Vote&Check: Secure Postal Voting with Reduced Trust Assumptions

Véronique Cortier
Université de Lorraine, CNRS, Inria, LORIA, France

Alexandre Debant
Université de Lorraine, CNRS, Inria, LORIA, France

Pierrick Gaudry
Université de Lorraine, CNRS, Inria, LORIA, France

Léo Louistisserand
Université de Lorraine, CNRS, Inria, LORIA, France

Abstract

Postal voting is a frequently used alternative to on-site voting. Traditionally, its security relies on organizational measures, and voters have to trust many entities. In the recent years, several schemes have been proposed to add verifiability properties to postal voting, while preserving vote privacy.

Postal voting comes with specific constraints. We conduct a systematic analysis of this setting and we identify a list of generic attacks, highlighting that some attacks seem unavoidable. This study is applied to existing systems of the literature.

We then propose Vote&Check, a postal voting protocol which provides a high level of security, with a reduced number of authorities. Furthermore, it requires only basic cryptographic primitives, namely hash functions and signatures. The security properties are proven in a symbolic model, with the help of the ProVerif tool.

1 Introduction

Electronic voting and voting in general aim at two main security properties, namely vote secrecy and verifiability of integrity. Vote secrecy guarantees that no one learns information about how a certain voter voted while verifiability allows to check that the result corresponds to the actual votes of the eligible voters.

Internet voting has attracted a lot of attention in the past two decades. Several countries use Internet voting for legally binding elections, such as Estonia [32], Australia [23], Switzerland [36], or France [19]. Many academic systems have been proposed as well, such as the simple protocol Helios [2], used by the IACR (International Association for Cryptologic Research), or more advanced protocols that aim at achieving higher guarantees such as coercion resistance (e.g. Civitas [14, 25] or VoteAgain [31]). Another form of remote voting is postal voting. Its advantage is its simplicity: voters do not need any computer to cast a vote, they simply send a paper ballot by mail. This is a common practice in many countries: it is used by 90% of the voters in Switzerland [34] and 46% of the votes were cast via mail ballot in the 2020 US presidential election [42].

Surprisingly, while postal voting is used for high stake elections, it has deserved much less attention than Internet voting and its security level is typically low. For example, in the United States, the voters simply fill-in their ballot and manually sign the return envelop that contains their ballot. Not only this is a weak form of authentication but the authorities need to be trusted for vote privacy: they may open the envelope and read the vote right after having authenticated the voter. Verifiability is not provided either. Once the ballots are received, they need to be securely stored until the tally. Any person having access to them may remove or replace them. Moreover, when ballots travel through postal services, postmen may selectively drop ballots that come from some area known to vote for a certain candidate. A study in Switzerland [26] shows that postal voting is actually complex and involves many parties.

There has been some recent effort to improve the security of postal voting. STROBE [3] makes a significant first step by introducing verifiability. An entity, called the printer, prepares the voting material, prints it, and sends it to voters. Roughly speaking, the voting sheet contains the name of the candidate in clear but also its encrypted version. The voter selects their favourite candidate and can then check that the corresponding encryption appears on some public board after the tally. In order to verify that the ballots have been correctly generated, each voter actually receives two ballots and randomly selects one for audit: the printer must prove that it was encrypted correctly by providing the corresponding randomness. RemoteVote and SAFE Vote [18] further improve this approach in terms of usability, so that the voter no longer receives two ballots. An important drawback of these systems however is that they improve verifiability at the cost of sacrificing privacy. Indeed, an honest but curious printer knows perfectly well to whom it has sent the ballots and it also knows the correspondence between candidates and encrypted ballots. Hence, after the tally, it can simply read the accepted ballots from the public board and deduce who voted what, for the entire population. We explain this attack in more detail in Section 2. Another recent work is the system by Devillez et al. [22]. An additional authority is introduced, the verification server, that the voters should contact to verify that their ballot has been correctly counted. Indeed, there is no public bulletin board where the voters can look at to find for their ballot. While this system has better security properties than STROBE, RemoteVote and SAFE Vote, specially regarding privacy, it only provides proxy-verifiability instead of the usual stronger notion of universal verifiability.

Design choices. While being called postal voting, all these systems make use of Internet. In particular, voters need to access a public bulletin board or a verification server and they may need to perform cryptographic checks. However, such systems remain

Proceedings on Privacy Enhancing Technologies 2025(3), 333–348

https://doi.org/10.56553/popets-2025-0101

paper-based in the sense that voters still cast their vote without any devices. They can then *optionally* use devices for verifiability. We believe that such a design is a promising approach that may ease the replacement of old style (insecure) postal voting: voters may still vote as usual and they cast a vote for the candidate they *see* printed on the ballot. Of course, voters who do not verify get less guarantees but the fact that a proportion of voters will verify may be sufficient to incentivize the authorities to behave as expected, because dropping or modifying ballots becomes more risky. Moreover, proposing two simultaneous systems (postal and Internet) would be more expensive.

Our contributions. We first provide a list of attack scenarios, that are applicable to most postal voting systems. Whether the attack is successful then depends on the analysed system and of course, other attacks may exist. One of our generic attacks is the full privacy breach scenario that we mentioned above: the printer (or another authority) records all the data it has provided to voters. Then given the public information available after tally (e.g., the bulletin board), it looks for identifying data such as encrypted votes, which may allow it to deduce the vote of each voter. We apply our attack scenarios to the STROBE protocol and we unveil several flaws, beyond the weaknesses that were acknowledged by the author in the paper. In addition to the privacy attack, we show how authorities can manipulate the votes. Interestingly, some of our attacks seem inherent to postal voting and hence will probably apply to any system. For example, complaints may be used to break privacy. Indeed, if the authority in charge of collecting the ballots drops one ballot that votes for candidate A, then a voter will complain, and the authority will learn that this voter voted for A. This applies to other Internet voting protocols such as Selene [38] or sElect [29]. We further study whether our attack scenarios successfully apply to RemoteVote, SAFE vote, and Devillez et al.'s protocols.

Our second and main contribution is the design of Vote&Check, a simple postal voting system that aims at providing vote privacy against a dishonest printer and any other dishonest authority (but not against a collusion of them). The idea is simple: the voter receives a voting sheet that contains a credential c from the printer, the list of candidates in clear, and some authentication data. The voter selects their candidate and sends back their ballot, with the credential. Then the vote will appear in clear on the bulletin board, next to a public tracker $w = h(c, t)$ that is obtained by hashing c with a tracker t that the voter can obtain by connecting to an external authority, the Tracker Server. This protects the voter from privacy attacks from both the printer and the Tracker Server, unless they collude. Vote&Check also guarantees provides individual and universal verifiability, without having to rely on a proxy. A voter can check on a public bulletin board that their vote intent has been counted. Furthermore, anyone can check that the result corresponds to the votes on the bulletin board. Indeed, since the votes are provided in clear, it is sufficient to count. Finally, the fact that votes only come from legitimate voters (eligibility verifiability) is guaranteed as soon as one authority is honest and up to the fact that a dishonest printer may always cast a ballot for an absentee voter. This is due to the fact that the material received by post solely suffices to cast a vote. This is also one of our generic attack scenarios that we describe in the first part of our work.

Interestingly, our protocol achieves a higher degree of security than previous systems relying only on basic standard cryptographic primitives: it only requires hash functions and standard digital signatures, available in almost all cryptographic libraries (OpenSSL, libsodium, WolfCrypt, etc) and/or secure devices (e.g., HSM). ElGamal encryption, which is widely used in electronic voting, is less ubiquitous; in general, it would require simple, but specific development based on cryptographic libraries APIs. Furthermore, ElGamal encryption often comes with a distributed, and often thresholdized, key generation and decryption in order to distribute trust among several authorities. This comes at a cost: it imposes organisational constraints, but also computation and communication overheads for the participants.

Another simplicity aspect of Vote&Check comes from the number of independent authorities that are required. It is customary to increase the number of authorities, in order to share the trust and avoid giving too much power to a single entity, and in that sense, there is a tradeoff between security and simplicity. Since there is no encryption in Vote&Check, it does not need independent decryption authorities as in the previous systems, which simplifies the organization of elections. More generally, we claim that Vote&Check provides a good balance between a number of authorities that is manageable, and high security guarantees. Vote&Check requires a public bulletin board, but this is a rather simple version of it, since it is not used during the voting phase. A webpage containing the data signed by all authorities at the end of the setup and at the end of the tally is enough for implementing it.

