# Themis: an On-Site Voting System with Systematic Cast-as-intended Verification and Partial Accountability

Mikaël Bougon, Hervé Chabanne, Véronique Cortier, Alexandre Debant, Emmanuelle Dottax, Jannik Dreier, Pierrick Gaudry, Mathieu Turuani

# Themis: an On-Site Voting System with Systematic Cast-as-intended Verification and Partial Accountability

Mikaël Bougon
mikael.bougon@idemia.com
IDEMIA
Paris, France

Hervé Chabanne
herve.chabanne@idemia.com
IDEMIA
Paris, France

Véronique Cortier
veronique.cortier@loria.fr
Université de Lorraine, Inria, CNRS
Nancy, France

Alexandre Debant
alexandre.debant@loria.fr
Université de Lorraine, Inria, CNRS
Nancy, France

Emmanuelle Dottax
emmanuelle.dottax@idemia.com
IDEMIA
Bordeaux, France

Jannik Dreier
jannik.dreier@loria.fr
Université de Lorraine, Inria, CNRS
Nancy, France

Pierrick Gaudry
pierrick.gaudry@loria.fr
Université de Lorraine, Inria, CNRS
Nancy, France

Mathieu Turuani
mathieu.turuani@loria.fr
Université de Lorraine, Inria, CNRS
Nancy, France

## ABSTRACT

We propose an on-site voting system Themis, that aims at improving security when local authorities are not fully trusted. Voters vote thanks to voting sheets as well as smart cards that produce encrypted ballots. Electronic ballots are systematically audited, without compromising privacy. Moreover, the system includes a precise dispute resolution procedure identifying misbehaving parties.

We conduct a full formal analysis of Themis using ProVerif, with a novel approach in order to cover the modular arithmetic needed in our protocol. In order to evaluate the usability of our system, we organized a voting experiment on a (small) group of voters.

## 1 INTRODUCTION

Electronic voting, and voting in general, aims at achieving two main properties: vote privacy and verifiability. Vote privacy ensures that no one knows how a particular voter voted. Verifiability guarantees that the result corresponds to the votes of the voters. Verifiability typically refers to:

- *individual verifiability*: voters should be able to check that their ballot has been counted and corresponds to their *intent*. This last property is often referred to as *cast-as-intended*.
- *universal verifiability*: voters and external auditors should be able to check that the result corresponds to the ballots and that the ballots come only from legitimate voters.

Other important security properties include resistance against coercion or vote buying, and accountability.

There are two main families of voting systems. Internet voting is a form of remote voting where voters vote from home or at work, using their own device (phone, tablet, or computer). Their advantage is that voters do not need to travel to polling stations. Many Internet voting protocols have been proposed, e.g., Civitas [11, 16], Helios [1], Belenios [12], or Selene [24]. On-site voting systems assume that voters attend a polling station. Their goal is to enhance the trust compared to pure paper-based voting, for example when counting the ballots is tedious or when the local authorities are not trustworthy. Many protocols have been proposed, such as Prêt-à-voter [25], Scantegrity [10], or STAR-Vote [5].

For systems of both families, the *cast-as-intended* property is often an unresolved issue. Some systems simply assume a trusted device and hence do not guarantee this property. For example, Civitas, Selene, or Belenios simply assume that voters use a trusted voting platform. In Helios and STAR-Vote, a cast-or-audit challenge has been proposed by Benaloh [4]: just before casting their ballot, a voter may decide to audit it to check that it actually encodes their vote. This approach can be applied to other protocols as well. However, this audit procedure is rarely done in practice and has been shown to be hard to understand by voters [23]. Prêt-à-Voter allows voters to spoil ballots of their choice and have them audited. In both cases, audits are optional, hence the security strongly depends on voters' behavior. Moreover, for on-site voting, such systems are subject to "Dieselgate attacks": if a local authority suspects that a voter is likely to audit, it may instruct the system to behave as expected. In STAR-Vote, this may be implemented with a special, hidden, button. In Prêt-à-Voter, local authorities could add malicious voting sheets only when naive voters are present.

Another often overlooked property is *accountability* [3, 20]. Verifiability is certainly a prerequisite, but when something wrong is detected, how should the election proceed? Typically the election cannot be restarted and it is tempting to simply ignore that some verifiability checks have failed. Accountability guarantees not only that any attack will be detected, but also that someone will be blamed. Conversely, a participant should not be blamed if they behaved honestly. This last property is also called *defensibility* [2].

*Our contributions.* We propose an on-site voting system, Themis, that provides a systematic cast-as-intended: voters always audit their own ballot to check that it corresponds to their voting intent, without losing vote privacy. Our system has been designed in collaboration with a company, IDEMIA. An important requirement was accountability (for liability) but also the fact that the system should not use on-site printers (as opposed to STAR-Vote) since

*xx, xx, xx*
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$xx
https://doi.org/10.1145/nnnnnnn.nnnnnnn

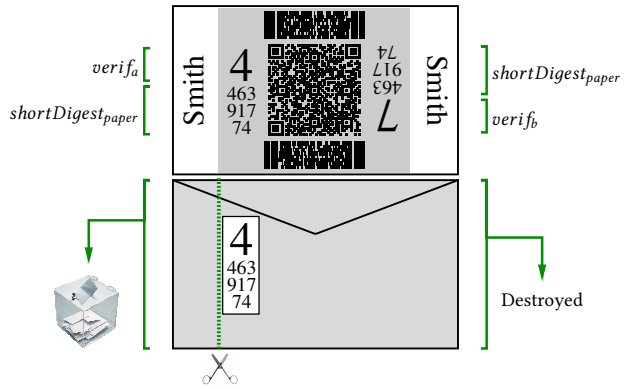



**Figure 1: Design of the ballots and envelope. The grey part is hidden under an opaque covering like a scratchcard.**

printers have been identified as an important risk of failure in some contexts. Instead, we propose a system based on smart cards with screens, as well as voting sheets printed in advance. Smart cards allow to protect against dishonest local authorities. However, compared to paper, malicious smart cards may change their content and their display. Thanks to the systematic cast-as-intended property, smart cards do not need to be trusted either. Interstingly, our system still offers a paper box that can be further audited using standard RLA (Risk Limiting Audit) [22] or used as a backup if the entire electronic system crashes.

Our systematic audit is based on the following idea. A ballot is built from the encrypted vote $v$ of the voter. It also contains the encryption of two small integers $a$ and $b$, such that $a + b \equiv v \bmod c$ where $c$ is (roughly) the number of candidates. A zero-knowledge proof guarantees that the three ciphertexts indeed satisfy this relation. Hence the smart card produces a ballot of the form:

$$\mathsf{enc}(\mathsf{pk}, v), \mathsf{enc}(\mathsf{pk}, a), \mathsf{enc}(\mathsf{pk}, b), \mathsf{ZKP}(a + b \equiv v \bmod c)$$

where pk denotes the public key of the election. The ballot also contains a proof that $v, a, b$ are integers between 0 and $c - 1$. The voter can see $v$, $a$ and $b$ on their voting sheet. They request to audit either $a$ or $b$. Assume for example that the voter requests $a$. The smart card must then provide the randomness $r_a$ used to produce $\mathsf{enc}(\mathsf{pk}, a)$. The extended ballot

$$a, \mathsf{enc}(\mathsf{pk}, v), \mathsf{enc}(\mathsf{pk}, a), r_a, \mathsf{enc}(\mathsf{pk}, b), \mathsf{ZKP}(a + b \equiv v \bmod c)$$

is published so that the voter can check their value $a$, while external auditors will check that $\mathsf{enc}(\mathsf{pk}, a)$ indeed encrypts $a$. All ballots must have been audited in order to be counted, either with the first integer or the second one, depending on each voter's decision. If a malicious smart card wishes to encrypt $v'$ instead of $v$, it must also change either $a$ or $b$ (or both). Hence a cheating smart card will be caught with probability 1/2 each time it tries to modify a vote.

The paper ballot can resemble the format of Figure 1, where the name of the candidate (Smith), and the two audit values $a$ and $b$ (4 and 7) are clearly visible. The random choice is implemented by the symmetry of the ballot, that can be inserted in two positions in the envelope. The roles of the other parts of the ballot are explained in Section 2.2.

Most existing schemes fail to explain what to do when a check fails. We provide a precise dispute resolution procedure that identifies the misbehaving party (or a group of parties) without breaking vote privacy. This procedure contains many sub-cases that intuitively correspond to the corruption scenarios that may arise. Luckily, this procedure is launched only when some anomaly is detected and does not interfere with the normal flow of the protocol.

As a second main contribution, we provide a full analysis of Themis using ProVerif [8]. ProVerif is a state-of-the-art tool dedicated to the analysis of security protocols such as TLS [6], Signal [18], or Noise [19]. We formally prove that Themis guarantees:

- Vote privacy, provided that the local authorities do not collude with the equipment (smart card and terminal);
- End-to-end verifiability, provided that printed ballots (that contain $v, a, b$) are generated honestly or provided that local observers check that printed ballots are well formed. This includes in particular the cast-as-intended property.
- Accountability, with the limitation that our system sometimes points to a group of possible misbehaving participants instead of precisely one.

One particular issue in the analysis is due to our novel ballot construction to ensure cast-as-intended. Modeling modular arithmetic is out of reach of a tool like ProVerif – even reducing our analysis to very small integers (e.g. with only two candidates) is impractical. Hence, we had to devise new strategies. For reachability properties such as verifiability, we identify sufficient conditions fulfilled by the modular arithmetic. More formally, we introduce an event `isSum` for each triple $(v, a, b)$ and we write properties of the form

$$\mathtt{isSum}(v, a, b) \wedge \mathtt{isSum}(v', a, b) \Rightarrow v = v' .$$

This is clearly satisfied if $v \equiv a + b \bmod c$ and $v' \equiv a + b \bmod c$. (Remember that $v, a, b$ are integers between 0 and $c - 1$.) Hence it is sufficient to consider executions where such a property is satisfied. We use here the recent introduction of *restrictions* in ProVerif [9].

Accountability can also be modeled as a reachability property, and thus can be analyzed in a similar way. We found out that accountability is rather easy to analyze for a tool like ProVerif, but hard to get right. We used ProVerif to guide the design of our dispute resolution procedure, and to identify missing corner cases.

The analysis of vote privacy cannot be conducted in the same lines, as vote privacy is expressed as an equivalence property and such an approach would be unsound. More precisely, vote privacy is expressed as the equivalence between two processes $P$ and $Q$ where $P$ models a scenario where a voter $A$ votes 0 and voter $B$ votes 1, while $Q$ models the same scenario where votes are swapped.

$$C \mid \mathsf{Voter}(A, 0) \mid \mathsf{Voter}(B, 1) \approx C \mid \mathsf{Voter}(A, 1) \mid \mathsf{Voter}(B, 0) .$$

We need to prove that for any trace $tr$ of $P$ that satisfies the properties of modular arithmetic, there is a trace $tr'$ of $Q$ that also satisfies the properties of modular arithmetic. We use ProVerif in an original way: We first use it to prove that $P \approx Q$, which is *a priori* too weak compared to the desired property. We note however that ProVerif proves diff-equivalence instead of observational equivalence, which is a stronger property (but still not the desired one). Thanks to the fact that ProVerif allows to prove lemmas in the context of diff-equivalence, we establish a property that controls how triples $(v, a, b)$ can be formed in $P$ and $Q$. We conclude with a small proof

by hand, taking advantage of this property and diff-equivalence. We believe that our proof strategies, both for reachability and equivalence properties can be of independent interest when someone wishes to go beyond the usual limits of ProVerif.

Finally, we have realized a prototype implementation of our protocol to get user feedback, in small, mock elections with a total of 14 voters, recruited within IDEMIA, that were observed and interviewed. Despite the complexity of our system, this shows that it can be used in practice, although not all voters understand the rationale behind the different steps of the voting procedure. We also provide estimations of the computational cost showing that our protocol can indeed by deployed on smart cards, despite a rather costly zero-knowledge proof.

*Related work.* STAR-Vote [5] is an on-site voting system that uses printers and relies on the cast-or-audit Benaloh mechanism. It aims at providing a much more secure alternative to many voting machines used in the US, still enabling risk limiting audits (as for our scheme). The audit mechanism is optional and it seems hard to make it mandatory. Indeed, a privacy preserving audit procedure would work as follows. 1. roll a dice to decide whether you will audit or vote. 2. In case of audit, roll a dice to decide which candidate to pick 3. repeat. This seems hard to instruct to voters and it would be even harder to check that they follow the procedure. The dispute resolution procedure associated to the optional audit is left unspecified in [5], when a voter discovers that their ballot does not contain their vote. Should the system be stopped or should the complaint be ignored?

Prêt-à-Voter [25] has been proposed about 20 years ago and is based on voting sheets, that voters can spoil for audit. It is one of the first system to offer an electronic counterpart of the physical ballot box. Audit is optional but could easily be mandatory: voters could be instructed to always pick a ballot at random and audit it. However, the audit mechanism requires a (trusted) device hence voters would probably need to do this at home. In this case, it would be harder for voters to complain and it is also difficult to check that voters indeed audited their ballots. Moreover, Prêt-à-Voter has been shown to be subject to a chain voting attack [15]: a coercer may first steal a voting sheet and instruct a voter to 1. vote with it; 2. bring back their own, unused, voting sheet. This way, the coercer will be able to check that the voter selected the requested candidates and is given a new voting sheet in order to coerce the next voter.

