mathcal{V}_{\mathsf{CHV}}(t)(v) &= |\{\mathsf{id} \in \mathsf{CHV}(t) | \exists v'.\ \mathsf{Verified}(\mathsf{id}, v') \in t \ \wedge\ \mathsf{Voted}(\mathsf{id}, v) \in t\}| && \text{(by definition of } \mathsf{CHV}(t)) \\
&= |\{\mathsf{id} \in \mathsf{CHV}(t) | \mathsf{Verified}(\mathsf{id}, v) \in t \ \wedge\ \mathsf{Voted}(\mathsf{id}, v) \in t\}| && (*) \\
&= |\{\mathsf{id} \in \mathsf{CHV}(t) | \exists \mathsf{cred}, b.\ \mathsf{Verified}(\mathsf{id}, v) \in t \ \wedge\ \mathsf{Voted}(\mathsf{id}, v) \in t \\
&\qquad \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}| && \text{(by the individual verifiability property)} \\
&= |\{\mathsf{id} \in \mathsf{CHV}(t) | \exists \mathsf{cred}, b, v'.\ \mathsf{Verified}(\mathsf{id}, v') \in t \ \wedge\ \mathsf{Voted}(\mathsf{id}, v) \in t \\
&\qquad \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}| && (*) \\
&= |\{\mathsf{id} \in \mathsf{CHV}(t) | \exists \mathsf{cred}, b.\ \mathsf{Voted}(\mathsf{id}, v) \in t \\
&\qquad \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}| && \text{(by definition of } \mathsf{CHV}(t))
\end{aligned}$$

The two steps labelled with (*) are justified by the assumption that $\mathsf{Verified}(\mathsf{id}, v') \in t \Rightarrow \mathsf{Voted}(\mathsf{id}, v') \in t$, and that t contains at most one Voted event for each honest voter.

Hence, by the recorded-as-intended property on identities, since by definition $\mathsf{CHV}(t) \subseteq \mathsf{HV}(t)$,

$$\mathcal{V}_{\mathsf{CHV}}(t)(v) = |\{\mathsf{id} \in \mathsf{CHV}(t) | \exists \mathsf{cred}, b.\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}|.$$

By assumption, all Going-to-tally events in t contain distinct ballots. Thus, by the consistent identities property, all Going-to-tally events in t also contain distinct ids. Hence

$$\begin{aligned}
\mathcal{V}_{\mathsf{CHV}}(t)(v) &= |\{b \in \mathsf{BB}(t) | \exists \mathsf{id} \in \mathsf{CHV}(t), \mathsf{cred}.\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}| \\
&= S_{\mathsf{CHV}}(v)
\end{aligned}$$

Therefore, $\mathcal{V}_{\mathsf{CHV}}(t) = S_{\mathsf{CHV}}$.

Similarly,

$$\mathcal{V}_{\overline{\mathsf{CHV}}}(t) = \{\!|v | \exists \mathsf{id} \in \overline{\mathsf{CHV}}(t).\ \mathsf{Voted}(\mathsf{id}, v) \in t|\!\}.$$

Since, by assumption, t contains at most one Voted event for each honest voter, for all v we have

$$\mathcal{V}_{\overline{\mathsf{CHV}}}(t)(v) = |\{\mathsf{id} \in \overline{\mathsf{CHV}}(t) | \mathsf{Voted}(\mathsf{id}, v) \in t\}|.$$

Thus

$$\mathcal{V}_{\overline{\mathsf{CHV}}}(t)(v) \geq |\{\mathsf{id} \in \overline{\mathsf{CHV}}(t) | \exists \mathsf{cred}, b.\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b, v) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b) \ \wedge\ \mathsf{Voted}(\mathsf{id}, v) \in t\}|.$$