Furthermore, a nice feature of our protocol compared to STROBE and other verifiable postal voting systems that we are aware of, is that it supports complex counting systems based on ranking the candidates or giving them a grade (STV, Condorcet, etc).

Security proof. We formally prove the security of Vote&Check using ProVerif [7], a popular tool for the analysis of security protocols. While the cryptographic primitives are very simple and hence easy to model, we had to account for particular physical channels such as postal voting where the attacker can send a mail to a targeted voter but it cannot open their mailbox (at least in some threat models). For privacy, we had to handle the fact that complaints break privacy, as indicated by one of our generic attacks. We instead show that our protocol preserves privacy provided that complaints can be made *anonymously*, for example through a trusted third party (a judge). For verifiability, we use the recently proposed framework [15] developed for ProVerif. We unveiled a limitation of the framework, that implicitly assumes that each ballot can be identified and linked to a voter. However, in Vote&Check, there is no identifying data. Instead, we show how to use the credential and the tracker, with a flexible association that can depend on the trust assumptions. We also had to circumvent the fact that ProVerif cannot easily reason on "else" branches and we introduced a new axiom, for which we provide a proof of correctness. We believe that these proof techniques can be used for other voting protocols.

Related work. The closest works are the protocols STROBE [3], RemoteVote and SAFE Vote [18], and Devillez et al.'s [22] that we already discussed. We conduct a more thorough analysis of them in Section 2. In brief, the main difference with STROBE, RemoteVote

and SAFE Vote is that we wish to prevent an authority from learning the votes of the entire population, without being detected. The security guarantees offered by Devillez et al.'s are closer to our system, although we also protect against some less severe attacks such as clash attacks [28]. Moreover, this protocol involves a larger number of independent authorities, namely four independent authorities plus n talliers, where we only require three independent authorities (the Printer, the Tracker Server and the Election Office) as well as a public bulletin board.

Other approaches [4, 33] are also called postal voting as voters cast their ballot by post but they need a computer to generate the voting material and they also need to print the material themselves. Hence, even if they help a lot to avoid breaking voter's privacy, they are closer to Internet voting.

Finally Prêt-à-Voter [39] and Pretty Good Democracy (PGD) [40] are voting systems where voters can vote using only paper ballots. However, these protocols are designed for on-site voting, which leads to a different threat model. It is not obvious to turn them into (remote) postal voting. Vote&Check belongs to the family of tracker-based voting systems. In particular, it borrows to sElect [29] and Selene [38] the idea that the voter will see their vote in clear on the public board, aside a tracker that they can recognize. These two systems strongly assume that the voter uses an electronic device to cast their vote and are not directly applicable to postal voting.

2 Attack scenarios

Verifiable postal voting systems have a common structure. Voters receive some material by post, that has been issued by one or several authorities (among them, the printer). Voters then select their candidate and send back their selection, possibly retaining some part of the voting material. At the end of the election, voters typically have access to some additional data, for example on a public bulletin board, and they can perform some checks to verify that their vote has been properly counted.

2.1 Generic attack scenarios

We list generic attacks against privacy and verifiability which may apply to any system which does not implement dedicated countermeasures. For each property, we consider several corruption scenarios and mention possible countermeasures if they exist.

2.1.1 Threat model. The list of parties involved in a postal voting system varies across the systems but typically contains at least: the voters; a printer, that prints and sends the voting material; a ballot collector; and a bulletin board. The ballot collector is in charge of collecting the ballots sent by voters and announcing the result, possibly publishing some data on a bulletin board. While voters, printer, and ballot collector may all be dishonest, the bulletin board is typically assumed to be a secure, public, and append-only board that anyone can see in an authenticated manner.

In what follows, we list potential attacks against privacy and verifiability, when at most one of these parties is corrupted. Some postal voting systems involve other parties, that may also be corrupted but these system specific scenarios are not considered here.

2.1.2 Privacy.

Targeted privacy attack (corrupted voters). A dishonest voter Charlie gives his material to Alice (e.g., drops his material in Alice's mailbox) and keeps a copy. Alice uses this material instead of hers. Then Charlie uses the verification mechanism to learn Alice's vote. A more powerful variant is when Charlie can actually generate valid ballots by himself, which allows him to attack privacy of several voters, while keeping his right to vote.

Possible countermeasures: by signing the material (or part of it), one can prevent Charlie from generating valid ballots by himself. Moreover, if this signature binds the identity of the voter it could prevent Charlie from switching his material with Alice's. However, it introduces a new trusted party (the signer). Of course the trust can be put elsewhere, e.g., assuming the presence of secure channels between the voter and some authorities/parties. This is mechanism used in [22] and, to some extent, in Vote&Check as well.

Full privacy breach (corrupted printer). Some authority, typically the printer, knows the link between the voter and some identifying data (e.g., an authentication token). This identifying data may appear on the bulletin board, next to the vote in clear. In that case, the authority (honest but curious) breaks vote privacy of all voters. A variant of the attack is when the authority generates all possible encrypted votes for a voter and the selected encrypted ballot appears on the bulletin board. The ballot identifies the voter and hence vote privacy is broken again.

This is a powerful attack since the privacy of all voters is broken w.r.t. this authority, without any detection.

Possible countermeasures: a trivial countermeasure is to keep the bulletin board secret (of course it raises other challenges for verifiability). Another approach is to distribute the generation of the identifying data as done in Vote&Check.

Privacy breach by complaints (corrupted ballot collector). When votes appear in clear on the paper ballot, the authority collecting the ballots may alter or remove a vote (without knowing who voted for it) and see who complains.

Such an attack also applies to pure Internet voting schemes as well, such as sElect [29], Selene [38], or Hyperion [37]. This attack is detectable, by construction, but a few complaints may not draw attention. This attack was not detected in their respective privacy proofs [13, 29] because they assume that the collecting authority behaves honestly w.r.t. the honest voters.

Possible countermeasures: at the time of writing, we don't know how to prevent such attacks in practical system.

2.1.3 Verifiability.

Ballot stuffing and vote flooding attacks (external attackers). In some systems, anyone may create a fake ballot and vote, which leads to ballot stuffing. This is typically avoided by authenticating some part of the voting material. A weaker form of this attack is when an attacker may find sufficient information (e.g., on the bulletin board) such that they can cast a vote without knowing for whom they voted. This allows to artificially increase the turnout and change the proportion of votes for each party, allowing e.g., a party to reach a certain quorum. We call this attack *vote flooding*.

Possible countermeasure: binding each ballot to a unique identifier prevents these attacks. However, this binding must remain secret until it is used to prevent flooding attacks. This binding can be ensured by different authorities (like in [22] or Vote&Check) to avoid single point of failures.

Weak eligibility attack (corrupted printer). An authority (typically the printer) can keep copies of the voting material and vote for absentees.

Possible countermeasure: Such an attack seems unavoidable if the printed material is solely sufficient to cast a vote.

Clash attack (corrupted printer). An authority (typically the printer) may send the same ballot to two voters that vote the same way (this can be guessed easily for some voters). Then only one of the two votes is counted. Such an attack leaves traces since the ballot collector will typically receive duplicated ballots. However, it is hard to decide what to do when two identical ballots are received since some (dishonest) voters may also try to vote twice by reusing their material. Hence, it is hard to identify who misbehaved.

Possible countermeasure: binding the ballots (or voters) to a unique identifier prevents this attack. However, it requires that injectiveness of the mapping is guaranteed by an authority other than the printer (who is assumed to be compromised).

Alter votes of non-verifying voters (corrupted ballot collector). Not all voters perform the verification steps. For non-verifying voters, an authority may try to flip their vote. Note that the collecting authority may always drop the vote (since the voters do not verify) but flipping the vote yields a more powerful attack.

Possible countermeasure: while there are techniques to prevent vote flipping in Internet voting protocols, we are not aware of a technique that is usable for postal voting since they require cryptographic operations performed by the voter/voting device.

2.2 Example of the STROBE protocol

The STROBE [3] voting scheme is a postal voting scheme designed by Josh Benaloh. It aims to provide verifiability while staying as close as possible to traditional vote by mail. The different entities involved are the printer, the postal service, the cast officer, the voters, and a set of trustees. There is a public board where the printer, the cast officer and the trustees can write any message.