Demos [17] is an postal voting system that proposes a systematic cast-as-intended mechanism. The setup authority sends two ballots to each voter. One ballot is used for audit, the other one is used to cast the vote, with no modification. Hence the setup authority learns the voter's vote. This can be acceptable for postal voting but not for on-site voting. Moreover, as designed, the verifications made by voters are done only after the tally, which renders contestability harder.

Many traditional EVM (Electronic Voting Machines) do not use any cryptography. Instead, the voter reviews a paper ballot, printed by the machine. A fraction of machines, selected at random are audited, in order to check that the outcome of the machine corresponds to the ballots counted by hand. Risk Limit Audit [22] allows to determine the number of machines to be audited and to provide a guarantee on the margin error. This approach suffers from the fact that several manual counts are needed, which means that ballot boxes need to be constantly supervised in the meantime. This may be difficult when local authorities are not sufficiently trusted.

Accountability has been identified as a missing property more than 10 years ago [2, 20] but most systems fail to provide precise dispute resolution mechanisms. [3] proposes a general formalism for dispute resolution and combines machine-checked proofs with traditional pen-and-paper proofs to study a mini-voting protocol.

In short, the main advantage of Themis over [5] and [25] is that it provides a higher level of cast-as-intended since *all* ballots are audited. Compared to [17], it preserves ballot privacy w.r.t. the printing authority (as [25], while [5] does not need a printing authority at all). Compared to these three systems, Themis also offers a precise dispute resolution mechanism, yielding partial accountability: the misbehaving party can be identified, except in some case where the search for the guilty authority is only reduced to a group of parties.

## 2 THEMIS DESIGN

The voting protocol follows roughly the traditional paper voting process. There is a first point where voters are *registered*, their eligibility is checked, and they pick ballot(s). Then there is a *voting booth* where the voters make their choices. Finally there is a *confirmation point* where the ballot is confirmed and put into the ballot box.

In terms of equipment the protocol uses (per polling station):

- a large number of preprinted *paper ballots* of a particular design (described below);
- a pool of special *smart cards* with a small e-ink display;
- one *voting terminal* with a card reader and an optical reader (able to read QR-codes and barcodes) per *voting booth*. There can be several voting booths in a polling station;
- a single *confirmation terminal* per polling station, with a card and an optical reader, next to the *paper ballot box*;
- a large screen visible from the entire polling station, showing a list of the last electronic ballots and their status, called the *general election screen*, connected to a single local server maintaining the *electronic ballot box*;
- a *reset terminal* used to reset the smart cards after use so that they can be used again;
- a *paper logbook* used to note important events independently of the electronic devices

The voting terminals and the confirmation terminal are connected to the server via a local network, and we assume secure (authenticated and confidential) channels between the terminals and the server, typically established through TLS connections. For the security properties, we consider the following stakeholders:

- the *voters*;
- the *local authorities*, in charge of a polling station, in particular the officials at the registration and confirmation points;
- the *global authorities*, in charge of setting up and organizing the election, in particular generating the keys, computing the final tallying, etc;
- the *printing authority*, in charge of generating and printing the ballots before the election starts;
- the *observers*, present in the polling station(s), and auditing voting equipment (ballots, cards, terminals, ...);

- the *supplier* for the hard- and software (cards, terminals, servers, ...).

## 2.1 The voter's user experience

For a voter, the voting process runs as follows (see also Figure 5).

First of all, their identity and eligibility are verified at the registration desk. This is outside the scope of the protocol, and can for example be accomplished using traditional voter rosters.

The voter then picks a smart cart from a pool of cards, one preprinted ballot for each candidate, and one envelope with a window in a specific place (see Figure 1 for the ballot design). These ballots have a special design: the name of the candidate is printed on both sides, while there is a QR-code containing some machine-readable data in the middle, and further numbers and barcodes next to it. Moreover, everything except for the name of the candidate is hidden under an opaque covering that can be scratched away.

The voter then enters the voting booth, which is equipped with a voting terminal with a card reader and an optical reader able to read the QR-codes and barcodes on the ballot. The voter then scratches the covering from the ballot of the candidate they want to vote for, and inserts the smart card into the reader. They then scan the QR-codes and barcodes, and the terminal shows a copy of the ballot, including in particular the name of the candidate the voter selected. They confirm their selection, and insert the ballot into the envelope. When the terminal confirms that their electronic ballot has been successfully created and inserted into the database, they can remove the smart card from the terminal. The smart card now shows an identifier of their electronic ballot and a short identifier of the paper ballot on its screen. Before leaving the voting booth, the voter destroys the unused ballots to prevent deducing who they voted for, by elimination.

Outside the voting both, using the identifier shown by the card, they can check whether their electronic ballot appears on the list of ballots shown on the general election screen.

In the final step, the voter needs to confirm their ballot. At the confirmation point, the voter inserts their smart card into the confirmation terminal. Then the entry corresponding to their ballot on the general election screen will be marked as *Under Confirmation*. Together with the official, they confirm that the identifier shown on the general screen corresponds to the identifier shown on their smart card, and that the *paper ballot digest* visible through the window of the envelope ("46391774" in Figure 1) corresponds to the one shown on their smart card. They then scan the number visible through the window of the envelope (the small *verification code*, "4" in the example from Figure 1). The general election screen also shows the verification code, which must match the code printed on the ballot. The official confirms that the scan was correct, and then the ballot status changes to *In Audit*. If the verification code on the screen and on the paper ballot match, the voter and the official confirm the vote, and the electronic ballot changes status to *Confirmed* on the general election screen. The voter then inserts their envelope into a machine that cuts off and destroys the part containing all codes and numbers, while the remainder is put into the ballot box (symbolized by the scissors in Figure 1). This part only contains the name of the selected candidated (hidden by the envelope). They also return their smart card, which is reset in the reset terminal, and can then be returned to the pool of cards at the voter registration point.

## 2.2 Detailed description

We now present the protocol in more details. We assume that the *global authorities* fix the list of candidates, and assign to each of the $\text{num}_{\text{cand}}$ candidates a unique odd integer $\text{id}_{\text{cand}}$ between 1 and $2 \times \text{num}_{\text{cand}} - 1$. They also establish the list of eligible voters, organize the necessary equipment and infrastructure for all polling station(s) (ballots, smart cards, terminals, screens, etc.). The global authorities are also in charge of the cryptographic setup (key generation) and the tallying of the (electronic) ballots. The exact tallying procedure is out of scope for this paper, standard solutions are mix-networks (e.g., [14, 26]) or homomorphic tallying (such as in Helios [1]).

In the following, we write $\texttt{enc}(PK, m, r)$ for the encryption of a message $m$ using the public key $PK$ and randomness $r$.

*2.2.1 Electronic ballot box, server & general election screen.* The server maintains the electronic ballot box, which is published at the end of the election. The ballot box is implemented using a hash chain, where each new ballot or status update is inserted in the form of block containing a hash of the previous block. Each block is also signed by the server. Each polling station has only one server, one electronic ballot box, and one general election screen.

The general election screen shows a human-readable excerpt of the last blocks in the electronic ballot box, plus some QR-codes that allow observers to check the integrity of the chain, and to verify that the ballot box published at the end of the election corresponds to the one built during the polling (see Section 2.2.5).

*2.2.2 Ballot construction.* The printing authority is responsible for generating and printing the paper ballots, and has a secret signing key $\text{SK}_{\text{Print}}$. For each ballot, it needs to compute:

- A unique serial number per ballot called $\text{SN}_{\text{paper}}$.
- A random value called $\text{rand}_{\text{paper}}$.
- Two random verification codes called $\text{verif}_{\text{a}}$ and $\text{verif}_{\text{b}}$ ("4" and "7" in the example in Figure 1), where
  - $\text{verif}_{\text{a}}$ is even and $\text{verif}_{\text{b}}$ is odd,
  - both values are between 0 and $2 \times \text{num}_{\text{cand}} - 1$,
  - $\text{verif}_{\text{a}} + \text{verif}_{\text{b}} \equiv \text{id}_{\text{cand}} \bmod (2 \times \text{num}_{\text{cand}})$ where $\text{id}_{\text{cand}}$ is the number assigned to the candidate.
- A $\text{digest}_{\text{paper}} = \text{Hash}_{\text{paper}}(\text{SN}_{\text{paper}}, \text{rand}_{\text{paper}})$ called *paper ballot digest*, and its short version $\text{shortDigest}_{\text{paper}}$ (often abbreviated by $\text{SD}_{\text{paper}}$) computed by truncating it to the first 8 digits. $\text{shortDigest}_{\text{paper}}$ has to be unique *among all ballots* (the printing authority has to choose $\text{rand}_{\text{paper}}$ such that this property holds).

The ballots (as shown in Figure 1) contain

- The name of the candidate.
- The two verification codes $\text{verif}_{\text{a}}$ and $\text{verif}_{\text{b}}$.
- The short paper ballot digest $\text{shortDigest}_{\text{paper}}$.
- The small QR code containing $\text{SN}_{\text{paper}}$, $\text{rand}_{\text{paper}}$, and a signature on these values.
- The big QR code containing the number of the candidate $\text{id}_{\text{cand}}$, the confirmation codes $\text{verif}_{\text{a}}$ and $\text{verif}_{\text{b}}$, and a signature on these values.

The reason for having two different QR codes is that the small QR code contains data that can be revealed without breaking privacy, during the first step of the dispute resolution procedure.

*2.2.3 Voting Booth.* The voting booth is equipped with a voting terminal with a card reader and an optical reader for the QR-codes and barcodes on the ballot. The voter enters with a smart card and (several) paper ballots, which they picked up at the registration point.

The terminal initially waits for the voter to insert the smart card, then it invites the voter to scan their paper ballot. Once the paper ballot is scanned, it checks the two signatures contained in the QR codes, as well as the well-formedness of the other ballot data. If a signature is invalid or the ballot is not well-formed, an error message is displayed and the terminal waits for the voter to scan another ballot. Otherwise, the terminal displays a copy of the ballot, including the name of the candidate as well as $\text{verif}_\text{a}$ and $\text{verif}_\text{b}$, and asks the voter to confirm. When the voter confirms their choice, the data is sent to the smart card. The terminal then asks the voter to insert the paper ballot into the envelope, and to wait until the electronic ballot is produced. Note that the ballot can be inserted in two different directions into the envelope; this choice determines whether $\text{verif}_\text{a}$ or $\text{verif}_\text{b}$ is visible through the envelope, and hence fixes the audit challenge for the confirmation step.

In the meantime, the smart card produced the electronic ballot. It received all data scanned by the voting terminal, and checked all signatures and whether the ballot is well-formed. If all checks succeed, it creates the ballot as follows:

- It recomputes the paper ballot digest and the shortened human-readable version.
- It generates a fresh random value $\text{rand}_\text{elec}$ (the electronic counterpart of $\text{rand}_\text{paper}$) and computes the *electronic ballot digest* $\text{digest}_\text{elec} = \text{Hash}_\text{elec}(\text{SN}_\text{paper}, \text{rand}_\text{elec})$. It also deduces a *short electronic ballot digest* $\text{shortDigest}_\text{elec}$ (or $\text{SD}_\text{elec}$) by considering only the first 8 digits of $\text{digest}_\text{elec}$. The card checks with the server whether the short digest $\text{shortDigest}_\text{elec}$ is unique within the electronic ballot box. If this is not the case, it generates a new random nonce and tries again.
- The card then computes the *electronic ballot*

$$\text{ballot}_\text{elec} = \big(\mathsf{enc}(\text{PK}_\text{Elect}, \text{id}_\text{cand}, \text{r}_\text{cand}), \mathsf{enc}(\text{PK}_\text{Elect}, \text{verif}_\text{a}, \text{r}_\text{a}), \mathsf{enc}(\text{PK}_\text{Elect}, \text{verif}_\text{b}, \text{r}_\text{b}), \text{ZKP}, \text{digest}_\text{elec}\big)$$

  where ZKP is a zero knowledge proof showing that (1) $\text{verif}_\text{a}$ is even and $\text{verif}_\text{b}$ is odd, (2) both values are between 0 and $2\times\text{num}_\text{cand}-1$, and (3) $\text{verif}_\text{a} + \text{verif}_\text{b} \equiv \text{id}_\text{cand} \mod (2\times \text{num}_\text{cand})$; $\text{r}_\text{cand}$, $\text{r}_\text{a}$ and $\text{r}_\text{b}$ are randomness used for encryption.
- It computes a signature on $\text{ballot}_\text{elec}$, noted $\text{sig}_{\text{ballot}_\text{elec}}$.

When the terminal receives the electronic ballot produced by the smart card, the terminal checks the card's signature, and the validity of the zero-knowledge proof, before forwarding everything to the server. The server confirms the insertion of the ballot into the electronic ballot box by returning the signed block of the hash chain. Figure 2 sums up the interactions in the voting booth.