Hence, by the recorded-as-intended property, since by definition $\overline{\mathsf{CHV}}(t) \subseteq \mathsf{HV}(t)$, we have

$$\mathcal{V}_{\overline{\mathsf{CHV}}}(t)(v) \geq |\{\mathsf{id} \in \overline{\mathsf{CHV}}(t) | \exists \mathsf{cred}, b.\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}|.$$

As previously shown, all Going-to-tally events in t contain distinct ballots, and distinct ids. Thus

$$\begin{aligned}\mathcal{V}_{\overline{\mathsf{CHV}}}(t)(v) &\geq |\{b \in \mathsf{BB}(t) | \exists \mathsf{id} \in \overline{\mathsf{CHV}}(t), \mathsf{cred}.\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \ \wedge\ v = \mathsf{open}(b)\}| \\ &= S_{\overline{\mathsf{CHV}}}(v)\end{aligned}.$$

Therefore, $S_{\overline{\mathsf{CHV}}} \subseteq_{\mathsf{m}} \mathcal{V}_{\overline{\mathsf{CHV}}}(t)$.

Finally, by definition,

$$\begin{aligned}S_{\mathsf{DV}} &= \{\!|\mathsf{open}(b), b \in \mathsf{BB}_{\mathsf{DV}}|\!\} \\ &= \{\!|\mathsf{open}(b) | \exists \mathsf{id} \in \mathsf{DV}(t), \mathsf{cred}, v. \\ &\qquad \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)|\!\}.\end{aligned}$$

As previously explained, it follows from the assumptions that t contains at most one Going-to-tally event for each id. Hence, $|S_{\mathsf{DV}}| \leq |\mathsf{DV}(t)|$. In addition, by the eligibility property, $S_{\mathsf{DV}} \subseteq_{\mathsf{m}} \mathcal{V}$.

Therefore,

$$\mathsf{result}(t) = \rho(\mathcal{V}_{\mathsf{CHV}}(t)) * \rho(S_{\overline{\mathsf{CHV}}}) * \rho(S_{\mathsf{DV}})$$

with $S_{\overline{\mathsf{CHV}}} \subseteq_{\mathsf{m}} \mathcal{V}_{\overline{\mathsf{CHV}}}(t)$, $S_{\mathsf{DV}} \subseteq_{\mathsf{m}} \mathcal{V}$, and $|S_{\mathsf{DV}}| \leq |\mathsf{DV}(t)|$, and the verifiability property holds.

E.2. Credential based verifiability: Proof of Theorem 2

Consider a process P that satisfies the assumptions stated in the beginning of section 4. Assume P guarantees recorded-as-intended, eligibility, individual verifiability (on credentials), and consistent credentials, We then prove that P is verifiable.

Let t be a trace of P.

The proofs of this theorem follows the same structure as the one for Theorem 1.

As defined in section 3.6, the bulletin board $\mathsf{BB}(t)$ is

$$\mathsf{BB}(t) = \{\!|b \in \mathcal{B}. \mid \exists \mathsf{id}, \mathsf{cred}.\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b)|\!\}.$$

$\mathsf{HV}(t)$, $\mathsf{DV}(t)$, $\mathsf{CHV}(t)$ respectively denote the sets of honest voters, dishonest voters, and honest voters who verify their vote, *i.e.* the sets of voters id such that $\mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H})$ (resp. $\mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{D})$ and $\mathsf{Verified}(\mathsf{id}, *)$) occurs in t for some credential cred. The set of voters in t is then $\mathsf{Voters}(t) = \mathsf{CHV}(t) \cup \mathsf{DV}(t)$.

As in the previous proof, let $\overline{\mathsf{CHV}}(t) = \mathsf{HV}(t) \backslash \mathsf{CHV}(t)$ be the set of honest voters that do not successfully verify their votes. Then, as explained before, $\mathsf{CHV}(t) \uplus \overline{\mathsf{CHV}}(t) \uplus \mathsf{DV}(t)$ is a partition of $\mathsf{Voters}(t)$.