2.2.1 Protocol. STROBE uses probabilistic homomorphic encryption (for example ElGamal) under the public key of the set of trustees. For each voter, the printer encrypts the votes, i.e. the identity matrix of size the number of candidates (blank vote is encoded as a candidate named "None"). Each line of the encrypted matrix is hashed, only the last byte of the hash is kept as a short code (shown on Figure 1a). This is repeated until all short codes are different. Lines of the matrix are permuted such that the short codes increase. The identifier of this ballot is the hash of the permuted matrix. The permuted matrix, the short codes and the ballot identifier are published on the board, as well as zero-knowledge proofs (produced by the printer) that this matrix is the encryption of a permutation (cf Figure 1c). Ballots (shown on Figure 1b) are paired and the pairings are made public. Each voter receives by postal mail two paired ballots. They can verify that the ballots are well-formed by looking if the information on the board corresponds to what they have received. Then they simply choose one of the ballots, tick the box of their choice and send it by postal mail to the cast officer. The cast officer publishes the ballot identifier and the chosen short code, that a conscientious voter can verify. Finally, the trustees multiply all the selected encryptions and decrypt the product, giving the result of the election. The printer also reveals the random numbers used to encrypt the unused ballots; auditors can then check that they were well-formed.

2.2.2 Instantiation of the attacks. Most of the generic attacks apply to the STROBE protocol, in different threat models.

Full privacy breach. The printer is supposed to not retain the link between voters and ballots. However, there is no way to verify that a dishonest printer indeed deleted it and if they have not, then they can learn the correspondence between plaintext votes and voters, by looking at the selection code and recalling to which candidate it corresponds.

Targeted privacy breach. If Charlie is a dishonest voter and wants to learn Alice's vote, he can copy his ballot and drop the original one in Alice's mailbox. In STROBE, ballots are all similar and unrelated to the voters, so that Alice has no way to distinguish her ballot from Charlie's. If she votes with the latter, Charlie will learn her vote by reading the selection code on the public board.

Privacy breach by complaints. The cast officer collects all the ballots. They know all plaintext votes, but they do not know the link between those votes and the voters. A compromised officer can willingly drop ballots for a specific candidate. Then, they can deduce that all the voters who complain have voted for this candidate.

Vote flooding. Using the public information from Figure 1c, an attacker can produce fake ballots, indistinguishable from real ones. They just have to rank the selection codes in a random order and send the forged ballot to the polling station to have it counted. This will artificially raise the turnout, as well as the score of the smaller candidates.

Alter votes of non-verifying voters. The cast officer receives all plaintext votes and have to record the corresponding selection codes on the public board. A compromised officer can drop the ballots they dislike or record other selection codes instead. This will be undetected if the corresponding voters do not verify.

Among all of these, Full privacy breach and Alter non-verifying voters are acknowledged in the STROBE paper: it is made very clear by the author that the printer is assumed to be honest, and there is no claim of verifiability if the voter does not verify. The other three attacks seem to have been overlooked by the author.

2.3 Other protocols

RemoteVote and SAFE Vote [18] are two postal voting schemes inspired by STROBE. The goal was to gain everlasting privacy as well as some usability. The approach proposed in [22] by Devillez et al. still relies on return codes, but goes further away from STROBE. It has the highest security features among the 3.

2.3.1 RemoteVote. The main difference between STROBE and RemoteVote is that each voter receives a single ballot instead of two, to improve usability. The ballot contains the two columns of selection codes that would have been on the two STROBE ballots. Similarly, one of these columns is spoiled after the election for verification while the second is used for tallying. But unlike in STROBE, it is not the voter who arbitrarily chooses which column is audited, but it is the result of a computation with the data of the ballot and a nonce randomly chosen after the election by a third party. This makes a *clash attack* possible, that we now describe.

Candidate	Encryption	Code
Alice	$(\mathrm{enc}(1), \mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(0))$	Q4
Bob	$(\mathrm{enc}(0), \mathrm{enc}(1), \mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(0))$	D6
Charlie	$(\mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(1), \mathrm{enc}(0), \mathrm{enc}(0))$	L7
None	$(\mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(1))$	E1

(a) Structure of the encryption

Alice	□	Q4
Bob	□	D6
Charlie	□	L7
None	□	E1

1960-857c-c5db-3939-2711-95e3

(b) Paper ballot

Encryption	Code
$(\mathrm{enc}(0), \mathrm{enc}(1), \mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(0))$	D6
$(\mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(1))$	E1
$(\mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(1), \mathrm{enc}(0), \mathrm{enc}(0))$	L7
$(\mathrm{enc}(1), \mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(0), \mathrm{enc}(0))$	Q4
Ballot code : 1960-857c-c5db-3939-2711-95e3	

(c) Information published on the board

Figure 1: Voting material in the STROBE protocol.

		STROBE [3]	RemoteVote / SAFE Vote [18]	Devillez et al. [22]	Vote&Check (this work)
	# of authorities (excl. post and auditor)	2 + **1**	2 + **1**	4 + **1**	3
	public board	yes	yes	no	yes
privacy attacks	targeted privacy breach	✘$^{\text{new}}$	✘$^{\text{new}}$	✔	✔
	full privacy breach	✘	✘	✔	✔
	privacy breach by complaints$^{\text{new}}$	✘	✘	✘	✘
verifiability attacks	ballot stuffing	✔	✘$^{\text{new}}$	✔	✔
	vote flooding	✘$^{\text{new}}$	✘$^{\text{new}}$	✔	✔
	weak eligibility$^{\text{new}}$	✘	✘	✘	✘
	clash attack	✔	✘$^{\text{new}}$	✔*	✔
	alter non-verifying voters	✘	✘	✘	✘
	universal verifiability	✔	✔	proxy	✔

Table 1: Application of the generic attacks to postal voting schemes. For the number of authorities, we indicate with a bold font "**+ 1**", when one authority is actually a set of thresholdized authorities, typically the decryption trustees. The symbol ✔ means we did not find this kind of attack on the protocol, while the symbol ✘ indicates that the protocol is subject to this attack. The symbol $^{\text{new}}$ means that this attack is our finding; otherwise, it was part of the threat model of the authors of the said protocol. When the symbol qualifies a property, it means that this is a refined security property that we introduce in this paper. Finally, for [22], we marked the symbol with *, because a verification step required for this property is left implicit in the description of the protocol.

If the printer suspects several voters to cast the same vote, they may provide them with the same ballot. The polling station cannot distinguish this from one voter sending multiple copies of their ballot. Since which column is spoilt is the result of an unpredictable but deterministic computation, it is the same for all the copies of the ballot, so each voter will be able to conduct the verifiability even if only one vote has been recorded for all of them. This attack would have been detected in STROBE, where the voters with the same two ballots would have cast either one or the other. The polling station would have accepted only one of them and the voters that used the other one could have detected it.

Another major difference with STROBE is that, in RemoteVote, there is no permutation. Hence, from the data that is published, anyone can create a fake ballot, and vote for the candidate of their choice. Therefore, the vote flooding attack that was present in STROBE becomes a more powerful ballot stuffing attack.

2.3.2 SAFE Vote. In this variant, each ballot contains a single column of selection codes. The randomness that allows to perform the audit for this ballot is printed directly on the ballot, but is concealed behind a scratch-off surface.

This does not really change the situation. The clash attack by the printer works exactly the same as in RemoteVote. Also, since the audit is not performed by the authorities, anyone can use the public data to create fake ballot, putting random values behind the scratch-off. We therefore have the same ballot stuffing attack as in RemoteVote.

2.3.3 Devillez et al.'s protocol. In this protocol, the printer generates all the voting material to be sent to voters, together with the associated cryptographic data to be sent to other parties. The paper ballots received by the voters contain one selection sheet, that is very similar to a classical paper ballot, with the only addition of a random token tk. The voter ticks the boxes near the candidates they select, and they send back this sheet by post. They also receive from the printer a list of return codes, 2 for each candidate, corresponding to whether or not they selected this candidate. They also get the value $\mathrm{hash}(\mathrm{tk})$.

After the result is announced, the voters can contact (electronically) a verification server, send them hash(tk), and they will receive back the return codes corresponding to their choices.

In order to make this process secure, the verification server does not receive directly the return codes from the printer. Instead, the printer will send them encrypted to talliers, who share the decryption key. These talliers operate after the tally; they collectively decrypt only the relevant return codes and send them to the verification server.

We skip many important details, but the outcome is that the scheme is claimed to be secure as soon as there is no collusion between two authorities, nor between one authority and the post.

However, we noticed a path to a clash attack when the printer colludes with the post. If the printer suspects that two voters will vote in the same way, it can send them exactly the same voting material, and send to the authorities the same cryptographic data for these voters. In this attack, the post will drop one of the ballots sent by those two voters. Based on the cryptographic material received from the election office, the verification server will answer correctly to both voters, unless they check that there are no duplicates in the data they received from the printer. While this check is not present in the description of the protocol, the authors had this verification in mind, because the security proof explicitly relies on it.