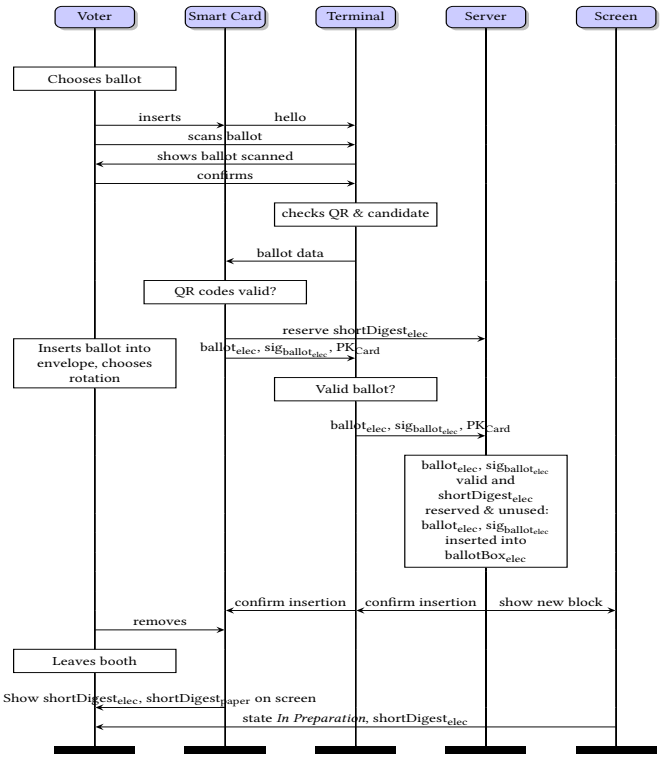



**Figure 2: Interactions in the voting booth, simplified**

*2.2.4 Confirmation.* At the confirmation point, the voter inserts their smart card into the confirmation terminal, and the card transmits the short digest $\text{shortDigest}_\text{elec}$ together with a signature on this value, noted $\text{sig}_{\text{shortDigest}_\text{elec}}$, to the server via the terminal. Then the entry corresponding to their ballot on the general election screen will be marked as *Under Confirmation*. The terminal is constructed in such a way that even when card is inserted, its screen is still readable both for the voter and the election official. Together with the official, the voters confirms that the identifier $\text{shortDigest}_\text{elec}$ shown on the large screen corresponds to the identifier $\text{shortDigest}_\text{elec}$ shown on their smart card, and that the identifier of the paper ballot $\text{shortDigest}_\text{paper}$ shown by the screen of the card is identical to the one shown on the paper ballot.

Then, the voter scans the numbers visible through the window of the envelope (the small verification code $\text{code}_\text{verif}$, i.e., $\text{verif}_\text{a}$ or $\text{verif}_\text{b}$, depending on the orientation of the ballot inside the envelope, and the larger short paper ballot digest). The terminal transmits the scanned code to the server, who adds a block to the electronic ballot box and shows the number on the general election screen. The voter and the official confirm together that the scan was correct. (If the scan is incorrect, the scan is repeated until it matches.) The signed challenge is then sent to the card. The card then needs to prove that it encrypted the challenge value correctly by providing the randomness used to encrypt. More precisely, the card then checks the signature of the challenge and whether the challenge value actually corresponds to $\text{verif}_\text{a}$ or $\text{verif}_\text{b}$. If this is the case, it responds with $\text{shortDigest}_\text{elec}$, a bit specifying whether

$\text{verif}_a$ or $\text{verif}_b$ is challenged, and the randomness used to encrypt the value, together with a signature. Otherwise, it responds with an error. Note that the card will only accept a single challenge to ensure privacy of the vote: It refuses any further challenge, even if the first challenge was invalid. If it receives a second challenge, it returns the first one to prove its honesty.

The server checks the card's response and verifies the randomness. If this succeeds it creates an new block containing the response, and on the general election screen the ballot will be marked as *In Audit* and the verification code ($\text{verif}_a$ or $\text{verif}_b$) is shown. If the numbers match, the voter and the official confirm the vote together, and the electronic ballot changes status to *Confirmed* on the general election screen. The voter then inserts their envelope into a machine that cuts off and destroys the part containing all codes and numbers, while the remainder is put into the ballot box (symbolized by the scissors in Figure 1). They also return their smart card, which is reset in the reset terminal, and can then be returned to the pool of cards at the voter registration. Figure 3 sums up the interactions at the confirmation point. Note that there is only one confirmation point per polling station, and only one ballot can be confirmed at a time.

The zero-knowledge proof ensures the correct link between the vote, $\text{verif}_a$ and $\text{verif}_b$. Hence, if the card encrypted a vote for a different candidate, either the encryption of $\text{verif}_a$ or $\text{verif}_b$ (or both) must contain a different value. The audit at the confirmation point thus ensures that a misbehaving card will be caught with a probability 1/2.

*2.2.5 Observers.* To ensure the integrity of the election, observers (and interested voters) can observe and audit different parts of the election process. As in traditional pen and paper voting, observers can monitor the behavior of the local authorities in each polling station, and check that they adhere to the rules. In addition, this protocol contains multiple special checks:

- The electronic ballot box published after the election is audited, i.e., the integrity of the hash chain, the validity of all signatures, zero-knowledge proofs, and encryption randomnesses are checked. All checks supposed to be executed by the server are re-checked by the observers.
- Voters and/or observers take regular "snapshots" (i.e., pictures) of the general election screen, e.g., using their smartphones, and audit those using a special app which (1) checks the validity of the server's signatures inside the QR-codes, and (2) shows the data contained in the QR-codes so that the voters/observers can compare this to the human-readable data ($\text{shortDigest}_{\text{elec}}$, the current state of the ballot, and the scanned or confirmed verification code, depending on the state of the ballot) shown on the screen.
  Moreover, these snapshots are compared to the electronic ballot box published after the election. As these snapshots contain valid signatures, the server can be blamed if the final electronic ballot box published after the election does not include these signatures, preventing modifications.
- The number of confirmed electronic ballots at the end of the election is compared to the number of voters who signed the register. This is to avoid ballot stuffing in the case where local authorities are honest but the voting server and the smartcard are dishonest. They would have enough material to forge electronic ballots but this would be eventually caught by the authorities.

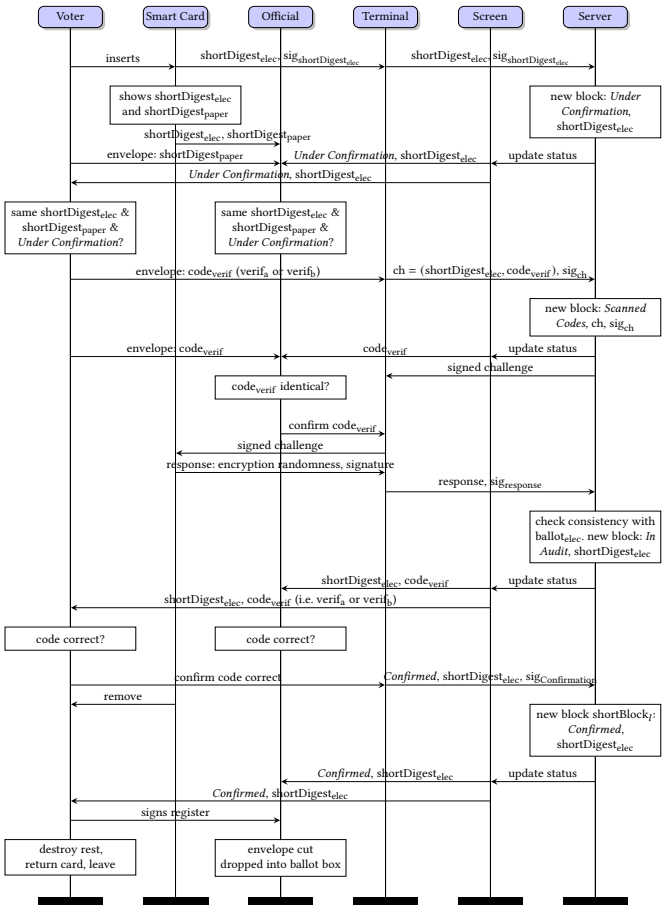



**Figure 3: Interactions at the confirmation point, simplified**

These checks are essential to guarantee the integrity of the election, however the protocol also offers other possibilities for further (optional) audits. For example, one can use risk-limiting audits [22] on the paper ballots to increase confidence in the announced results computed from the electronic ballot box. Moreover, the smart cards and terminals can also be audited. Finally, auditors can pick paper ballots randomly, uncover all hidden values, and audit them to check that they were correctly generated and printed. These audits are destructive: the ballots that have been audited can no longer be used and must be destroyed.

*2.2.6 Dispute resolution.* If the confirmation of the electronic ballot fails, the voters and locals can start the dispute resolution procedure to identify who is responsible for the failure. During the procedure, the paper ballot, the smart card's data, as well as the links to the electronic ballot box are checked.

When auditing all this data, the voter's vote is inevitably leaked, hence the procedure is executed in two parts: the first part preserves the voter's privacy by only revealing part of the data, and can hence

directly be executed in the polling station in case of a problem. The second part of the procedure reveals the vote, but can be executed after the election by external auditors, who do not know the identity of the voter. The detailed procedure is given in Appendix A.

## 2.3 Desired security properties and threat model

The protocol strives to achieve the following security guarantees:

- **End-to-End Verifiability**: If all checks by the voters and the auditors succeed, and the paper ballots used to vote are well-formed, then voters are assured that ballots containing their intended votes are contained in the database, without having to trust the authorities or the system (often called "individual verifiability").
  The correctness of the tallying (often called "universal verifiability") ensures that the final result corresponds to the encrypted electronic ballots. As the tallying is out of scope of this paper, the exact solution will depend on the tallying method chosen, but can be cryptographically guaranteed by the zero-knowledge proofs.
  End-to-End Verifiability typically also covers "eligibility verifiability", i.e., anyone should be able to check that a ballot has been issued by an eligible voter (with at most one ballot per eligible voter). In this protocol, as the eligibility checks are performed by the local authorities, they cannot be audited after the election, as in traditional pen and paper voting.
- **Dispute Resolution including Contestability and Defensibility**: In case a verification fails, the protocol provides procedures to identify the causes for this failure, or at least recover a consistent state, enabling both an honest supplier and honest authorities to defend themselves against false accusations (defensibility), as well as honest voters and observers to objectively incriminate the system or the authorities if they misbehave (contestability).
  Unfortunately it is not always possible to assign blame unambiguously, for example if the voter's envelope contains an invalid paper ballot, this can be caused by the voter messing with their ballot, or by somebody else inserting false ballots into the stack of ballots at the registration point. In these cases we provide a minimal set of possible "blames".
- **Vote Privacy**: As long as the cards, machines and the printing authorities are honest, and only official ballots are used, the local authorities or outside observers cannot break privacy. Moreover, without help of the local authorities or other local observers, the cards and machines cannot break privacy either, meaning that no stakeholder alone can break privacy. However, if there is collusion between the devices and locals, privacy might not be guaranteed in all scenarios.

Note: interstingly, our voting protocol can be downgraded to a standard paper balot system in case the entire electronic system crashes, or if attackers (e.g., voters) steal or destroy smart cards (then pretending the card does not work). In that case, the resulting security is exactly the security offered by a standard paper ballot system, namely vote privacy (with the additional assumptions listed above), no accountability, and verifiability, assuming that local authorities are honest or constantly supervised by honest observers.

# 3 MODEL

The Themis protocol is complex in the sense that it involves many participants with different roles, and the threat model must assign a variety of honest/dishonest assumptions for each security goal. At the same time, the cryptographic building blocks are standard. The difficulty of the security analysis therefore lies in the modeling of the scheduling of the interactions, and the numerous cases to study, making manual analysis error-prone. This is the realm where automated tools based on formal methods are of great help. Among off-the-shelf, well-studied, reliable tools, we chose ProVerif, which is well-suited to our case, in terms of balance between expressivity and automation. In the following, we provide an overview, but refer the reader to [7, 9] for a comprehensive description of the tool.

## 3.1 The ProVerif protocol verifier

ProVerif is a software tool that allows to analyze the security of cryptographic protocols. The general setting is the Dolev–Yao model, where messages are abstract bit-strings (without even their size being modeled), the cryptographic primitives are assumed to be perfect (therefore only modeled by their functionality), and finally, the network is fully and actively controlled by the attacker. For example, in ProVerif, asymmetric encryption is modeled by a symbol $\mathsf{enc}(\mathsf{pk}, m, r)$, where $m$ is the cleartext and $r$ is the randomness used for encryption. The public key $\mathsf{pk}$ is derived from the secret key $\mathsf{sk}$ with a function symbol $\mathsf{pk} = \mathsf{pubkey}(\mathsf{sk})$, and finally, the asymmetric decryption is defined by the fact that the following equation is true:

$$\mathsf{dec}(\mathsf{sk}, \mathsf{enc}(\mathsf{pubkey}(\mathsf{sk}), m, r)) = m.$$

Here, $m$ and $r$ and $\mathsf{sk}$ are variables representing bit-strings. Other cryptographic primitives (hash functions, symmetric encryption, signatures, etc) are similarly defined. Concretely, ProVerif supports any cryptographic primitives that can be modeled by convergent equations or rewriting rules.

Messages that are exchanged between the participants are modeled as terms that actors can build from fresh symbols (nonces), cryptographic functions (like above), and non-cryptographic functions (like pairs, or projections) to combine them. For instance,

$$\mathsf{hash}((\mathsf{enc}(\mathsf{pk}, v, r), \mathsf{zkp}(\mathsf{pk}, v, r))),$$

is a term representing the fingerprint of a ballot: the hash value of an encrypted vote, together with a zero-knowledge proof of well-formedness, where $\mathsf{pk}$ (the public key of the election) must have been received by the voter before, $r$ is a fresh nonce, and $v$ is their voting choice.

Each actor is modeled as a process using a variant of the applied pi-calculus. Such a process is a sequence of instructions where the actor can operate on the messages they know using the allowed re-writing rules, and send and receive messages on communication channels. Usually, dishonest actors are not modeled by processes but by the underlying attacker controlling the network. Hence, all their secret are revealed by sending them on a public channel. The attacker can then receive and use them.

Internally, ProVerif models the protocol with Horn clauses, and performs automated reasoning on them. It typically considers an unbounded number of sessions in parallel, thus covering attacks that could be hard to detect manually.