By the eligibility property on credentials, for any $\mathsf{id}, \mathsf{cred}, b$, if $\mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}, \mathsf{cred}, b) \in t$ then $\exists \mathsf{id}' \in \mathsf{Voters}(t). \exists l.\ \mathsf{Voter}(\mathsf{id}', \mathsf{cred}, l) \in t$.

We may thus partition $\mathsf{BB}(t)$ into the ballots from voters in $\mathsf{CHV}(t)$, $\overline{\mathsf{CHV}}(t)$, and $\mathsf{DV}(t)$. For $\mathsf{V} \in \{\mathsf{CHV}, \overline{\mathsf{CHV}}, \mathsf{DV}\}$, let

$$\mathsf{BB}'_{\mathsf{V}} = \{\!|b \in \mathsf{BB}(t). \mid \exists \mathsf{id} \in \mathsf{V}(t), \mathsf{id}' \in \mathcal{A}, \mathsf{cred}, l.\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, l) \in t \ \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b)|\!\},$$

and

$$S'_{\mathsf{V}} = \{\!|\mathsf{open}(b), b \in \mathsf{BB}'_{\mathsf{V}}|\!\}.$$

We have

$$\mathsf{BB}(t) = \mathsf{BB}'_{\mathsf{CHV}} \uplus \mathsf{BB}'_{\overline{\mathsf{CHV}}} \uplus \mathsf{BB}'_{\mathsf{DV}}.$$

By the partial tally assumption, we then have

$$\mathsf{result}(t) = \rho(S'_{\mathsf{CHV}}) * \rho(S'_{\overline{\mathsf{CHV}}}) * \rho(S'_{\mathsf{DV}}).$$

By definition,

$$\mathcal{V}_{\mathsf{CHV}}(t) = \{\!|v | \exists \mathsf{id} \in \mathsf{CHV}(t).\ \mathsf{Voted}(\mathsf{id}, v) \in t|\!\}.$$

Since, by assumption, t contains at most one Voted event for each honest voter, for all v we have

$$\mathcal{V}_{\mathsf{CHV}}(t)(v) = |\{\mathsf{id} \in \mathsf{CHV}(t) | \mathsf{Voted}(\mathsf{id}, v) \in t\}|,$$

and thus

$$\begin{aligned}
\mathcal{V}_{\mathsf{CHV}}(t)(v) &= |\{\mathsf{id} \in \mathsf{CHV}(t) | \exists v'.\ \mathsf{Verified}(\mathsf{id}, v') \in t \ \wedge\ \mathsf{Voted}(\mathsf{id}, v) \in t\}| && \text{(by definition of } \mathsf{CHV}(t)) \\
&= |\{\mathsf{id} \in \mathsf{CHV}(t) | \mathsf{Verified}(\mathsf{id}, v) \in t \ \wedge\ \mathsf{Voted}(\mathsf{id}, v) \in t\}| && (*) \\
&= |\{\mathsf{id} \in \mathsf{CHV}(t) | \exists \mathsf{id}', \mathsf{cred}, b. \\
&\qquad \mathsf{Verified}(\mathsf{id}, v) \in t \ \wedge\ \mathsf{Voted}(\mathsf{id}, v) \in t \ \wedge\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H}) \in t \\
&\qquad \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}| && \text{(by the individual verifiability property)} \\
&= |\{\mathsf{id} \in \mathsf{CHV}(t) | \exists \mathsf{id}', \mathsf{cred}, b, v'. \\
&\qquad \mathsf{Verified}(\mathsf{id}, v') \in t \ \wedge\ \mathsf{Voted}(\mathsf{id}, v) \in t \ \wedge\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H}) \in t \\
&\qquad \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}| && (*) \\
&= |\{\mathsf{id} \in \mathsf{CHV}(t) | \exists \mathsf{id}', \mathsf{cred}, b.\ \mathsf{Voted}(\mathsf{id}, v) \in t \ \wedge\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H}) \in t \\
&\qquad \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}| && \text{(by definition of } \mathsf{CHV}(t))
\end{aligned}$$

The two steps labelled with (*) are justified by the assumption that $\mathsf{Verified}(\mathsf{id}, v') \in t \Rightarrow \mathsf{Voted}(\mathsf{id}, v') \in t$, and that t contains at most one Voted event for each honest voter.