Compared to our Vote&Check protocol, other more fundamental drawbacks of the protocol by Devillez et al. are the following. First, it is not really universally verifiable; one has to trust authorities for this property. Furthermore, it uses more advanced cryptographic tools, such as a distributed threshold key generation, for the talliers. Finally, it requires more authorities than in Vote&Check, which might be a problem for practical deployment.

3 Vote&Check

3.1 High level description

3.1.1 Participants. The Vote&Check protocol involves the following participants:

- **Printer**. This entity is responsible for sending by postal mail the voting material to the voters.
- **Tracker Server**. This server provides a tracker to each voter, to let voters check that their ballot is counted.
- **Election Office**. This entity receives the postal ballots from the voters, and in the end publishes the votes in clear, together with verification data.
- **Public board**. A publicly readable place, that contains the result and verifiability data.
- **Auditors**. One or several entities who perform consistency checks of what is written on the board.
- **Voters**. The voters are assumed to have a valid postal address, and, for verifiability, an electronic way to receive securely data from the Tracker Server, and to read the public board.

In the description of Vote&Check and in its security analysis, we will often separate the (honest) voters in two groups: those who perform all the optional verifications steps, thus requiring an electronic device which is able to scan QR-codes and which is connected to the Internet, and those who stick to the traditional steps that require only pen and paper. Voters in the first group are called *conscientious*, and the others are called *offline*.

3.1.2 Protocol phases. The protocol is divided into several phases: setup, vote, tally and verification, as shown on Figure 2. During the setup, authorities send material to voters: they get voting material (right part of Figure 2), by post, from the Printer and (optionally) receive electronically verification material from the Tracker Server. The Tracker Server also sends tracking information to the Election Office. During the voting phase, voters fill in their paper ballot with a pen, and send them, by post, to the Election Office. During the tallying phase, the Election Office performs some validity checks and publishes every valid vote next to the corresponding tracking information. During the verification phase, voters can verify that their vote has been recorded using their verification material.

3.2 Detailed description

The details of each phase of the protocol, with the flow of messages between the participants, are summarized in Figure 3.

3.2.1 Channels and authentication. The protocol relies on secure channels between the participants, which guarantee integrity, authenticity, and confidentiality of all messages. The exceptions are that reading the board does not require authentication and that the postal channels are anonymous. Writing on the board requires authentication and we assume that each piece of data on the board is available together with the identity of the writer of this data, which could be implemented by each writer signing the data that they put on the board. We let these implicit, because we consider them as part of the public board functionality.

Furthermore, the Printer needs to sign some material that is sent to the voter, and this is made explicit in the protocol. Any classical signature scheme, like Schnorr or ECDSA, can be used, as long as it fits in a QR-code. The public key of Printer is assumed to be known by all parties. We denote by sig the Printer's signature function, and we omit the key. We also omit the fact that the signature must be bound to a precise election, so that it can not be replayed, for instance if there are two rounds of elections with the same participants. Therefore, the notation $\mathrm{sig}(x)$ that we will use from now must be understood as $\mathrm{sig}_{\mathrm{sk(Printer)}}(\mathrm{context}, x)$. The protocol will also use a cryptographic hash function, denoted by hash.

3.2.2 Setup phase. Let $\mathcal{V} = \{V_i, i \in \mathcal{I}\}$ be the set of all n voters, where $\mathcal{I} = [1, n]$. We assume $\mathcal{V}$ is known by all the authorities.

First, for every voter V_i, the Tracker Server generates a pseudonym a_i (a nonce) and a tracker t_i (another nonce). The Tracker Server sends their pseudonym and tracker (a_i, t_i) to each voter V_i over an electronic channel. The Tracker Server also sends the permuted list $\{(a_{\pi(i)}, t_{\pi(i)}), i \in \mathcal{I}\}$ to the Election Office, where π is the permutation over $\mathcal{I}$ that sorts the a_i alphabetically. Finally, the Tracker Server sends the list $((V_i, a_i), i \in \mathcal{I})$ to the Printer.

The Printer checks that the pseudonyms are pairwise distinct and that the identities match the already published list $\mathcal{V}$. If it is the case, they generate for each voter V_i a credential c_i and send to V_i over the postal channel a ballot containing a_i, c_i, $\mathrm{sig}(a_i)$ and $\mathrm{sig}(a_i, c_i)$ in a QR-code, plus a sheet that contains $\mathrm{sig}(a_i, c_i, V_i)$. This last signature is sent to the voter in a separate sheet so that the Election Office will not receive it, to protect voter's privacy. This signature is there to guarantee the voter that they are the intended

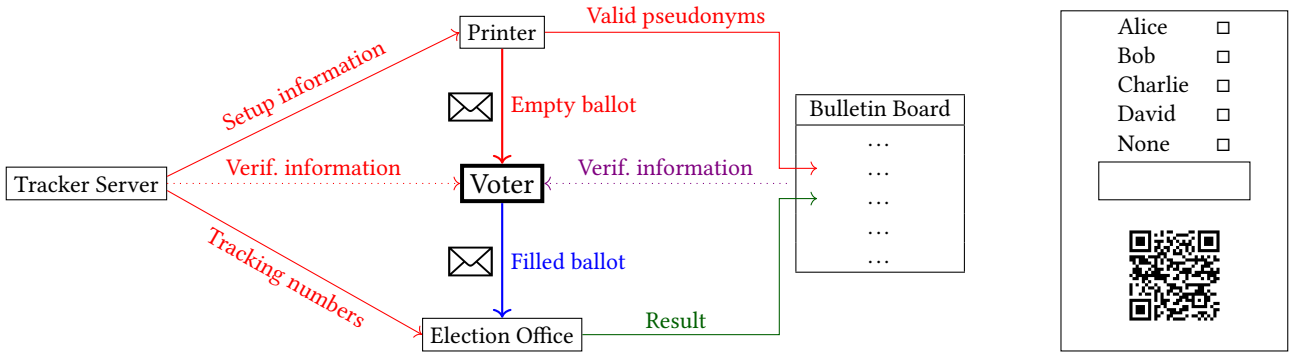

Figure 2: Vote&Check protocol overview (left part). A different colour is assigned to each phase: red for the setup phase, blue for the voting phase, green for the tally phase, and violet for the verification phase. Dotted lines are optional: voters can still vote if they are offline. **Example of a ballot** (right part). The QR-code contains the voters' pseudonym a_i, the ballot's credential c_i, and signatures by the Printer. The empty box is for the voter to write their nonce n_i.

recipient of this voting material and protect against targeted privacy breach. It also serves as a way for the voter to keep the value c_i for the verification phase. Finally, the Printer publishes the list $\mathcal{A} = (a_{\pi(i)}, i \in \mathcal{I})$ on the board, where π is the same sorting permutation as before. The other authorities, namely the Printer and the Election Office, verify that this list matches their own view. The Auditors check that the list contains the correct number of items.

3.2.3 Voting phase. Each voter receives by post the two sheets of paper sent by the Printer: their paper ballot and the additional paper with just a signature. Each conscientious voter, with their verification device, also receives their verification material from the Tracker Server and verifies that the three signatures on the ballot and on the additional paper are correct and consistent, that their name appears (signed) on the additional sheet, and that the pseudonym on the ballot matches the one on the verification material and that this pseudonym is present in the list $\mathcal{A}$ from the board.

The voting procedure itself, done by both conscientious and offline voters involves just pen and paper. They pick a short nonce n_i and write it on their ballot. Then, they tick the box corresponding to their vote v_i and send the ballot to the Election Office via the post. Combining the handwritten and the electronic data contained in the QR-code, the ballot contains $(a_i, c_i, v_i, n_i, \mathrm{sig}(a_i), \mathrm{sig}(a_i, c_i))$.

The nonce n_i is chosen and written by the voter and must therefore be short. However, it must be large enough to ensure that a clash attack will go undetected with only a small (but non-negligible) probability. We typically suggest a 3-digit number.