There are two main classes of security properties that ProVerif can verify. First, reachability queries consist in asking if the attacker can reach a certain state in the protocol. This includes *secrecy queries*, whose purpose is to verify whether a value remains secret, or *correspondence queries* which draw relations between elements in the protocol (e.g., if something occurs, then necessarily something else occurred before). To do so, ProVerif defines *events* which are used to identify specific steps in the protocol and, then, state correspondence queries. Second, equivalence properties consist in asking whether the attacker can distinguish between two distinct situations. This is related to the indistiguishability properties that are well-known in cryptography. In our case, this is the key tool to model the voter's privacy.

ProVerif can give three answers to these security questions. The first one, `true`, is that there is no attack. In that case, this is a guaranteed result (up to errors or approximations when the user wrote the model). The second one, `false`, is that there is an attack. In that case, ProVerif provides an example of execution that falsifies the security property. The third one, `cannot be proved`, is when the tool cannot conclude. Indeed ProVerif analyses only the security of a sound over-approximation of the initial process (mainly because the translation in Horn clauses). When this happens the user needs to refine their modeling to avoid such approximations; this can be done by examining the *false* witness of attack that ProVerif often returns. Finally, it may also occur that ProVerif does not terminate.

In Section 4, we explain a few modeling subtleties that had to be done to obtain a clear "no attack" answer to all our queries.

## 3.2 Overview of Themis modeling in ProVerif

Modeling the Themis protocol in ProVerif is a challenging task for various reasons. First, this is a protocol with many different roles, and with long instructions for them, especially when we consider the dispute resolution. Second, the protocol involves paper, envelopes, scratchcards, for which the modeling is not immediate. Third, we target several properties, and for each of them, we want to be able to be as precise as possible about the trust assumptions. Furthermore, the resulting model is inevitably close to the limit of what can be handled by the tool, making it sensitive to phenomenons where innocent-looking changes lead to non-termination.

For each trust assumption, a new ProVerif file must be designed, since the process of dishonest actors must be disabled to allow the attacker to play their role. We start with a description of our modeling where all the participants are honest. We model each role by a process expressed as a sequence of instructions. As an example, in Figure 4, we give a small excerpt of the modeling of the voter, of the smart card, and of the voting terminal, during the phase where the voter is in the voting booth.

In this example, we see how the communications are expressed in ProVerif, with the `in` command to read on a channel and the `out` command to write on it. Hence, the first line of the voter part sends data on the `c_voter_vTerm` channel, and the first line of the terminal part receives this data. We can remark the following points, which are typical of our modeling choices.

- All the interaction between the agents is modeled through channels, including non-electronic communications. The `c_voter_vTerm` channel models the scanning capability of the terminal that can therefore read the paper ballot presented by the voter, and, in the other direction, the terminal screen seen by the voter. This all occurs in the voting booth, and therefore this channel is assumed to be private unless one of the parties is corrupted. Similarly the `c_card_vTerm` channel models the communication between the smart card and the terminal when the card is inserted in the terminal's card reader. Again, this is a private, unless the card or the terminal is corrupted.
- All communications are prepended with a message header that identifies a precise place in the protocol. This modeling matches a good security practice and greatly improves ProVerif accuracy without altering the model. Indeed, these headers are public constants. The attacker can thus modify them for all the communications on public channels.
- Cryptographic verifications are gathered in function macros. In our example, `CheckQRCodesData` is a function macro that contains two verifications of cryptographic signatures and a verification of an arithmetic property. As presented in Section 3.1, cryptographic signatures (and zero-knowledge proofs, for instance) are idealized (as usual) by rewriting rules. However, for the arithmetic property, the situation is more complicated, and we refer to the next section for this.
- The role of Observers, described in Section 2.2.5 is not modeled in a classical way with a process. We used the *restrictions* capabilities of ProVerif to directly write consistency properties that are guaranteed by the presence of Observers. Each of them corresponds to a verification procedure of the Observers, for which a failure leads to a clear blame. For instance, there is a restriction to ensure that each $\text{shortDigest}_{\text{elec}}$ is accepted only once by the server:

```
restriction SD_elec, Ballot_elec1, Ballot_elec2;
    event(Unicity(SD_elec,Ballot_elec1))
 && event(Unicity(SD_elec,Ballot_elec2))
==> Ballot_elec1=Ballot_elec2.
```

As usual in ProVerif, the roles of honest participants are instantiated so that an unbounded number of agents can vote and/or revote. In order to study a scenario where some entities are assumed to be dishonest, the corresponding roles are not instantiated; instead the attacker will play their role. For this, the secret data of these entities (for instance, their signing key) is made public by out-ing them on a public channel. Furthermore, the channels where one of the communicating entities is corrupted are also made public.

Our approach to keep track of the changes made for each scenario is to derive all the ProVerif files from one master file `protocol.pv` that describes the all-honest scenario. Then, for each scenario, an additional file is provided, that describes the security property that is under study, and the list of honest entities that is sufficient for the property to hold. A small tool automatically combines the master file and this scenario description to create a specific ProVerif file that can then be analyzed by the ProVerif tool.

Voting terminal:

```
let Voting_terminal(...) =
  ...
  in(c_voter_vTerm, (=ct_Ballot, Ballot:bitstring));
  let (...) = ReadQRCodes(Ballot) in
  let (=true) = (CheckQRCodesData(...)) in
  let Name:candidate = getName(Id_cand) in
  !out(c_voter_vTerm, (ct_Name, Name, ...)) |
  in(c_voter_vTerm, (=ct_Choice,=true));
  !out(c_card_vTerm, (ct_BData, SN_paper,...)) |
  in(c_card_vTerm, (=ct_Request_SD_elec1, SD_elec, ...));
  let (=smartcard) = (PKIrole(PKI_card)) in
  !out(c_vTerm_serv, (ct_Request_SD_elec2, SD_elec)) |
  ...
```

Voter:

```
let Voter(...) =
  ...
  !out(c_voter_vTerm, (ct_Ballot, Ballot)) |
  in(c_voter_vTerm, (=ct_Name, Name, ...));
  let (...) = ReadBallot(Ballot) in
  if Name <> BltName || (Verif_a<>BltVerif_a || ...)
  then
    ...
  else
    !out(c_voter_vTerm, (ct_Choice, true)) |
  ...
```

Smart card:

```
let SmartCard(...) =
  ...
  in(c_card_vTerm, (=ct_BData, SN_paper, ...));
  let (=true) = (CheckQRCodesData(...)) in
  let (SD_elec, ...) = CreateEBallot(SN_paper, ...) in
  !out(c_card_vTerm, (ct_Request_SD_elec1, SD_elec, ...) |
  ...
```

**Figure 4: Excerpt of our ProVerif code. Lines that correspond to communications have been colored, using the same color for matching send and receive commands.**

# 4 SECURITY ANALYSIS

A comprehensive analysis of the Themis protocol has been conducted using ProVerif. The claimed security properties are verifiability, vote privacy, and partial accountability. For these three properties, we explain how we model them in our setting.

In Tables 1, 2 we present the results for verifiability and vote secrecy. Details about accountability are discussed in Section 4.3. The ProVerif files are available at: https://bit.ly/3rYCSFR.

## 4.1 End-to-end verifiability

Classically, verifiability can be decomposed in 4 sub-properties: cast-as-intended, recorded-as-cast, counted-as-recorded, and eligibility.

In our context, the cast-as-intended property will actually be considered after the confirmation point, which is the place where the audit takes place; therefore, this contains both cast-as-intended and recorded-as-cast (assuming that the global election screen is consistent with the hashchain, i.e., the electronic ballot-box). Hence, we actually consider the combined property, recorded-as-intended: if a voter thinks they have voted for X then there is indeed a recorded ballot that encrypts X. ProVerif can easily proves this property in a *non injective* way (NI-recorded-as-intended). However, two honest voters should not be associated to the same recorded ballot. In other words, we also need to check that the attacker cannot make two voters happy confirming the same ballot; this is called a clash attack in the terminology introduced by Küsters *et. al.* [21].

Counted-as-recorded requires a verification procedure for the tallying process, ensuring that this part is correctly executed. As the tallying is left unspecified in Themis, we assume a classical solution with decryption trustees, possibly with a threshold, and the use of verifiable mixnets or homomorphic encryption. Then, zero-knowledge proofs can be added as usual to guarantee that this tallying step is correct. Thus, this property is not analyzed here.

Since Themis is an on-site voting system, most of the eligibility issues are dealt with in a non-electronic manner, at the entrance of the polling station. We thus only need to prove that there is no ballot stuffing by the system. Even if this property could be ensured by comparing the number of ballots in the electronic ballot box and the number of signatures on the register, we would like to go one step further and prove that in some scenarios, the property is ensured by design and, by consequence, counting the number of ballots is not the only safeguard.

To summarize, due to the specificities of the Themis protocol, proving end-to-end verifiability boils down to proving the following three properties:

- NI-recorded-as-intended: if Alice thinks she has voted for candidate $X$, then she confirmed an electronic ballot corresponding to $X$ at the confirmation desk;
- No-clash-attacks: two voters (Alice and Bob) cannot agree on the same electronic ballot.
- No-ballot-stuffing: recorded ballots correspond to legitimate voters only, and there is at most one electronic ballot per voter.

*Main difficulties.* In order to achieve verifiability, the Themis protocol relies on two peculiarities that make the analyses more complex: probabilistic audits and arithmetic operations.

First, unlike the Benaloh audit mechanism [4], the Themis protocol enforces that the voter makes an audit before confirming their vote, by auditing one of the two ciphertexts. Attacks against verifiability (mainly *recorded-as-intended*) will thus be detected with probability 1/2. Unfortunately, ProVerif is not suitable to conduct probabilistic analyses. We thus decided to model the fact that each verification code is audited with the same probability by the voter using a process in which both codes are audited. This models that an attacker cannot anticipate which code is audited, and hence must be prepared for both audits. This approach allows us to abstract away the probabilities.

Second, the audit mechanism of the Themis protocol relies on three numbers $x, a, b$ such that $x \equiv a + b \bmod n$ for some $n$. Unfortunately, ProVerif does not support the addition in finite fields. To overcome this limitation, we decided to over-approximate it through events and restrictions. More precisely, we define two events $\texttt{isSum}(x, a, b)$ and $\texttt{isNotSum}(x, a, b)$ that are executed each time an agent respectively tests that $x \equiv a + b \bmod n$ and $x \not\equiv a + b \bmod n$. We then define restrictions to model the only few arithmetic properties that are needed to make the protocol secure. For instance, we define:

$$\texttt{isSum}(x, a, b) \wedge \texttt{isSum}(x, a, b') \rightarrow b = b' \quad (1)$$

$$\texttt{isSum}(x, a, b) \wedge \texttt{isNotSum}(x, a, b') \rightarrow b \neq b' \quad (2)$$

The complete set of restrictions is presented in Appendix B.

*ProVerif queries.* Verifiability properties are modeled using reachability queries. For the NI-recorded-as-intended property, this takes

the following (simplified) form:

$$\texttt{event}(\texttt{HappyUser}(Alice, Name, \mathrm{SD}_{\mathrm{elec}}, \mathrm{verif}_{\mathrm{a}}, \mathrm{verif}_{\mathrm{b}})) \\ \&\& \texttt{event}(\texttt{Snaphshot}(Alice, \mathrm{SD}_{\mathrm{elec}}, \ldots)) \\ \&\& \texttt{event}(\texttt{isSum}(\mathrm{cand}, \mathrm{verif}_{\mathrm{a}}, \mathrm{verif}_{\mathrm{b}})) \Rightarrow \\ Name = \texttt{GetName}(\mathrm{cand})$$

where:

- $\texttt{HappyUser}(,,,,)$ is an event raised when a user has confirmed their vote and believes that they voted for the given candidate $Name$ using an electronic identified by $\mathrm{SD}_{\mathrm{elec}}$;
- $\texttt{Snaphshot}(,\ldots)$ models that an electronic ballot identified by $\mathrm{SD}_{\mathrm{elec}}$ has been shown on the general election screen. Thanks to the QR codes allowing for audits, we can assume that this ballot contains consistent data;
- $\texttt{GetName}(\cdot)$ is the function that translates the integer cand that encodes the candidate in the electronic ballot present in the ballot box into the name of the candidate, hopefully the same as the one Alice saw on her paper ballot when she was in the voting booth.

Similarly, the no-clash-attack property will take the form of the following (simplified) query, stating that two honest users cannot confirm the same electronic ballot:

$$\texttt{event}(\texttt{HappyUser}(Alice, Name_A, \mathrm{SD}_{\mathrm{elec}\,A}, \mathrm{verif}_{\mathrm{a}A}, \mathrm{verif}_{\mathrm{b}A})) \\ \&\& \texttt{event}(\texttt{HappyUser}(Bob, Name_B, \mathrm{SD}_{\mathrm{elec}\,B}, \mathrm{verif}_{\mathrm{a}B}, \mathrm{verif}_{\mathrm{b}B})) \Rightarrow \\ \mathrm{SD}_{\mathrm{elec}\,A} \neq \mathrm{SD}_{\mathrm{elec}\,B}$$

Finally, the no-ballot-stuffing property will be modeled by two queries. The first one states that every $\texttt{CardMadeBallot}()$ event corresponds injectively to an event modeling that a voter succeeded the registration step at the entrance of the polling station. The second query tells that whenever the attacker sees an entry on the general screen saying that a ballot has been confirmed and entered the electronic ballot box, then a $\texttt{CardMadeBallot}()$ event was emitted before for this ballot. For space reasons we do not reproduce the queries here; they are available in the ProVerif file(s).