Hence, by the recorded-as-intended property on credentials,

$$\mathcal{V}_{\mathsf{CHV}}(t)(v) = |\{\mathsf{id} \in \mathsf{CHV}(t) | \exists \mathsf{id}', \mathsf{cred}, b.\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H}) \in t \wedge \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \wedge \mathsf{valid}(b) \wedge v = \mathsf{open}(b)\}|.$$

By assumption, each id in $\mathsf{CHV}(t)$ appears in a single Voter event, and thus is associated with only one credential. By the consistent credentials assumption, each of these ids is associated with a different credential. We therefore have

$$\begin{aligned}
&|\{\mathsf{id} \in \mathsf{CHV}(t) | \exists \mathsf{id}', \mathsf{cred}, b.\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H}) \in t \ \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ v = \mathsf{open}(b)\}| \\
\leq\ &|\{\mathsf{cred} | \exists \mathsf{id} \in \mathsf{CHV}(t), \mathsf{id}', b.\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H}) \in t \ \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}|.
\end{aligned}$$

In addition, by the consistent credentials assumption, each credential is associated with a different id. Thus

$$\begin{aligned}
&|\{\mathsf{id} \in \mathsf{CHV}(t) | \exists \mathsf{id}', \mathsf{cred}, b.\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H}) \in t \ \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ v = \mathsf{open}(b)\}| \\
\geq\ &|\{\mathsf{cred} | \exists \mathsf{id} \in \mathsf{CHV}(t), \mathsf{id}', b.\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H}) \in t \ \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}|.
\end{aligned}$$

Hence

$$\mathcal{V}_{\mathsf{CHV}}(t)(v) = |\{\mathsf{cred} | \exists \mathsf{id} \in \mathsf{CHV}(t), \mathsf{id}', b.\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H}) \in t \wedge \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \wedge \mathsf{valid}(b) \wedge v = \mathsf{open}(b)\}|.$$

By assumption, all Going-to-tally events in t contain distinct ballots. Thus, by the consistent credentials property, all Going-to-tally events in t also contain distinct credentials. Hence

$$\begin{aligned}
&|\{\mathsf{cred} | \exists \mathsf{id} \in \mathsf{CHV}(t), \mathsf{id}', b.\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, \mathsf{H}) \in t \ \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}| \\
=\ &|\{b \in \mathsf{BB}(t) | \exists \mathsf{id} \in \mathsf{CHV}(t), \mathsf{id}', \mathsf{cred}, l.\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, l) \in t \ \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}| \\
=\ &S'_{\mathsf{CHV}}(v)
\end{aligned}$$

Therefore, $\mathcal{V}_{\mathsf{CHV}}(t) = S'_{\mathsf{CHV}}$.

Similarly,

$$\mathcal{V}_{\overline{\mathsf{CHV}}}(t) = \{\!|v | \exists \mathsf{id} \in \overline{\mathsf{CHV}}(t).\ \mathsf{Voted}(\mathsf{id}, v) \in t|\!\}.$$

Since, by assumption, t contains at most one Voted event for each honest voter, for all v we have

$$\begin{aligned}
\mathcal{V}_{\overline{\mathsf{CHV}}}(t)(v) &= |\{\mathsf{id} \in \overline{\mathsf{CHV}}(t) | \mathsf{Voted}(\mathsf{id}, v) \in t\}| \\
&\geq |\{\mathsf{id} \in \overline{\mathsf{CHV}}(t) | \exists \mathsf{id}', \mathsf{cred}, b, l. \\
&\qquad \mathsf{Voted}(\mathsf{id}, v) \in t \ \wedge\ \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, l) \in t \ \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}|.
\end{aligned}$$