3.2.4 Tally phase. The Election Office controls the signatures on the received ballots and discards those with incorrect ones. They also discard ballots with a pseudonym a_i that does not belong to $\mathcal{A}$, as seen on the board. Lastly, they discard every ballot using a pseudonym that has already been used. The Election Office is responsible for ensuring these three properties. For every remaining ballot $(a_i, c_i, v_i, n_i, \mathrm{sig}(a_i), \mathrm{sig}(a_i, c_i))$, the Election Office finds the corresponding couple (a_i, t_i) in the list received from the Tracker Server during setup, and inserts $(\mathrm{hash}(c_i, t_i), v_i, n_i)$ in the list $\mathcal{B}$ of valid votes and $(a_i, \mathrm{sig}(a_i))$ in the list $\mathcal{P}$ of used pseudonyms. To publish the result, the Election Office publishes the shuffled list of valid votes $\mathcal{B} = \{(\mathrm{hash}(c_{\rho(i)}, t_{\rho(i)}), v_{\rho(i)}, n_{\rho(i)})\}_i$ where ρ sorts the $\mathrm{hash}(c_i, t_i)$ in alphabetical order. The index i belongs to $\mathcal{I}_{\mathrm{eff}} = [1, n_{\mathrm{eff}}]$ where n_{eff} is the size of $\mathcal{B}$. The Election Office also publishes the list of used pseudonyms $\mathcal{P} = \{(a_{\pi'(i)}, \mathrm{sig}(a_{\pi'(i)}))\}_i$, where, as for $\mathcal{A}$, π' sorts the a_i in alphabetical order.

The Printer and the Tracker Server can each reconstruct the voter list by checking which pseudonyms a_i have been used.

3.2.5 Verification phase. Auditors verify that the lists of used pseudonyms and of valid votes have the same length, that each pseudonym is signed and belong to $\mathcal{A}$, and that $\mathcal{A}$ has a size equal to the number of registered voters. Each conscientious voter computes $\mathrm{hash}(c, t)$ from c and t received resp. from the Printer and Election Office and checks that it appears on $\mathcal{B}$ next to their vote v and nonce n.

3.3 Security claims

As usual in analyzing the security of voting systems, we assume that at least one of the Auditors is honest and will raise an alert if something goes wrong. Also, we allow several voters to be dishonest and to help the attacker in attacking privacy and individual verifiability of honest voters.

3.3.1 Privacy. Vote secrecy is guaranteed as long as at least two authorities among the Tracker Server, the Printer and the Election Office are honest. The key to this is that the information published on the board can not be tracked back to voters without data coming from both the Tracker Server and the Printer.

Not surprisingly, as soon as two authorities collude, they can break privacy, as we now explain briefly. If the Tracker Server and the Printer collude, they hold all the information that voters use to perform the verification: the Tracker Server knows (V_i, t_i), the Printer knows (V_i, c_i) and on the board stands $(\mathrm{hash}(c_i, t_i), v_i)$. Similarly, the Election Office knows (t_i, v_i) and (c_i, v_i) so they may collude either with the Tracker Server or the Printer to break the confidentiality.

Moreover, Vote&Check is subject to privacy breach by complaints as introduced in Section 2.1: if the Election Office removes a ballot for candidate A, it can then observe who complains: the complaining voter has voted for A. This can be scaled to several voters

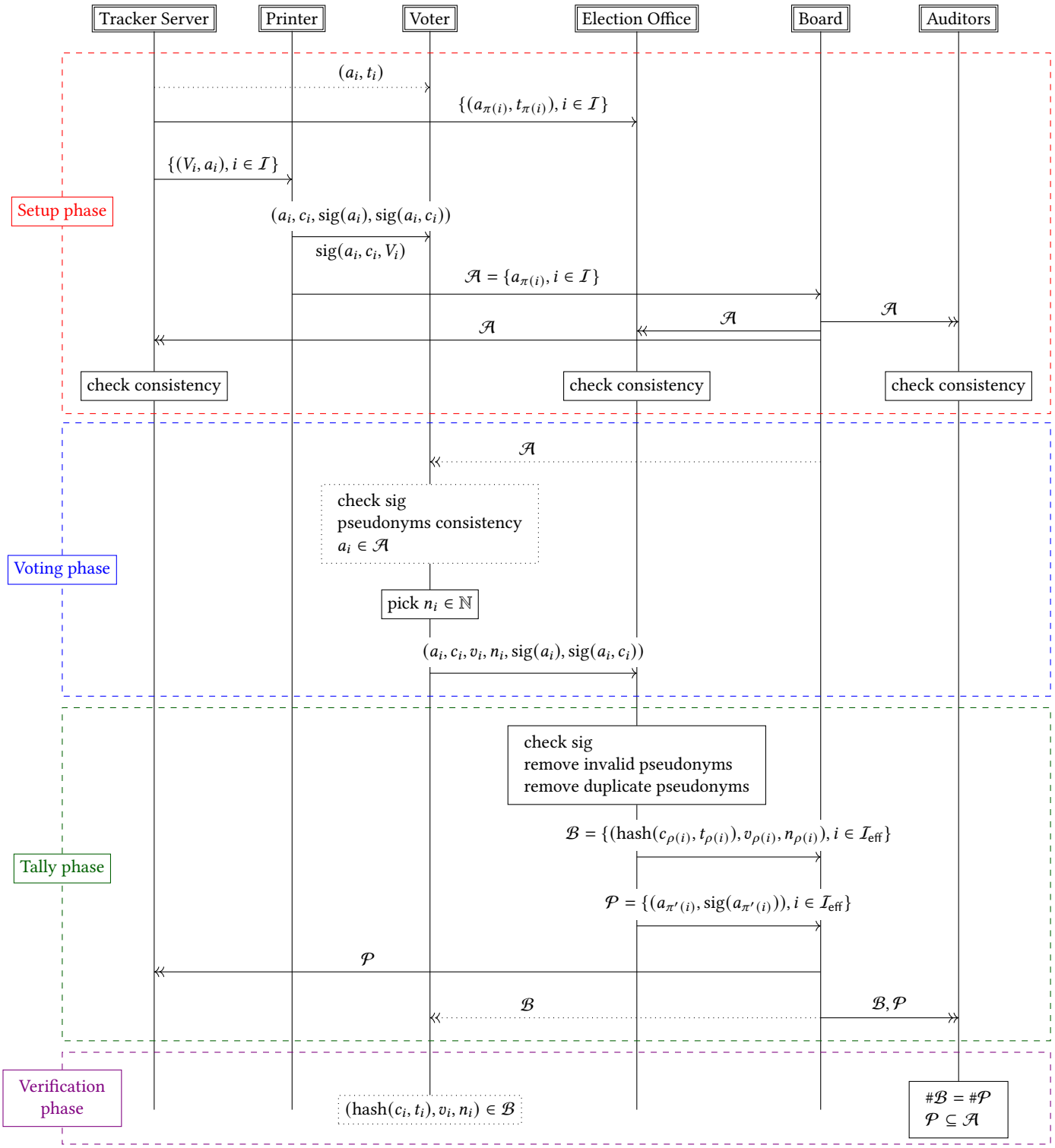

Figure 3: Vote&Check protocol. Dotted lines and boxes represent optional steps for the voter. The conscientious voters perform them, while the offline voters do not. Reading the Board is possible at any time, but we emphasize some natural moments for reading it with double-tipped arrows.

by removing several ballots for A. Instead, Vote&Check ensures privacy in the context of *anonymous complaints*, where we assume that complaining voters can anonymously contact a third party (a judge for example). In practice, a voter will lose privacy if they complain publicly, for example on a social network. But they will more reasonably be instructed to contact some dedicated authority that does not know which ballots have been removed.

3.3.2 Verifiability. Vote verifiability covers four main properties: *cast-as-intended*, i.e., the ballot cast by the voter contains their intended vote; *individual verifiability*, i.e., the ballot registered in the ballot box is the ballot cast by the voter; *universal verifiability*, i.e., the result corresponds to the ballots in the ballot box; *eligibility verifiability* the ballots only come from legitimate voters.

These properties are intuitive but they assume a particular setting (e.g., a global ballot box) and they may miss some attacks such as clash attacks [28] where two voters agree on the same ballot. Moreover, they do not cover the case of voters who do not verify, while it corresponds to the most frequent case. Hence, a more general notion of *end-to-end verifiability* has been proposed [17]. It guarantees that the result of the election is the disjoint union of:

- all the votes of conscientious voters. This combines the *individual verifiability* property and *no clash attack*.
- a subset of votes of honest voters who did not verify their votes (called offline voters). Intuitively, an attacker may always drop such votes hence at best a subset of these votes will be counted.
- a set of votes corresponding to the corrupted voters (as many as the number of corrupted voters).

In Vote&Check some properties come for free: the voter *sees* their vote on their ballot hence they are guaranteed that their ballot contains their intended vote. Moreover, anyone can count the votes on the ballot box since they appear in clear.