*Results.* The results of this analysis are presented in Table 1. For each scenario we identify the minimal trust assumptions that lead to an attack or provide a proof of security. They can be summed up as follows: verifiability holds as soon as either the Server is honest or there are observers, and either the Printing Authority or the Devices (as a whole) are honest. Intuitively, the second condition ensures that the paper ballot, or the electronic ballot are well-formed and can be trusted by the voter, while the first condition ensures that the global screen corresponds to the hashchain and thus can be trusted by the voter to safely perform their checks at the confirmation desk[1]. Note that if the smart card or the terminals are malicious then the voter must perform a few extra checks to ensure the well-formedness of the paper ballot.

Our *no ballot stuffing* property holds when both the local authorities and the smart cards are not corrupted. When only the local authorities are honest then ballot stuffing is prevented by comparing the number of paper/electronic ballots. However, in a classic on-site setting, as it is the case here, malicious local authorities can always stuff ballots by impersonating existing voters (signing the register and using smart cards on their behalf).

[1]Technically, our ProVerif models assume that the display of the global election screen is consistent with the content of the hashchain. This assumption is met at least when the Server is honest, or observers are present.

## 4.2 Vote privacy

As mentioned in Section 1, vote privacy is encoded using an equivalence property. Here we give more details about its modeling in our case. The results are presented in Table 2.

*Modeling.* Following Kremer *et. al.*'s definition [13], a protocol ensures vote privacy if an attacker cannot see whether two voters, Alice and Bob, exchange their choices.

Because the tally might give away information about the votes, we need to model a tallying functionality. The ProVerif model includes a basic mix and decrypt procedure that mixes the votes of Alice and Bob (only), before it outputs the plaintext of both ballots.

The vote privacy definition considers two scenarios that only differ by terms (the choices of Alice and Bob). In ProVerif, this is encoded by a (bi)process $P$ that contains the specific keyword $\texttt{diff}$, In the first scenario, all the terms of the form $\texttt{diff}[a, b]$ are replaced by $a$, while in the second scenario they are replaced by $b$. Concretely, the classical vote privacy definition would be encoded by the process

$$P = C \mid \texttt{Voter}(Alice, \texttt{diff}[0, 1]) \mid \texttt{Voter}(Bob, \texttt{diff}[1, 0])$$

where $C$ models all the parties different from Alice and Bob.

In the Themis protocol, a ballot does not only include the choice of the voter, but also data for the audit. More precisely, a ballot contains three integers $x$, $a$, and $b$ where $x$ is the voter's choice, and $a$ and $b$ are the two audit codes, such that $x \equiv a + b \mod n$ for some $n$. W.l.o.g. in our modeling we assume that Alice and Bob audit with the first code $a$. Because this value is revealed during the voting process, it must remain the same for each voter in the two scenarios. We thus model vote privacy considering the process

$$\begin{aligned} P = \; & C \\ & \mid \texttt{Voter}(Alice, \texttt{diff}[\texttt{ballot}(x_1, a_1, b_1), \texttt{ballot}(x_2, a_1, b_3)]) \\ & \mid \texttt{Voter}(Bob\;, \texttt{diff}[\texttt{ballot}(x_2, a_2, b_2), \texttt{ballot}(x_1, a_2, b_4)]), \end{aligned}$$

where $C$ models all the parties different from Alice and Bob, $x_1$ and $x_2$ encodes Alice's and Bob's choices, and $a_1, a_2, b_1, b_2, b_3, b_4$ are the audit codes. It is important to note that these codes are well-defined. Indeed, from the properties of modular arithmetic, for all $x_1, x_2, a_1, a_2$, we know that there exists $b_1$, $b_2$, $b_3$, $b_4$ such that

$$x_1 \equiv a_1 + b_1 \equiv a_2 + b_4 \mod n,$$

$$x_2 \equiv a_1 + b_3 \equiv a_2 + b_2 \mod n.$$

*Main difficulty: arithmetic operations.* As discussed, modeling natural numbers and addition in finite fields is a hard task for symbolic tools. However, unlike reachability properties, it is not possible to use events and restrictions to over-approximate the relation. Indeed, soundness issues come up when considering an over- or under-approximated relations for equivalence properties. To overcome this limitation, we follow a different approach: we prove that the $\texttt{isSum}(x, a, b)$ relation is preserved from left to right in the equivalence scenario (and conversely). More formally, thanks to a ProVerif lemma, we prove that, given a bi-trace $\mathrm{tr}_b$,

$$\texttt{isSum}(x, a, b) \in \texttt{fst}(\mathrm{tr}_b) \Leftrightarrow \texttt{isSum}(x, a, b) \in \texttt{snd}(\mathrm{tr}_b) \quad (3)$$

where $\texttt{fst}(\mathrm{tr}_b)$ (resp. $\texttt{snd}(\mathrm{tr}_b)$) is the trace where $\texttt{diff}[a, b]$ is replaced by $a$ (resp. by $b$). This lemma allows us to prove (see Appendix C) the soundness of our analysis.

| Participants | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Observer | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Printing Authority* | | 1 | 0 | 0 | 0 | x | x | x | x | x | x |
| Local Authorities | | x | x | x | x | x | x | x | x | x | x |
| Devices | Smart card | x | 1 | x | 0 | x | 0 | 1 | x | 0 | 1 |
| | Terminals | | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| | Server | | x | x | x | 0 | 0 | 0 | 1 | 1 | 1 |
| Results | | | | | | | | | | | |
| NI-recorded-as-intended | | ✓ | ✓ (5min) | ✗⋆ | ✗⋆ | ✗ | ✗ | ✗† | ✗⋆ | ✗⋆ | ✓ (5min) |
| No clash attacks | | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |

⋆ property proved if the voter verifies $\text{verif}_a \neq \text{verif}_b$ and $\text{id}_{cand} = \text{verif}_a + \text{verif}_b$

† property proved if the display of the global election scree is consistent with the content of the hashchain

* assuming that voters use honestly generated ballots if honest printer

| Participants | | | | |
|---|---|---|---|---|
| Observer | | x | x | x |
| Printing Authority | | x | x | x |
| Local Authorities | | 0 | 1 | 1 |
| Devices | Smart card | x | x | 1 |
| | Terminals | x | x | x |
| | Server | x | x | x |
| Results | | | | |
| No ballot stuffing | | ✗ | ✓⋆ | ✓ |

⋆ requires to compare the number of paper and electronic ballots to prevent malicious additions/deletions. Cannot be proved in ProVerif

1: honest ✓: proved secure
0: dishonest ✓: expected to hold, but not proved in ProVerif
x: honest OR dishonest ✗: attack

**Table 1: Security analyses w.r.t. verifiability properties (times omitted if less than a few seconds)**

Theorem 4.1. *Let $P_{priv}$ be the configuration that encodes* vote privacy. *If $P_{priv}$ satisfies Equation (3) and* diff-equivalence *then vote privacy holds w.r.t. arithmetic operations.*

*Results.* Results are presented in Table 2. First note that in general a single corrupted entity cannot break the privacy alone, except for a special case: an attack exists if the local authorities or the printer are able to forge fake paper ballots for a targeted candidate and give them to a targeted voter. Indeed, if a voter scans this malformed ballot to the voting terminal, it will be rejected and the voter will be sent back to the registration desk to pick up a different ballot and re-vote. This abnormal behavior will thus be visible, and breaks vote privacy as an observer knows that the voter wanted to vote for the targeted candidate. The same kind of attacks exist when assuming a malicious printer who colludes with an accomplice to be able to target a specific voter. Luckily, these attacks can be prevented by using physical countermeasures that make the creation of fake ballots difficult (e.g., the use of watermarks, special or colored papers, etc.) and audits. These last should be designed to ensure that voters get (with high probability) well-formed paper ballots.

Remark 1. *We do not provide a ProVerif proof for the scenario in which the Printing Authority is malicious, but without the help of a local accomplice. Indeed, in this scenario, the symbolic proof trivially holds: since there is no attacker in the polling station and ballots do not contain any data that can be used to associate a specific voter to a given ballot, the attacker cannot distinguish whether a vote or ballot is submitted by Alice or Bob. Hence, vote privacy trivially holds.*

| Participants | | Dis. local | | Dis. printer | | |
|---|---|---|---|---|---|---|
| Printing Authority | | 1 | 1 | 0 | 0 | 0 |
| Local | Authorities | 0 | 0 | 1 | 1 | 1 |
| | Accomplice | x | x | 1 | 0 | 0 |
| Devices | Smart card | 1 | 1 dis. | x | 1 dis. | 1 |
| | Voting term. | 1 | | x | | 1 |
| | Confirm. term. | 1 | | x | | 1 |
| | Server | x | x | x | x | x |
| Results | | | | | | |
| Privacy | | ✓⋆ (3h57) | ✗ | ✓ | ✗ | ✓⋆⋆ (47min) |

⋆ assume that local authority cannot forge fake ballots.
⋆⋆ assume random audits to detect fake paper ballots in the stack.
1: honest (absent) ✓: proved secure
0: dishonest (present) ✗: attack
x: honest OR dishonest

**Table 2: Security analysis w.r.t. vote privacy**

## 4.3 Accountability

Accountability can be split in two parts: first, *contestability*, meaning that if something goes wrong, the voters will be able to enter a branch of the protocol that ends-up in blaming the entity responsible for the problem; second, *defensibility*, meaning that voters should not have the possibility to wrongly accuse the system, if it behaved honestly.

We therefore prove the following:

- defensibility: a participant is never wrongly blamed;
- contestability: a voter cannot be stuck in the voting process. Either their vote is confirmed, or they can return to the booth to vote again, or the dispute resolution is executed (which means that somebody will be blamed).

Our dispute resolution however does not always identify a single party but sometimes a group of parties, among which one is guilty. In that sense, Themis only offers partial accountability.

In our protocol, some illegal behaviors from voters may lead to the confiscation of a smart card, that is then removed from the

pool of cards of the voting station. Since this could be abused by malicious voters, we additionally analyze the exact situation where this could occur, so that it can be mitigated using practical means:

- card-capture resistance: Whenever a smart card is captured, either the system is malicious[2] or the voter (or the printer) managed to forge a fake paper ballot with certain properties.

*Defensibility.* Verifying the *defensibility* property consists in proving the correctness of the blames that occur in the protocol, i.e., ensuring that an honest participant cannot be blamed. To do so, each time an agent $X$ is claimed guilty in the protocol, an event `blame(X)` is added in the ProVerif model. Sometimes, the protocol is not able to decide whether the culprit is agent $X$ or agent $Y$. In this case, we define the event `blame(X ∨ Y)`.

Given an agent $X$, the correctness of the event `blame(X)` is modeled by the unreachability of this event when considering a scenario in which everyone but $X$ is malicious. This definition is extended to the events `blame(X ∨ Y)` by considering scenarios in which everyone but $X$ and $Y$ is malicious.

*Contestability.* The *contestability* property is a correspondence property which is encoded with a ProVerif query as expected. More precisely, ProVerif proves the following query considering a scenario in which everyone but the voter is malicious:

$$\texttt{event(end\_of\_voter\_process)} \Rightarrow \texttt{event(HappyUser}(\mathit{Voter}, \text{cand}, \text{SD}_{\text{elec}}, \text{verif}_{\text{a}}, \text{verif}_{\text{b}})) \;||\; \texttt{event(return\_to\_Booth)} \;||\; \texttt{event(complain\_for\_dispute)}.$$

This property is weaker than true contestability since it does not say that a party will be blamed. Ideally, we would like to prove that each time a voter launches a dispute resolution procedure (event `complain_for_dispute`), a party is eventually blamed. However, ProVerif cannot prove such liveness properties. Instead, we can manually inspect the dispute resolution procedure and check that each branch ends with at least one blame. We conclude by noting that, in real life, a voter will not accept to be locked in the process (just because, in practice, they will ask to continue).

According to our protocol specification, each time a voter enters in dispute resolution, a smart card is confiscated, which means that the following property holds:

$$\texttt{event(end\_of\_dispute)} \Rightarrow \texttt{event(cardCaptured(SD}_{\text{paper}})).$$

*Card-capture resistance.* *Card-capture resistance* is a correspondence property which ensures that a malicious voter will not be able to break an election process by capturing all the smart cards. More precisely, it characterizes how such a voter must behave to reach their goal.

First, we prove that whenever a smart card is captured, either the system is malicious or a fake paper ballot has been used. Formally, we verify the following query considering scenarios in which everyone but the voter and the printer is honest:

$$\texttt{event(cardCaptured(SD}_{\text{paper}})) \Rightarrow \texttt{event(fake\_paper\_ballot}(\mathit{ballot}_{\mathit{paper}})) \;\&\&\; \texttt{getSD}(\mathit{ballot}_{\mathit{paper}}) = \text{SD}_{\text{paper}}$$

where $\texttt{getSD}(\mathit{ballot}_{\mathit{paper}})$ is a function symbol that extracts the short digest paper that occur in the ballot $\mathit{ballot}_{\mathit{paper}}$.

Second, we characterize the shape of fake paper ballots that can cause a dispute resolution procedure. This description can be used to detect and prevent the use of such ballots adding physical counter measures or guiding the random audits of paper ballots from the stack of ballots. In ProVerif, we prove a query of the form:

$$\texttt{event(fake\_paper\_ballot}(\mathit{ballot}_{\mathit{paper}})) \Rightarrow \text{inconsistent human readable data} \vee \text{inconsistent QR codes} \vee \text{invalid signatures} \vee \ldots$$

Third, we prove that an honest voter, with the help of an honest voting terminal or random audits of the stack of paper ballots, will never use a fake paper ballot by accident. We prove that the event $\texttt{fake\_paper\_ballot}(\mathit{ballot}_{\mathit{paper}})$ is not reachable when considering a scenario in which everyone but the voter and the voting terminal is malicious, or everyone but the voter is malicious and the voter gets their ballot from a stack of well-formed paper ballots.