By assumption, each voter is only registered once and $\overline{\mathsf{CHV}}(t) \subseteq \mathsf{HV}(t)$. Thus for all $\mathsf{id} \in \overline{\mathsf{CHV}}(t)$ such that $\mathsf{Voter}(\mathsf{id}, \mathsf{cred}, l) \in t$ for some cred, l, we know that $l = \mathsf{H}$. Hence, by the recorded-as-intended property on credentials,

$$\begin{aligned}
\mathcal{V}_{\overline{\mathsf{CHV}}}(t)(v) &\geq |\{\mathsf{id} \in \overline{\mathsf{CHV}}(t) | \exists \mathsf{id}', \mathsf{cred}, b, l. \\
&\qquad \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, l) \in t \ \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}| \\
&\geq |\{\mathsf{cred} | \exists \mathsf{id} \in \overline{\mathsf{CHV}}(t), \mathsf{id}', \mathsf{cred}, b, l. \\
&\qquad \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, l) \in t \ \wedge\ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge\ \mathsf{valid}(b) \ \wedge\ v = \mathsf{open}(b)\}|
\end{aligned}$$

since, as before, each Going-to-tally event contains a different credential. Then, following the same reasoning, we have

$$\begin{aligned}\mathcal{V}_{\overline{\mathsf{CHV}}}(t)(v) &= |\{b \in \mathcal{B} | \exists \mathsf{id} \in \overline{\mathsf{CHV}}(t), \mathsf{id}' \in \mathcal{A}, \mathsf{cred}, l. \\ &\qquad \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, l) \in t \ \wedge \ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge \ \mathsf{valid}(b) \ \wedge \ v = \mathsf{open}(b)\}| \\ &= S'_{\overline{\mathsf{CHV}}}(v).\end{aligned}$$

Therefore, $S'_{\overline{\mathsf{CHV}}} \subseteq_{\mathsf{m}} \mathcal{V}_{\overline{\mathsf{CHV}}}(t)$.

Finally, by definition,

$$\begin{aligned}&S'_{\mathsf{DV}} \ \{\!|\mathsf{open}(b), b \in \mathsf{BB}'_{\mathsf{DV}}|\!\} \\ &\quad = \{\!|\mathsf{open}(b) | \exists \mathsf{id} \in \mathsf{DV}(t), \mathsf{id}', \mathsf{cred}, v, l. \\ &\qquad \mathsf{Voter}(\mathsf{id}, \mathsf{cred}, l) \in t \ \wedge \ \mathsf{Going\text{-}to\text{-}tally}(\mathsf{id}', \mathsf{cred}, b) \in t \ \wedge \ \mathsf{valid}(b) \ \wedge \ v = \mathsf{open}(b)|\!\}\end{aligned}.$$

As previously explained, it follows from the assumptions that t contains at most one Going-to-tally event for each cred, and that each cred is associated with a different id. Hence, $|S'_{\mathsf{DV}}| \leq |\mathsf{DV}(t)|$. In addition, by the eligibility property, $S'_{\mathsf{DV}} \subseteq_{\mathsf{m}} \mathcal{V}$.

Therefore,

$$\mathsf{result}(t) = \rho(\mathcal{V}_{\mathsf{CHV}}(t)) * \rho(S'_{\overline{\mathsf{CHV}}}) * \rho(S'_{\mathsf{DV}})$$

with $S'_{\overline{\mathsf{CHV}}} \subseteq_{\mathsf{m}} \mathcal{V}_{\overline{\mathsf{CHV}}}(t)$, $S'_{\mathsf{DV}} \subseteq_{\mathsf{m}} \mathcal{V}$, and $|S'_{\mathsf{DV}}| \leq |\mathsf{DV}(t)|$, and the verifiability property holds.