Since every conscientious voter can verify that their vote stands on the board next to their tracker $\text{hash}(c_i, t_i)$ and their anti-clash number n_i, then individual verifiability is guaranteed in Vote&Check, even if all authorities are dishonest. To obtain end-to-end verifiability, one must also control that no votes can be added for offline voters (or absentee voters). In Vote&Check, end-to-end verifiability is guaranteed only if both the Printer and the Election Office are honest. Let us explain why this is the case. If the Printer is compromised, since they have all the voting material, they can vote in place of absentees. Also, if the Election Office is corrupted, they can record every cast vote v_i into another vote v_i'. If the voter V_i does not perform the verification, this will be undetected. However, this attack no longer works if every voter that cast a ballot verifies.

In order to show that these are the only obstructions to end-to-end verifiability, we introduce a variant of end-to-end verifiability, called *weak verifiability* that now tolerates that votes from offline voters can be modified. Then Vote&Check guarantees weak verifiability as long as the Printer is honest.

4 Security analysis

To conduct a formal security analysis of Vote&Check, two main techniques exist: computational proofs (game-based or Universal composability) and symbolic proofs. These are complementary approaches: computational proofs allow an in-depth analysis of the underlying cryptographic primitives, while symbolic proofs allow to consider more subcases of the protocol and various corruption scenarios. Symbolic and computational proofs are both required by the Swiss Chancellery to get an Internet voting system approved for use in federal elections [41]. We chose to conduct a symbolic analysis using the state-of-the-art tool, ProVerif [7]. ProVerif demonstrated its ability to analyse (i.e., find flaws or prove security) complex protocols such as TLS [6], Signal [27], or LAKE-EDHOC [24], and in particular e-voting protocols Swiss Post [35], CHVote [5]. Moreover, a specialized framework [15] for proving verifiability of voting protocols has been designed based on ProVerif.

4.1 ProVerif

ProVerif analyses the security properties in a symbolic model where messages are modelled with terms, roles by processes, and the network by input/output on communication channels. We recall here the main features of the tool. Interested reader can refer to ProVerif documentation [9, 11] to get further details.

Terms are inductively defined as atomic values, e.g., `n,m,k`, or function symbols, representing cryptographic primitives, applied to terms, e.g., `f(t1,...,tn)`. Rewriting rules and/or equational theories equip terms to model the functional properties of the primitives.

More concretely, a digital signature is modelled as follows: we define a symbol `pk()` of arity one that produces a public verification key from a secret signing key `sk`, and a symbol `sign` of arity 2 that produces a signature from a message `m` and a key `sk`. Checking a signature is modelled by a symbol `check` and a rewriting rule that is, for all `m` and `sk`, the term `check(sign(m,sk),m,pk(sk))` rewrites as `true`. By default, ProVerif defines `true` and `false` two atomic value modelling boolean values. Moreover, it defines rewriting rules to model logical operations such as conjunction (`b1 && b2`), and disjunction (`b1 || b2`).

Communication channels are declared public or private. In the former case, the attacker has full control over it. It can read, intercept, modify, or inject messages. On the contrary, when private, the channel guarantees confidentiality, integrity, and authenticity.

Finally, the different roles of the protocol are modelled by processes that describe the actions done by each agent. The command `new n` allows to create a fresh nonce. These are atomic values, unknown from the attacker and indistinguishable from each other.

Communications are modelled by input and output: `in(c,x)` inputs a message on channel `c`; the variable `x` being instantiated by the input message upon reception. The command `out(c,m)` outputs message `m` on channel `c`. Agents can perform tests with the command `if b then P else Q`. For instance, the term `b` can be of the form `check(...)` which is expected to reduce to *true* when the check succeeds, or simply an equality test, `m1 = m2`. Another command can be used to perform tests: `let (a,b) = m in P` allows to test whether `m` is a pair and in that case extracts its two components.

Using *tables* ProVerif can model stateful protocols that rely on an append-only memory. Given a table `tbl`, a process can insert an element `m` using the command `insert tbl(m)`. It can then look for a specific entry using the command `get tbl(x) suchthat b in P else Q`. This command executes to `P` if there is an entry `m` in the table such that `b` (where occurrences of `x` are replaced by `m`) reduces

to `true`. Otherwise, it executes to `Q`. Tables are extensively used in the *verifiability framework* we are building on, to record data generated during the initialization of the protocol.

Finally, ProVerif allows to verify security protocols in rich scenarios by defining concurrent and replicable processes. The command `P|Q` denotes the concurrent execution of `P` and `Q`, and `!P` denotes that `P` can be replicated as often as desired. Concretely, `!P` can be rewritten as `P|!P` and is used, for instance, to model that there is an arbitrary number of voters or elections.

4.2 Model

Most actors are modelled in a natural way. For example, the role of the Voter is represented by the process depicted in Figure 4. The voter receives their voting material in their postbox, modelled by the `voter_letterbox(voter)` channel. The material includes in particular their pseudonym `a` and their credential `c`. They then check that the material is valid in order to avoid targeted privacy breach, as defined in Section 2.1. This is modelled here by checking the cryptographic signature to represent the fact that the voter receives some authenticated material. In practice, we can assume that the voters made a visual inspection of the received material, looking for example for an official stamp. The case where the voter may be fooled into using fake voting material is modelled by considering a dishonest Printer. Then the (conscientious) voter contacts the Tracker Server in order to get its pseudonym and tracker, and it checks that it corresponds to the pseudonym received from the Printer. This check is optional and won't be made by offline voters. The voter then simply selects their vote `v` and writes some (small) random number `n` on their voting sheet and send it by mail to the Election Office. This is modelled by sending `a, c, sig1, sig2, v, n` on the channel `deposit_letterbox`. Finally, once the election is tallied, the conscientious voter checks that their hashed tracker `hash(c, t)`, their vote `v` and their random `n` appear on the ballot box. They raise a complaint otherwise. Note that in case of success, the process executes an *event* `Happy(voter)`. In ProVerif, events play the role of "trace annotations" which are then used to express security properties (see Section 4.4). They do not interfere with the semantics of the process.

Physical channels. Because our protocol relies on physical channels, we have to model the postbox. For voters, this is a special channel `voter_letterbox(V)` where anyone can write (anyone can send a letter to a dedicated voter) but only the voter `V` can read (only the voter can open their postbox). This is modelled by defining `voter_letterbox` as a private function (that the attacker cannot use) but we let the attacker posts any message with an explicit process that writes anything on Alice's mailbox:

```
!(in(public, x: bitstring); out(voter_letterbox(Alice), x))
```

Dishonest authorities. Since Vote&Check involves 3 authorities (Tracker Server, Printer, Election Office), we consider multiple corruption scenario ($2^3 = 8$ in total) depending on who is honest. In most cases, this is easy to model: a corrupted party simply gives all its secrets to the adversary who can then send the messages they want on their behalf. The case of the Election Office requires more care. Indeed, when it is honest, it is in charge of writing on the Bulletin Board (BB). When dishonest, it can then freely control

```
let Voting(voter, v) =
  in(voter_letterbox(voter), (a, c, sig1, sig2, sig3)); (* The
       voter gets their ballot in their postbox *)
  if check(sig1, a, pk(sk_Pri)) then
  if check(sig2, (a, c), pk(sk_Pri)) then
  if check(sig3, (a, c, voter), pk(sk_Pri)) then
  in(trackerserver_to_voter(voter), (a', t)); (* Get the
       verification material from the Tracker Server *)
  if a = a' then (* Check that the identifier is correct *)
  new n;
  new date_sent;
  out(deposit_letterbox, ((a, c, sig1, sig2, v, n), date_sent)); (*
       Send their ballot *)
  in(bulletin_board, (hashv, =v, =n));
  if hashv = hash(c, t) then event Happy(voter)
  else out(public, complaint).
```

Figure 4: Process for the voter. This is a simplified version: for privacy the verification steps have to be modified (see Section 4.3) and for verifiability, the use of the framework imposes some changes as well (see Section 4.4).

the content of BB, up to the fact that BB is monitored by Auditors. The Auditors check in particular that the number of accepted votes corresponds to the number of valid signatures. We model this by letting a dishonest Election Office write on BB only if it can first produce a pseudonym a that is properly signed by the Printer. This lets the Election Office add one element in $\mathcal{B}$. To ensure that the Election Office does not reuse the same signed pseudonym several times, during the setup phase, the Printer outputs all a_i on a private channel `token_list` and the Election Office needs to input the a_i from this channel, consuming them one by one.