*Results.* Contestability and card-capture resistance have been proved as expected, in the scenarios as presented above. Regarding defensibility, 9 scenarios have been studied to cover the 17 exit cases of the dispute resolution procedure (some cases contain the same blames)[3]. Among them, 10 lead to individual blames (i.e uniquely identify the culprit(s)), 2 reduce the set of possible culprits to only two agents (but further investigations are needed to decide who is the real culprit), and 5 do not allow to precisely identify the culprit(s). These 5 last cases are related to fake paper ballots that can be introduced in the process by many participants. In particular, they can be produced by a malicious printer, a malicious voter, or malicious locals manipulating the stack of paper ballots. Moreover, in some cases introducing a fake paper ballot requires to corrupt devices to avoid being detected. The resulting blame is thus:

$$\texttt{blame(voter} \vee \texttt{(card} \vee \texttt{terminal)} \wedge \texttt{(stack} \vee \texttt{printer))}$$

where *stack* means that the stack of paper ballots contained invalid ballots. Although our ProVerif analysis is unable to provide a more precise blame in this case, in practice one can analyze the physical properties of the paper ballot to more precisely determine the culprit (for example, a voter is unlikely to produce a fake paper ballot with exactly the same type of paper as the printer, etc.). Also note that the third property of card-capture resistance shows that it is not that easy to reach this case within the dispute resolution procedure.

All files are analyzed in a few seconds, except one (card capture resistance) that takes 117min.

# 5 EXPERIMENTS

## 5.1 Smart card implementation

The smart-card needs to perform a lot of computations: verification of signatures, encryption of the vote, creation of the proofs and signature of the ballot. In order to minimize the waiting time for the voter, we use the initialization step to perform computations

[2]The system is made of the devices and the local authorities – note that these agents do not need to capture smart cards to crash an election.

[3]Compared to its description in Appendix A, some cases of the dispute resolution are split into several sub-cases in ProVerif.

independent from the voter's choice in advance. It turns out that a significant part can be pre-computed.

Table 3 summarizes the main computations to be done on the card at the different stages of the process, in terms of random number generation (RND bytes), scalar multiplications (SM), writing in non-volatile memory (STR bytes), hashing (H bytes) and data to transmit (COM bytes). We assume that the asymmetric encryption and the corresponding ZKPs are based on an elliptic curve over a prime finite field with 256 bit, thus providing the standard 128-bit security level. The bottleneck lies in the generation of the ZKPs proving the arithmetic properties between the encrypted vote and the encrypted audit data. This can be restated as proofs of disjunctions of equalities, for which we use classical ZKPs, like, for instance, in Helios. On the other hand, at the confirmation step, the only cryptographic task made by the card is a signature verification, hence 2 SM. On a modern smart-card component, equipped with an ARM Cortex-M3 secure core and with dedicated hardware for fast modular arithmetic, and for 20 candidates, the timing are estimated at less than 10 seconds for the initialization step, and less than 5 seconds for the computations done in the booth. In the context of voting, these numbers are fully acceptable.

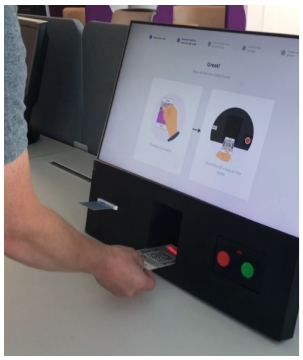
| | $n/2$ candidates | 20 candidates | |
|---|---|---|---|
| Initialization | $96n + 224$ | 4,064 | RND |
| | $3n + 10$ | 130 | SM |
| | $240n + 704$ | 10,304 | STR |
| Booth | $(3/2)n + 5$ | 65 | SM |
| | $192n + 1152$ | 8,832 | H |
| | $96n + 560$ | 4,400 | COM |
| Confirmation | 2 | 2 | SM |

**Table 3: Summary of computations by the smart card, for each voter. RND: number of random bytes; SM: number of elliptic scalar multiplications; STR: number of bytes to write in non-volatile memory; H: number of bytes to hash; COM: number of bytes to transmit.**

## 5.2 User Experience

Usability is a strong concern to ensure that the voter can go through all the required steps defined in this protocol. Our User Experience test aimed at assessing how the user reacts and (mis-)understands the process at different stages during the definition of the protocol. In addition, our experiment had the following goals:

- Challenge the benefit of the technical proposition;
- Get an overall feedback on the voting protocol and user behavior;
- Investigate how users felt about confidentiality of their votes.



Our testing method is based on a simplified prototype, which simulates technical operations such as cryptographic operations, but tries to provide a realistic user experience. Tests were qualitative at this stage, and performed against a small number (7) of users with a mix of people between 23 and 40 years old, men and woman alike. Additional testing with 7 persons focused more specifically on the user experience in the booth with a more advanced prototype (the figure depicted on the right depicts the voting terminal interface). Voters were recruited within the IDEMIA company. Admittedly, they are not representative from the general population. Moreover, we did not test user behaviours in an adversarial setting where local authorities or smart cards would try to tamper with voter ballots, in order to see whether voters would be able to detect this and react adequately. Conducting a thorough user study is left as future work.

Figure 5 presented in appendix shows the prototype at an early stage during the definition of the protocol.

The main results are:

(1) Most users (more than 80%) understand what they have to do in the booth, which is the only part where voters are left on their own. The main difficulty was encountered at the confirmation terminal, where the auditing with the $\text{verif}_a$ or $\text{verif}_b$ code takes place. This could be mitigated with further clear and intuitive indications.
(2) While most users (more than 80%) globally understand what they have to do during this voting journey, they do not understand why they have to follow such a process, in particular the complexity of the ballot and all the verification steps involving numbers displayed on the ballot. Some of those users are confused by the presence of paper ballots since the vote is electronic. Justification of the complexity introduced by this protocol is not well understood.
(3) Most users (more than 70%) have a doubt about the confidentiality of their vote, for the following reasons:
    - numbers being visible through the envelope window;
    - some information related to their vote is visible on the public screen.

Points (2) and (3) show that improving the user experience during the voting journey would not solve all the doubts that users may have in this protocol regarding usefulness and confidentiality with the limited explanations provided during the experiment. A specific communication campaign (not in the scope of this UX study) would be required prior to the voting process, to help voters gain confidence in this voting system, and to answer any question they could have, whether they are familiar to technology or not.

# 6 CONCLUSION

We presented a novel on-site voting system aiming at improving security when local authorities are not fully trusted. The system produces both an electronic and a paper ballot box, with the particular feature that all electronic ballots are systematically audited. It also includes a dispute resolution procedure identifying misbehaving parties. The system has been formally analyzed in ProVerif, which includes a novel approach to model the modular arithmetic used in the protocol. We also conducted a small usability experiment.

As future work, it would be desirable to further improve the dispute resolution procedure to always be able to uniquely assign blame, and to improve the user experience given the feedback from our small user study.

## REFERENCES


[1] Ben Adida. 2008. Helios: Web-based Open-Audit Voting. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, Paul C. van Oorschot (Ed.). USENIX Association, 335–348. http://www.usenix.org/events/sec08/tech/full_papers/adida/adida.pdf

[2] Andrew Appel, Richard DeMillo, and Philip Stark. 2019. *Ballot-Marking Devices (BMDs) Cannot Assure the Will of the Voters.* Technical Report. SSRN. Available at SSRN: https://ssrn.com/abstract=3375755.

[3] David Basin, Saša Radomirović, and Lara Schmid. 2020. Dispute resolution in voting. In *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*. IEEE, 1–16.

[4] Josh Benaloh. 2006. Simple Verifiable Elections. In *Voting Technology Workshop (EVT'06)*.

[5] Josh Benaloh, Michael D. Byrne, Bryce Eakin, Philip T. Kortum, Neal McBurnett, Olivier Pereira, Philip B. Stark, Dan S. Wallach, Gail Fisher, Julian Montoya, Michelle Parker, and Michael Winn. 2013. STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. In *EVT/WOTE*.

[6] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. 2017. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *IEEE Symposium on Security and Privacy (S&P'17)*. 483–502.

[7] B. Blanchet. 2001. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Computer Society Press, 82–96.

[8] Bruno Blanchet. 2016. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends in Privacy and Security* 1, 1-2 (2016), 1–135.

[9] Bruno Blanchet, Vincent Cheval, and Véronique Cortier. 2022. Machine-Checked Proofs of Privacy for Electronic Voting Protocols. In *42nd IEEE Symposium on Security and Privacy (S&P'22)*.

[10] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. 20008. Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems using Invisible Ink Confirmation Codes. In *USENIX/ACCURATE EVT*.

[11] Michael Clarkson, Stephen Chong, and Andrew Myers. 2008. Civitas: Toward a Secure Voting System. In *IEEE Symposium on Security and Privacy (S&P'08)*. IEEE Computer Society.

[12] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. 2019. Belenios: A Simple Private and Verifiable Electronic Voting System. In *Foundations of Security, Protocols, and Equational Reasoning (LNCS)*, Vol. 11565. Springer, 214–238.

[13] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. 2009. Verifying Privacy-type Properties of Electronic Voting Protocols. *Journal of Computer Security* 17, 4 (July 2009), 435–487. https://doi.org/10.3233/JCS-2009-0340

[14] Rolf Haenni, Reto E. Koenig, Philipp Locher, and Eric Dubuis. 2017. CHVote System Specification. Cryptology ePrint Archive, Report 2017/325.

[15] Douglas W. Jones. 2005. Threats to Voting Systems. https://homepage.divms.uiowa.edu/~jones/voting/nist2005.shtml.

[16] Ari Juels, Dario Catalano, and Markus Jakobsson. 2005. Coercion-Resistant Electronic Elections. In *ACM Workshop on Privacy in the Electronic Society (WPES'05)*. ACM.

[17] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. 2015. End-to-End Verifiable Elections in the Standard Model. In *Advances in Cryptology (EuroCrypt'15) (Lecture Notes in Computer Science)*, Vol. 9057. Springer, 468–498.

[18] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. 2017. Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach. In *IEEE European Symposium on Security and Privacy (EuroS&P'17)*. IEEE, 435–450.

[19] Nadim Kobeissi, Georgio Nicolas, and Karthikeyan Bhargava. 2019. Noise Explorer: Fully Automated Modeling and Verification for Arbitrary Noise Protocols. In *4th IEEE European Symposium on Security and Privacy (EuroS&P'19)*.

[20] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2010. Accountability: definition and relationship to verifiability. In *Proceedings of the 17th ACM conference on Computer and communications security*. 526–535.

[21] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2012. Clash Attacks on the Verifiability of E-Voting Systems. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 395–409. https://doi.org/10.1109/SP.2012.32

[22] M. Lindeman and P. B. Stark. 2012. A Gentle Introduction to Risk-Limiting Audits. *IEEE Security Privacy* 10, 5 (2012), 42–49.

[23] Karola Marky, Oksana Kulyk, Karen Renaud, and Melanie Volkamer. 2018. What Did I Really Vote For? On the Usability of Verifiable E-Voting Schemes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173750

[24] Peter Ryan, Peter Rønne, and Vincenzo Iovino. 2016. Selene: Voting with Transparent Verifiability and Coercion-Mitigation. In *Financial Cryptography Workshops 2016*. 176–192.

[25] Peter Y.A. Ryan and Steve Schneider. 2006. Prêt à Voter with re-encryption mixes. In *11th European Symposium on Research in Computer Security (Esorics'06)*. 313–326.

[26] Douglas Wikström. 2022. The Verificatum Mixnet. https://www.verificatum.org/.


## A DISPUTE RESOLUTION PROCEDURE

When the confirmation fails, a dispute resolution procedure is started to identify the participant responsible for the failure. In some cases this might not be possible directly, and in that case the evidence produced by the procedure can be analyzed later on.

The goal is to ensure that an honest participant will not be blamed incorrectly (defensibility), and that they can produce convincing evidence in case some other participant misbehaved (contestability).

The dispute resolution procedure is invoked if the last step of the confirmation fails, or if the card refuses to provide the required audit data. Hence the protocol is in a state where the verification code visible trough the window of the envelope does *not* correspond to the verification code shown on the general election screen for the ballot currently in state *In Audit*, or where the card failed to respond to the audit challenge for a ballot currently in state *Under Confirmation* (hence there is not yet any audit data on the general election screen).

As a first step, the official and the voter at the confirmation point tell the terminal to start the dispute resolution procedure.

The terminal sends a message containing the short ballot digest $\text{shortDigest}_{\text{elec}}$ of the ballot in question, a request to change the state to *Disputed* for the ballot in question, and a signature on both.

- If the signature is valid and the $\text{shortDigest}_{\text{elec}}$ corresponds to a ballot currently in state *In Audit*, the server creates a new block which is added to the electronic ballot box, to change the ballot's state. The server also computes a short block and a signature on it. The short block and the signature are then shown on the general election screen, where the short electronic ballot digest and the state are shown in plain, and the rest is contained in QR-codes. The short block, the hash and the signature are sent back to the terminal.
- Otherwise, the server responds with an error message.