4.3 Privacy

Intuitively, vote privacy is preserved if an attacker cannot distinguish the case where Alice votes 0 and Bob votes 1 from the case where the two votes are swapped [20]. This can be written

$$\texttt{Voter(Alice,0)|Voter(Bob,1)}|S \approx \texttt{Voter(Alice,1)|Voter(Bob,0)}|S$$

where S represents the overall system that runs in parallel with the processes of Alice and Bob. The relation $P \approx Q$ is an observational equivalence [1] that intuitively states that an adversary cannot distinguish P from Q. The privacy property can equivalently be written in ProVerif as follows

$$\texttt{Voter(Alice,choice[0,1])} \mid \texttt{Voter(Bob,choice[1,0])} \mid S$$

in a way that highlights the only few differences from the two processes. If ProVerif returns true, then the process instantiated with the left part of the choice operator is observationally equivalent to the process instantiated with the right part of the choice operator.

Anonymous complaints. As explained in Section 3.3.1, Vote&Check is subject to privacy breach by complaints: the Election Office or the postman in charge of delivering the cast ballots could destroy some ballots for a certain candidate A and observe who complains. We hence prove privacy under the assumption that voters complain *anonymously*. This is modelled by a "double complaint": in the same way that the vote of Alice is protected by Bob voting in another way, the privacy of a complaining Alice is protected by a complaining

Bob voting in another way. Hence, we remove the lines 11 to 13 of the process `Voter` presented in Figure 4 and we replace it by a process `Verification` where Alice and Bob simultaneously check their ballots and a complaint is raised if any of the two checks fail, without letting the attacker knows which test failed.

```
1  let Verification =
2      in(voter_to_verifier(Alice), (hashA, vA, nA));
3      in(voter_to_verifier(Bob), (hashB, vB, nB));
4      in(bulletin_board, (hash0, v0, n0));
5      in(bulletin_board, (hash1, v1, n1));
6      if (hashA, vA, nA) = (hash0, v0, n0) &&
7          (hashB, vB, nB) = (hash1, v1, n1)
8        ||
9          (hashA, vA, nA) = (hash1, v1, n1) &&
10         (hashB, vB, nB) = (hash0, v0, n0)
11     then event Happy(Alice); event Happy(Bob)
12     else out(public, complaint).
```

The verification done by Alice and Bob is simulated by looking at the Bulletin Board and using with their respective data received on channels `voter_to_verifier(Alice)` and `voter_to_verifier(Bob)`. The disjunction in lines 7-10 handles the fact that the ballots of Alice and Bob may appear in any order on the Bulletin Board.

Offline voters. Voters may not want to contact the Tracker Server before casting their vote. In that case, they may be subject to a targeted privacy attack if the Printer is dishonest. Indeed, assume that the attacker wants to learn Alice's vote. A dishonest Printer could print official voting material with Alice's name on it but with Charlie's pseudonym `a'`. Then the dishonest voter Charlie could use his own verification mechanism, and in particular his tracker `t'` to find out Alice's vote that will be associated to `hash(c,t')`. We show however that this is the only additional risk w.r.t. privacy for offline voters. That is, we prove privacy for offline voters when the Printer is honest and either the Tracker Server or the Election Office is honest. Offline voters are easily modelled by removing all optional checks (process `Verification` and line 7 of process `Voting`).

Multiple permutations. In Vote&Check, vote privacy is not ensured through cryptographic mechanisms such as encryption. Instead, it relies on multiple shuffles:

- the ballots are implicitly shuffled by the postal services when the voters send back their ballots to the Election Office. This prevents the Election Office from learning who sent what;
- the Printer and the Tracker Server shuffle respectively the a_i and the (a_i, t_i) (possibly by sorting them alphabetically). This prevents the Election Office from linking the ballots to the actual identities;
- the Election Office handles $\mathcal{B}$ and $\mathcal{P}$ separately, with a different shuffle, which prevents the Tracker Server and the Printer from linking the votes to the voting material.

All these shuffles need to be properly modelled to prove privacy. This can be easily done taking advantage of the non-determinism in ProVerif. For example, the following process receives two messages on a channel c and outputs them in some non-deterministic order:

$$\texttt{in(c,x}_1\texttt{);in(c,x}_2\texttt{); (out(c,x}_1\texttt{) | out(c,x}_2\texttt{)).}$$

An attacker does not know, *a priori*, if x_1 is output first or second.

The issue is that ProVerif actually does not prove observational equivalence but *diff-equivalence* [8], a stronger notion that checks, step by step that the two processes take exactly the same action. In particular, if ProVerif has to prove $P_1 \mid P_2 \approx Q_1 \mid Q_2$, it will instead try to prove that any action of P_1 can be mapped to an action of Q_1 (and not Q_2) and that any action of P_2 can be mapped to an action of Q_2 (and not Q_1). Intuitively, ProVerif chooses an arbitrary scheduler to resolve non-determinism when proving diff-equivalence. Even if this proof strategy is sound, it may lead to false attacks. One can try to solve this issue by asking ProVerif to prove $P_1 \mid P_2 \approx Q_2 \mid Q_1$ instead, if one thinks that the resulting mapping will prevent reaching false attack. This *swapping* technique (Q_1 and Q_2 are swapped) was firstly introduced by Delaune et. al. [21] and then proved sound by Blanchet and Smyth [10]. We isolated the two voter processes of Alice and Bob (for which privacy is proved) and applied *swaps* when necessary. We let the other voter processes unchanged (since they are not critical to prove vote secrecy and thus do not lead to false attacks).

4.4 Verifiability

End-to-end verifiability has been informally defined in Section 3.3.2. It can be stated as correspondence properties between *events*, which are process annotations used to identify specific steps of the protocol. The main difficulty lies in the fact that the definition requires to count the votes, which is a difficult task for most of the verification tools in the symbolic setting.

Verifiability framework. A framework has been recently developed [15] in order to prove end-to-end (E2E) verifiability. The authors of [15] leverage *injective correspondence queries* [7] (a refinement of correspondence properties supported by ProVerif) to encode E2E verifiability. They show that proving E2E verifiability is equivalent to proving the following two properties:

- individual verifiability: all votes of voters who verified should be counted.

```
event(Finish()) && inj-event(Verified(v_id,v)) ==>
    inj-event(Counted(v))
```

- universal verifiability: counted votes come from honest voters that did cast these votes, plus a set of valid votes, whose size is bounded by the number of dishonest voters

```
event(Finish()) && inj-event(Counted(v)) ==>
     inj-event(HV(v_id)) && event(Verified(v_id,v))
  || inj-event(HNV(v_id)) && event(Voted(v_id,v))
  || inj-event(Corrupt(v_id))
```

The events are placed as expected: `Counted(v)` is emitted as soon as a vote is counted during the tally phase; `Finish()` occurs once the tally is over, events `Voted(v_id,v)` and `Verified(v_id,v)` happen respectively when voter `v_id` has sent their ballot, resp. has verified their vote on the bulletin board. The events `HV(v_id)` comes from the framework' terminology and distinguish respectively between the conscientious voters, the offline voters, and the dishonest voters.

Vote&Check achieves E2E verifiability, that is individual and universal verifiability, when both the Printer and the Election Office are honest.

Weak universal verifiability. Unfortunately, universal verifiability no longer holds when the Election Office is dishonest. Indeed, when receiving a ballot from an offline voter, it could easily modify the candidate' name. Note that the Election Office does not know the

origin of a ballot, so it has to bet that the corresponding voter will not check. We therefore consider a weaker version of universal verifiability, where an attacker is allowed to modify the votes of voters who do not verify. This notion of verifiability is considered in [30] for example.

```
event(Finish()) && inj-event(Counted(v)) ==>
   inj-event(HV(v_id)) && event(Verified(v_id,v))
|| inj-event(HNV(v_id)) && event(Voted(v_id,v')) (*v' instead of v*)
|| inj-event(Corrupt(v_id))
```

Vote&Check achieves weak universal verifiability as soon as the Printer is honest. Conversely, a dishonest Printer can always vote in the name of absentees and hence break weak universal verifiability.

Finally, individual verifiability is preserved in all cases: voters who verify are guaranteed that their votes will be counted. When all parties are corrupted, individual verifiability relies solely on the nonce chosen by the voters. However, if either the Printer or the Tracker Server is honest, we show that individual verifiability holds even if voters all use an empty nonce.