Then, the following checks are done (in this order) using a trusted interface to the card and a trusted device for the cryptographic checks:

(1) *Verification of the smart card's screen.* Because of the previous checks, the card should be in a state where
   (a) the short electronic ballot digest shown by the card corresponds to the short electronic ballot digest shown on the general election screen for the ballot currently in state *Disputed*
   (b) the short paper ballot digest shown by the card corresponds to the short paper ballot digest visible trough the window of the envelope

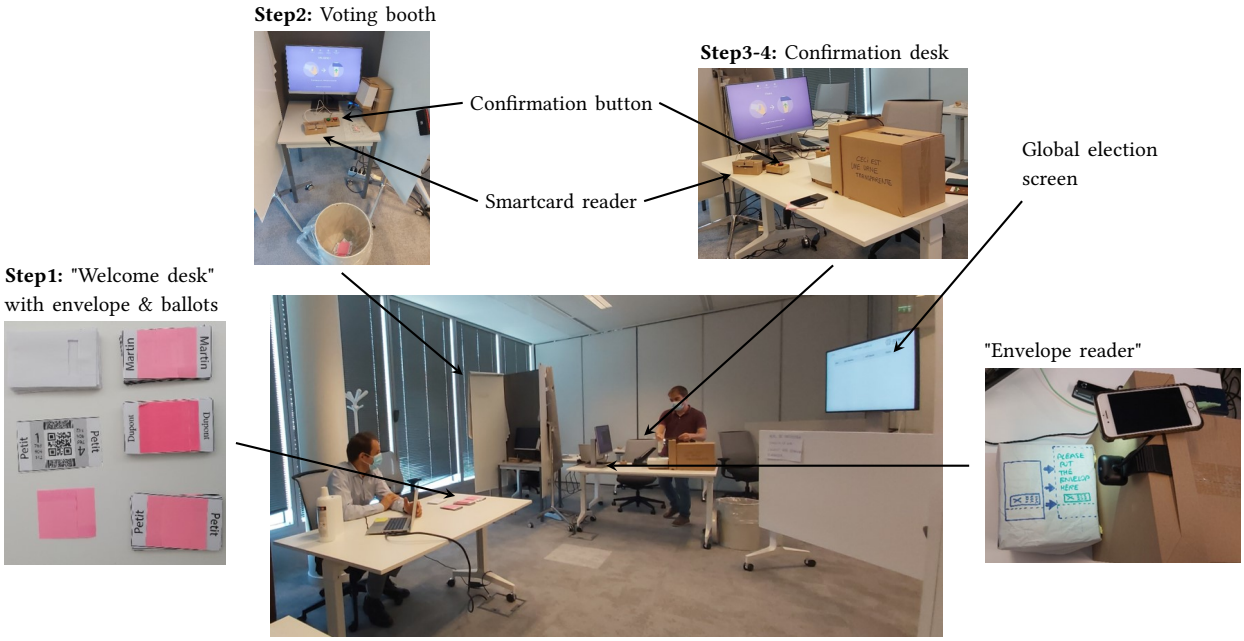



**Figure 5: Global view of the user journey tested**

If (1a) is not the case, the card is declared responsible. If (1b) is not the case, either the card, or both the voter and the official are declared responsible.
In both cases the server is told (by the user and official via the terminal) to add a corresponding block to the electronic ballot box: The terminal sends a message containing the short ballot digest $\text{shortDigest}_{\text{elec}}$ of the ballot in question, a request to change the state to *Incorrect Identifiers on Display* for the ballot in question, and a signature on both.

- If the signature is valid and the $\text{shortDigest}_{\text{elec}}$ corresponds to a ballot currently in state *Disputed*, the server creates a new block which is added to the electronic ballot box to change the ballot's state. The server also computes a short block and a signature. The short block and the signature are then shown on the screen, where the short electronic ballot digest and the state are shown in plain, and the rest is contained in QR-codes.
- Otherwise, the server responds with an error message.

The event is also added to the polling station's paper logbook, the card is secured and kept for further analysis (if required). The server also removes the card's public key from its internal list of accepted cards.

(2) *Verification of the paper ballot.* In this step, the envelope is partly opened to reveal the small QR-code containing the ballot's serial number $\text{SN}_{\text{paper}}{}^{ballot}$, the random $\text{rand}_{\text{paper}}{}^{ballot}$, and the signature $\text{sig}_{\text{SN}_{\text{paper}}}{}^{ballot}$ on these values. No other data should be revealed in order to ensure the privacy of the vote. This can be realized, e.g., using a special window in the envelope which can be opened if necessary. The QR code is scanned by the terminal and it checks whether

- the signature by the printing authority is valid
- the short paper ballot digest $\text{shortDigest}_{\text{paper}}$ it scanned previously actually corresponds to the truncated hash of the serial number and the random

If this is not the case, the paper ballot is incorrect, which could have been caused either (1) by the voter fabricating and inserting an incorrect ballot into the envelope in the voting booth, or (2) by somebody inserting incorrect ballots into the pile(s) AND the card or the terminal accepting them (this includes the printer!). In any case, the server is told (by the user and official via the terminal) to add a corresponding block to the electronic ballot box: The terminal sends a message containing the short ballot digest $\text{shortDigest}_{\text{elec}}$ of the ballot in question, a request to change the state to *Inconsistent Paper Ballot* for the ballot in question, and a signature on both.

- If the signature is valid and the $\text{shortDigest}_{\text{elec}}$ corresponds to a ballot currently in state *Disputed*, the server creates a new block which is added to the electronic ballot box, to change the ballot's state. The server also computes a short block and a signature. The short block and the signature are then shown on the screen, where the short electronic ballot digest and the state are shown in plain, and the rest is contained in QR-codes.
- Otherwise, the server responds with an error message.

As there is a realistic chance that the card behaved incorrectly, both the ballot and card are secured and kept for further analysis (if required). The server also removes the card's public key from its internal list of accepted cards. Finally, the event is also added to the polling station's paper logbook.

(3) *Verification of the smart card's data.* In this step, the trusted interface device sends the server's signature on the dispute block to card, which in return reveals the scan data (the ballots serial number $SN_{paper}{}^{card}$, the random $rand_{paper}{}^{card}$, and the printing authority's signature $sig_{SN_{paper}}{}^{card}$), and the random $rand_{elec}{}^{card}$ it used to compute the short electronic ballot digest.
The terminal then verifies if
- the signature $sig_{SN_{paper}}{}^{card}$ is valid for the given serial number $SN_{paper}{}^{card}$ and random $rand_{paper}{}^{card}$
- the short paper ballot digest $shortDigest_{paper}$ it scanned previously actually corresponds to the truncated hash of the serial number $SN_{paper}{}^{card}$ and random $rand_{paper}{}^{card}$
- the short electronic ballot digest $shortDigest_{elec}$ it received previously actually corresponds to the truncated hash of the serial number $SN_{paper}{}^{card}$ and random $rand_{elec}{}^{card}$

If this is not the case, the card misbehaved, and the server is told to add a corresponding block to the electronic ballot box: The terminal sends a message containing the short ballot digest $shortDigest_{elec}$ of the ballot in question, a request to change the state to *Inconsistent Card Data* for the ballot in question, and a signature on both.
- If the signature is valid and the $shortDigest_{elec}$ corresponds to a ballot currently in state *Disputed*, the server creates a new block which is added to the electronic ballot box, i.e., to change the ballot's state. The server also computes a short block and a signature. The short block and the signature are then shown on the screen, where the short electronic ballot digest and the state are shown in plain, and the rest is contained in QR-codes.
- Otherwise, the server responds with an error message.

Both the ballot and card are secured and kept for further analysis (if required). The server also removes the card's public key from its internal list of accepted cards. Finally, the event is also added to the polling station's paper logbook.
Note that if the card fails to respond to the challenge, it will also be declared as responsible for the failure.

(4) *Verification of the print constraints.* In this step, the trusted device verifies if
- the serial number given by the card $SN_{paper}{}^{card}$ is equal to the serial number scanned from the ballot $SN_{paper}{}^{ballot}$
- the random given by the card $rand_{paper}{}^{card}$ is equal to the random scanned from the ballot $rand_{paper}{}^{ballot}$.

If this is not the case, the printing authority misbehaved as they signed two different ballots with the same short paper ballot digest. The terminal sends a message containing the short ballot digest $shortDigest_{elec}$ of the ballot in question, a request to change the state to *Print Constraint Violated* for the ballot in question, and a signature on both.
- If the signature is valid and the $shortDigest_{elec}$ corresponds to a ballot currently in state *Disputed*, the server creates a new block which is added to the electronic ballot box, to change the ballot's state. The server also computes a short block and a signature. The short block and the signature are then shown on the screen, where the short electronic ballot digest and the state are shown in plain, and the rest is contained in QR-codes.
- Otherwise, the server responds with an error message.

Both the ballot and card are secured and kept for further analysis (or as evidence if required), otherwise the card can be reused. Finally, the event is also added to the polling station's paper logbook.

(5) *Postpone verification.* If all checks up to this point were successful, the dispute cannot be resolved without breaking the voter's privacy. Hence, the remainder of the procedure is postponed and will be done after the election, by a trusted third party that does not know the link between the ballot and the voter. In any case, the server is told (by the user and official via the terminal) to add a corresponding block to the electronic ballot box: The terminal sends a message containing the short ballot digest $shortDigest_{elec}$ of the ballot in question, a request to change the state to *Resolution Postponed* for the ballot in question, and a signature on both.
- If the signature is valid and the $shortDigest_{elec}$ corresponds to a ballot currently in state *Disputed*, the server creates a new block which is added to the electronic ballot box, to change the ballot's state. The server also computes a short block and a signature. The short block and the signature are then shown on the screen, where the short electronic ballot digest and the state are shown in plain, and the rest is contained in QR-codes.
- Otherwise, the server responds with an error message.

At the end of all these checks, the voter can return to the registration desk, pick a new card and new ballots, and vote again.

Note that if during the dispute resolution procedure any party refuses to respond or deviates from the procedure, it will automatically be declared responsible. This does not only hold for, e.g., the smart card, but also for the voter: if, e.g., they destroy their ballot before it can be audited, they will be declared responsible in the corresponding step.

After the election is over, the following further analysis can be carried out by a trusted third party.

- *Further analysis of the paper ballot.* The envelope is opened and the ballot is analyzed in detail. In particular, the second QR-code including the $id_{cand}{}^{ballot}$, $verif_a{}^{ballot}$, $verif_b{}^{ballot}$, and the signature $sig_{all}{}^{ballot}$ can be read. One can then check whether the signature is valid.
  - If the signature is invalid, either the voter fabricated a false ballot and inserted it into the envelope instead of the real one used for voting, or the card accepted a fake ballot that was hidden among the real ballots without the voter noticing. A more precise analysis of the physical properties (quality of impression, type of paper, etc.) of the ballot should help to identify a false bulletin fabricated by the voter.
  - Otherwise, one can check whether
    (1) $verif_a{}^{ballot}$ is even and $verif_b{}^{ballot}$ is odd
    (2) both values are between 0 and $2 \times num_{cand} - 1$
    (3) $verif_a{}^{ballot} + verif_b{}^{ballot} = id_{cand}{}^{ballot} \mod (2 \times num_{cand})$
    where $id_{cand}$ is the number assigned to the candidate

  - If any of these checks failed, the printing authority signed an incorrect ballot and is declared responsible.
  - Otherwise one can check whether the scanned values correspond to the human-readable values. If this check fails, again one has to analyze the physical properties of the ballot to distinguish who is responsible for the ballots. Note that ballots should be randomly audited by the observers, and the voter confirms their vote on the screen using the value read from the QR-code rather than the human-readable value. Hence we can declare the voting terminal or the voter responsible for failing to correctly execute this check.
- *Further analysis of the smart card.* If all checks on the paper ballot succeed, the card is given the signature of the server on the block proving that the ballot was postponed for further analysis. The card then returns all other values it used during the voting process, in particular $\mathrm{id}_{\mathrm{cand}}{}^{card}$, $\mathrm{verif}_{\mathrm{a}}{}^{card}$, $\mathrm{verif}_{\mathrm{b}}{}^{card}$, and the signature $\mathrm{sig}_{\mathrm{all}}{}^{card}$. It will also reveal the randomness values $\mathrm{r}_{\mathrm{cand}}$, $\mathrm{r}_{\mathrm{a}}$, and $\mathrm{r}_{\mathrm{b}}$, used to encrypt all three ciphertexts. If the signature is invalid, the card is declared responsible.
- *Further analysis of the print constraints.* If the signature is valid, but the other values do not match the paper ballot, i.e., $\mathrm{id}_{\mathrm{cand}}{}^{card} \neq \mathrm{id}_{\mathrm{cand}}{}^{ballot}$ or $\mathrm{verif}_{\mathrm{a}}{}^{card} \neq \mathrm{verif}_{\mathrm{a}}{}^{ballot}$ or $\mathrm{verif}_{\mathrm{b}}{}^{card} \neq \mathrm{verif}_{\mathrm{b}}{}^{ballot}$, then the printing authority is responsible for having signed two different ballots with the same identifier.
- *Verification of the audit challenge.* The card is asked to reveal the signed challenge it received from the confirmation terminal.
  - If the challenge is incorrectly signed, the card is declared responsible for accepting an invalid challenge.
  - If the challenge does not correspond to the values $\mathrm{verif}_{\mathrm{a}}{}^{card}$ and $\mathrm{verif}_{\mathrm{b}}{}^{card}$ previously sent by the card, the locals and the voter are declared responsible for scanning and accepting an incorrectly scanned challenge.
  - If the ballot was not in state *In Audit* at the beginning of the dispute resolution procedure, the card is declared responsible for refusing a valid challenge.
  - Otherwise:
    - If the challenge does not correspond to the challenge visible through the window of the paper ballot, then the locals or the voter are declared responsible for having scanned and accepted an incorrect challenge.
    - If both values match, the locals and the voter are also declared responsible for starting the dispute resolution without a valid reason.

# B RESTRICTIONS TO MODEL MODULAR ARITHMETIC

When considering a reachability query, we consider the following set of restrictions to model the properties that the zero-knowledge proofs must ensure (regarding the integers):

- by specification of the protocol, when someone checks that $x \equiv a + b \mod c$ it must checks that $a$ is even and $b$ is odd. Hence, both cannot be equal:

$$\mathtt{isSum}(x, a, b) \quad \rightarrow \quad a \neq b$$

- given two integers $a$ and $b$, there exists a unique third integer $x$ such that $x \equiv a + b \mod c$, and its variants:

$$\begin{aligned} \mathtt{isSum}(x, a, b) \wedge \mathtt{isSum}(x, a, b') &\rightarrow b = b' \\ \mathtt{isSum}(x, a, b) \wedge \mathtt{isSum}(x, a', b) &\rightarrow a = a' \\ \mathtt{isSum}(x, a, b) \wedge \mathtt{isSum}(x', a, b) &\rightarrow x = x' \end{aligned}$$

- the result of a check is deterministic:

$$\begin{aligned} \mathtt{isSum}(x, a, b) \wedge \mathtt{isNotSum}(x, a, b') &\rightarrow b \neq b' \\ \mathtt{isSum}(x, a, b) \wedge \mathtt{isNotSum}(x, a', b) &\rightarrow a \neq a' \\ \mathtt{isSum}(x, a, b) \wedge \mathtt{isNotSum}(x', a, b) &\rightarrow x \neq x' \end{aligned}$$

# C MANUAL PROOFS FOR PRIVACY

In this section we present the theoretical result that allows us to claim that the protocol ensures vote secrecy, even if our ProVerif models do not faithfully model the arithmetic.

## C.1 Notations and definitions in ProVerif

We recall here the few elements needed to state and prove the result. For sake if simplicity, we omit technical details to focus on the main steps of the proof. We invite the interested readers to refer to [8] for omitted elements.

*Type system.* In ProVerif, messages are defined with types (e.g. `channel`, `bool`, `bitstring`,...). ProVerif checks that the processes given a user are well-typed to detect mistakes in the protocol specification. By default, it then ignores types when verifying the security properties. However, in order to define a more accurate attacker model, ProVerif procedure can be configured to take the types into account. In our security analysis we follow this approach defining the following types: `channel` to model communication channels, `bool` to model booleans, `const` to model the numbers $x, a, b \in \mathbb{N}$, and `bitstring` to model the other messages.

*Processes.* ProVerif manipulates processes as introduced in Section 3. Note that since the command $\cdot \mid \cdot$ is associative and commutative, we do not make any different between the processes $P \mid Q \mid R$ and $P \mid R \mid Q$ or $Q \mid P \mid R$. We say that a process $P$ *can output the message $a$ on channel $N$*, noted $P \downarrow_a$, if $P = \mathtt{out}(N, a); Q \mid P'$ for some processes $Q$ and $P'$.

For sake of simplicity, the set of function symbols is split in *public symbols* (resp. constants) available to the attacker for computations, and *private symbols* (resp. constants) unknown from the attacker. In the following, we say that a process $Q$ is an *adversary process* if it contains public symbols of function and public constants only.

*Bi-processes.* ProVerif is not designed to prove that two arbitrary processes $P$ and $Q$ are observationally equivalent. Instead, it focuses on processes that only differ by terms. To this aim, it defines *bi-processes* that are processes in which terms can be replaced by the specific construction $\mathtt{diff}[u_1, u_2]$ where $u_1$ and $u_2$ are terms (as usual). Given a bi-process $P$, we define the two processes $\mathtt{fst}(P)$ and $\mathtt{snd}(P)$ as follows: $\mathtt{fst}(P)$ is obtained by replacing all the occurrences of $\mathtt{diff}[u_1, u_2]$ by $u_1$ in $P$ and $\mathtt{snd}(P)$ is obtained by replacing all the occurrences of $\mathtt{diff}[u_1, u_2]$ by $u_2$.

The operational semantics, denoted by the relation $\rightarrow$, is extended as expected when the rules apply on both sides (i.e. $\mathtt{fst}(P)$ and $\mathtt{snd}(P)$). However, when a rule applies on the first component

but fails on the second one, we say that the bi-process *diverges*, noted $P\uparrow$. We note $\rightarrow^*$ the transitive closure of $\rightarrow$.

In the following we configure ProVerif to take types into account. We will thus consider well-typed traces (resp. bi-traces) only, i.e. that satisfy the type-system. When tr is a bi-trace, we note $\mathtt{fst}(\mathsf{tr})$ the first projection of the trace tr and $\mathtt{snd}(\mathsf{tr})$ the second projection of tr. Moreover, given a trace (resp. bi-trace) tr, we note $e(u_1,\ldots,u_n) \in \mathsf{tr}$ if there is a step in tr that execute the event $e(u_1,\ldots,u_n)$.

*Definition C.1.* Let $P_0$ be a closed bi-process. The bi-process $P_0$ satisfies *diff-equivalence* if for all adversary processes $Q$, there is no process $P$ such that $P_0 \mid Q \rightarrow^* P\uparrow$.

## C.2 Notations and definitions for privacy

As presented in Section 4.2, vote privacy is modeled as the indistinguishability of the two following scenarios: Alice votes 0, Bob 1, and Alice votes 1 and Bob 0. In the context of the Themis protocol, this scenario is modeled by the following bi-process:

$$\begin{aligned} P_{priv} = \;& C \\ & \mid \mathtt{Voter}(\mathit{Alice}, \mathtt{diff}[\mathtt{ballot}(x_1,a_1,b_1), \mathtt{ballot}(x_2,a_1,b_3)]) \\ & \mid \mathtt{Voter}(\mathit{Bob}\;\;, \mathtt{diff}[\mathtt{ballot}(x_2,a_2,b_2), \mathtt{ballot}(x_1,a_2,b_4)]), \end{aligned}$$

where $C$ models all the parties different from Alice and Bob as before, $x_1$ and $x_2$ encodes Alice's and Bob's choices, and $a_1, a_2, b_1, b_2, b_3, b_4$ are integers modeling the audit codes. By construction, the following relations hold:

$$x_1 \equiv a_1 + b_1 \equiv a_2 + b_4 \mod n,$$
$$x_2 \equiv a_1 + b_3 \equiv a_2 + b_2 \mod n.$$

Hence, the usual definition of vote privacy applies as follows: the Themis protocol ensures vote privacy if $\mathtt{fst}(P_{priv})$ and $\mathtt{snd}(P_{priv})$ are observationally equivalent. Unfortunately, since our model abstract arithmetic properties away, this definition does not faithfully captures the notion of vote privacy. Indeed, we need to restrict ourself to traces that correspond to realistic execution of the protocol, i.e. taking into account the arithmetic relations between numbers.

*Definition C.2.* A trace (resp. bi-trace) tr is *real-like* if for all $\mathtt{isSum}(x,a,b) \in \mathsf{tr}$ we have $x \equiv a + b \mod c$.

*Definition C.3. Observational equivalence* between processes $\approx_o$ is the largest symmetric relation $R$ between processes such that $P\ R\ P'$ implies:

(1) if $P \downarrow_a$, then $P' \rightarrow^* P''$ and $P'' \downarrow_a$ for all names $a$;
(2) if $P \rightarrow P_1$, then $P' \rightarrow^* P'_1$ and $P_1\ R\ P'_1$ for some $P'_1$;
(3) $(P \mid Q)\ R\ (P' \mid Q)$ for all adversary process $Q$.

*Definition C.4. Real-like vote privacy* holds if for all adversary process $Q$ and real-like trace $\mathsf{tr} = \mathtt{fst}(P_{priv}) \mid Q \rightarrow^* P$, there exists a trace $\mathsf{tr}' = \mathtt{snd}(P_{priv}) \mid Q \rightarrow^* P'$ such that $\mathsf{tr}'$ is real-like and $P \approx_o P'$ (and conversely).

## C.3 Main proof

We prove a more precise version of Theorem 4.1

Theorem C.5. *Let $P_{priv}$ be the configuration that encodes* vote privacy. *If $P_{priv}$ satisfies Equation (3) and* diff-equivalence *then* real-like privacy *holds.*

Proof. Let $Q$ be an adversary process and tr be a real-like trace such that $\mathsf{tr} = \mathtt{fst}(P_{priv}) \mid Q \rightarrow^* P$ for some process $P$.

Since $P_{priv}$ satisfies diff-equivalence, we know that there exists a bi-trace $\mathsf{tr}_b = P_{priv} \mid Q \rightarrow^* P_b$ such that $P_b$ does not diverge and $\mathtt{fst}(P_b) = P$. We note $\mathsf{tr}' = \mathtt{snd}(\mathsf{tr}_b)$. Therefore, we have: $\mathsf{tr}' = \mathtt{snd}(P_{priv}) \mid Q \rightarrow^* \mathtt{snd}(P_b)$.

Show that $P \approx_o \mathtt{snd}(P_b)$: In [8], Blanchet *et. al.* prove (Theorem 3.5) that observational equivalence holds as soon as diff-equivalence does. Therefore, $\mathtt{fst}(P_{priv}) \approx_o \mathtt{snd}(P_{priv})$.

We now define $R_0$ the largest symmetric relation $R$ such that $P_1\ R\ P_2$ if and only if there exists a bi-process $P_{1,2}$ such that $P_1 = \mathtt{fst}(P_{1,2})$, $P_2 = \mathtt{snd}(P_{1,2})$, and item 1 and 3 of Definition C.3 hold. Moreover, if $P_{1,2} \rightarrow^* P'_{1,2}$, then $\mathtt{fst}(P'_{1,2})\ R_0\ \mathtt{snd}(P'_{1,2})$.

Following the same proof as Theorem 3.5, we have that: $P_{priv}$ satisfies diff-equivalence implies that $\mathtt{fst}(P_{priv})\ R_0\ \mathtt{snd}(P_{priv})$. Hence, by definition of $R_0$, we have that $P\ R_0\ \mathtt{snd}(P_b)$.

Moreover, by definition of $R_0$ we also have that $R_0 \subseteq \approx_o$. Therefore, we have that $P \approx_o \mathtt{snd}(P_b)$.

Show that $\mathsf{tr}'$ is real-like: Let $\mathtt{isSum}(x_2,a_2,b_2) \in \mathsf{tr}'$. Applying *Equation (3)*, i.e.,:

$$\mathtt{isSum}(x,a,b) \in \mathtt{fst}(\mathsf{tr}_b) \Leftrightarrow \mathtt{isSum}(x,a,b) \in \mathtt{snd}(\mathsf{tr}_b),$$

we obtain that $\mathtt{isSum}(x_2,a_2,b_2) \in \mathsf{tr}$. Since tr is real-like, we obtain that $x_2 = a_2 + b_2 \mod c$. This concludes the proof.

□

In the ProVerif models we do not directly prove Equation (3). Instead, we prove Property (1) which immediately implies Equation (3).

*Definition C.6.* A bi-process $P$ satisfies *Property (1)* if for all bi-trace $\mathsf{tr} \in \mathtt{BiTrace}(P)$,

$$\mathtt{isSum}(\mathtt{diff}[x_1,x_2], \mathtt{diff}[a_1,a_2], \mathtt{diff}[b_1,b_2]) \in \mathsf{tr}$$

implies that:

(a) $(x_1,a_1,b_1) = (x_2,a_2,b_2)$; or
(b) $\mathtt{isSum}(\mathtt{diff}[x_1,x_1], \mathtt{diff}[a_1,a_1], \mathtt{diff}[b_1,b_1]) \in \mathsf{tr}$ and $\mathtt{isSum}(\mathtt{diff}[x_2,x_2], \mathtt{diff}[a_2,a_2], \mathtt{diff}[b_2,b_2]) \in \mathsf{tr}$.

Proposition C.7. *Let $P$ be a bi-process. If $P$ satisfies* Property (1) *then $P$ satisfies Equation (3) too.*

Proof. Since *Property (1)* is symmetric, w.l.o.g. we only prove one direction, i.e., $\mathtt{isSum}(x,a,b) \in \mathtt{fst}(\mathsf{tr}) \Rightarrow \mathtt{isSum}(x,a,b) \in \mathtt{snd}(\mathsf{tr})$. Let $x_1, a_1, b_1$ such that $\mathtt{isSum}(x_1,a_1,b_1) \in \mathtt{fst}(\mathsf{tr})$. By construction, we have that:

$$\mathtt{isSum}(\mathtt{diff}[x_1,x_2], \mathtt{diff}[a_1,a_2], \mathtt{diff}[b_1,b_2]) \in \mathsf{tr}.$$

*Case (a):* we have that $\mathtt{isSum}(x_1,a_1,b_1) = \mathtt{isSum}(x_2,a_2,b_2)$ Therefore, we obtain that $\mathtt{isSum}(x_1,a_1,b_1) = \mathtt{isSum}(x_2,a_2,b_2) \in \mathtt{snd}(\mathsf{tr})$.

*Case (b):* $\mathtt{isSum}(\mathtt{diff}[x_1,x_1], \mathtt{diff}[a_1,a_1], \mathtt{diff}[b_1,b_1]) \in \mathsf{tr}$. Therefore, $\mathtt{isSum}(x_1,a_1,b_1) \in \mathtt{snd}(\mathsf{tr})$.

This concludes the proof.

□