Identifying public identifiers. We had to adapt our model to make it fit into the framework. In particular, the verifiability framework assumes that each voter can be associated to a *public identifier* and that this public identifier can then be extracted from a ballot. The public identifier is typically the signing key of the voter or another form of voting credential. However, in Vote&Check, there is no such public identifier. A natural candidate is the credential c provided by the Printer. However, there are immediately two issues. First, c cannot be read from a ballot of the form h, v, n since it is hidden by the hash. Second, when the Printer is dishonest, the credential can no longer be trusted. Indeed, a dishonest Printer could give the same credential to several voters, in order to try to perform a clash attack (see Section 2.1.3). Instead, one could reason on the tracker t provided by the Tracker Server. But again, the tracker cannot be trusted if the Tracker Server is compromised. Actually, when both the Printer and the Tracker Server are compromised, only the fresh nonce chosen by the voter still provide some verifiability.

To circumvent this issue, we proceed in two steps:

(1) We define as public identifier all material that is used by the voter, namely $a, \mathrm{hash}(c, t), n$. Note that it also includes n, the nonce generated by the voter. This identifier is added in a table `public_identifier` by the framework.
(2) We provide a dynamic association between ballots and public identifiers, depending on the honesty status of each party. Intuitively, this association is computed "magically", knowing all the private information of (honest) participants.

We believe that our approach is rather systematic and could be applied on other voting contexts, when the notion of public identifier is a blurry concept.

In Vote&Check, we compute this association through the function `get_ident_from_ballot` that associates an identifier to a ballot $a, \mathrm{hash}(c, t), v, n$. Note that a and $\mathrm{hash}(c, t), v, n$ are published separately on the bulletin board. When all parties behave honestly, `get_ident_from_ballot` returns $a, \mathrm{hash}(c, t), n$ as expected. When the Printer is honest and the other parties might be dishonest, given a ballot $a_1, \mathrm{hash}(c, t_1), v, n_1$, we look for a voter that used the credential c provided by the Printer, with an identifier of the form $a_2, \mathrm{hash}(c, t_2), v, n_2$ and we return $a_2, \mathrm{hash}(c, t_2), v, n_2$ since this ballot must have been built by the voter that received c, hence identifier by $a_2, \mathrm{hash}(c, t_2), v, n_2$. In case no such voter exist (for example, c has been given to a dishonest voter), then we return the identifier associated to a_1, This is our default case: any accepted ballot can be associated to one of the a's since the number of accepted ballots is bounded by the number of (signed) a's. This property is enforced by the Auditors.

```
1  letfun get_ident_from_ballot(bal) =
2    let ballot_of_bit((a1,h1, v,n)) = bal in
3    let c1 = getc(h1) in (* returns c such that h1=h(c,_) *)
4    if printer_status = honest
5    then (
6      get public_identifier(_, ident_of_triplet(a2, h2, n2)) suchthat
           c1 = getc(h2) in
7        ident_of_triplet(a2, h2, n2)
8      else
9       get public_identifier(_, ident_of_triplet(a2, h2, n2)) suchthat
            a1 = a2 in
10       ident_of_triplet(a2, h2, n2)
11     else dummy_ident ) (* this should never happen *)
12   else ... (* all other cases of corruption *)
13     else
14     get public_identifier(_, ident_of_triplet(a2, h2, n2)) suchthat
           a1 = a2 in
15       ident_of_triplet(a2, h2, n2)
16     else dummy_ident (* this should never happen *)
```

Proving verifiability. The verifiability framework [15] comes with a library of lemmas that take care of proving individual and universal verifiability, two injective properties, thanks to a variety of counters and temporal constraints. However, we detected a small discrepancy in the framework, acknowledged by the authors. The framework distinguishes between three types of voters: conscientious voters, honest voters that do not verify (called here offline voters), and dishonest voters. However, this was inaccurately modelled by first generating a voter and then, later on, assigning it to one of the three types, at the adversary's choice. This actually led to a fourth type of voters: voters that were not yet assigned to any category but that the adversary could try to use to vote on their behalf. We corrected this issue by requiring that all voters are assigned to one of the three types.

Then a remaining issue is caused by our complex function `get_ident_from_ballot` that calls the table `public_identifier` multiple times, with many `else` branches. Due to its internal translations into first order logic, ProVerif does not properly capture the fact that, whenever a `else` branch is considered, the previous conditions in the `if` branches must be falsified. In particular, if some term t has been inserted in a table `tab` and if the term t satisfies some condition D that appears in a branch of the form `get tab(x) suchthat D ... else event E` then the event `E` can no longer be emitted in the trace. We prove this implication and we add as an axiom the conclusion of the implication, as soon as the assumptions can be proved by ProVerif (as a set of lemmas). This is formally stated and proved in Appendix B.4.

4.5 Results

All our ProVerif files are available as supplementary material in [16]. The verifiability framework [15] requires to use a special version of the ProVerif prover. We used the commit `cc4f8cde5` of the

Corruption scenario			Privacy		Verifiability			
Tracker server	Printer	Election office	Offline voters	Conscientious voters	Common lemmas	Universal	Weak univ.	Individual
H	H	H	✔ (12.3 s)	✔ (12.3 s)	✔ (9.0 s)	✔ (0.5 s)	✔ (0.6 s)	✔ (0.4 s)
C	H	H	✔ (15.7 s)	✔ (15.6 s)	✔ (9.5 s)	✔ (0.4 s)	✔ (0.5 s)	✔ (0.3 s)
H	C	H	✘	✔ (2.1 s)	✔ (26.4 s)	✘	✘	✔ (0.3 s)
H	H	C	✔ (0.4 s)	✔ (0.4 s)	✔ (39.6 s)	✘	✔ (3.9 s)	✔ (0.5 s)
H	C	C	✘	✘	✔ (6.2 s)	✘	✘	✔ (0.2 s)
C	H	C	✘	✘	✔ (45.0 s)	✘	✔ (2.9 s)	✔ (0.6 s)
C	C	H	✘	✘	✔ (32.3 s)	✘	✘	✔ (0.2 s)
C	C	C	✘	✘	✔ (8.1 s)	✘	✘	✔ (0.2 s)

Figure 5: Security properties. Each line gives a corruption scenarios, where H means that the entity is honest, and C means that it colludes with the attacker. For each property, the ✔ symbol means that it has been proven in ProVerif, in the given amount of time; the ✘ means that we know that there is an attack. The special ✔ symbol means that in this case, the individual verifiability holds only if different voters uses different n, which is weak, since n is short. Privacy holds no matter if n is short or long.

`improved_scope_lemma` branch, from the official development repository. All experiments were run on a standard laptop equipped with a 4-core Intel i7-8665U CPU, with 16 GB of RAM; the running times are given for this machine. The resulting analysis is displayed in Figure 5 and confirms our security claims. The common lemmas refer to the lemmas provided by the framework, meant to help proving verifiability, and our own lemmas needed to infer our new axiom, as explained in Section 4.4.

Vote&Check preserves vote privacy when no more than one authority is corrupted. It always provides individual verifiability: the votes of conscientious voters are counted. Full end-to-end verifiability, that is, votes of offline voters can only be dropped, requires an honest Printer and Election Office. If we relax to weak verifiability (votes of offline voters may be changed), then we only need that the Printer is honest. Indeed, the Printer possesses the entire voting material hence it can always vote for absentee voters.

5 Discussion

Vote&Check achieves a better level of verifiability and privacy than STROBE, RemoteVote and SAFE Vote, while keeping a similar infrastructure: all protocols require a printer and a collecting authority (the Election Office). In Vote&Check, we introduce a Tracker Server, that replaces the set of decryption authorities used in STROBE, RemoteVote and SAFE Vote. Compared to Devillez et al, the security level is similar However, we reduce significantly the number of independent authorities (from 4 + a set of decryption authorities to 3) and we recover full universal verifiability (no proxy).

Vote&Check remains however subject to several attacks.

Privacy breach by complaints. This attack seems unavoidable as soon as the votes are sent in clear, which is an important feature for usability. The Election Office can selectively remove ballots that vote for a certain candidate A and see who complains. To circumvent this issue, one would need to provide a complaint mechanism that preserves vote privacy, e.g., encouraging voters to launch false complaints to hide true ones. But this also requires a robust accountability mechanism to avoid accepting wrong complaints. So in short, there is no easy solution to prevent privacy breach by complaints.

Weak eligibility. The printer in charge of printing and sending the material to voters may vote in place of absentees. To improve on this, one could use several printers that each sends a part of the voting material. Such an approach has cost and usability issues. Alternatively, voters may be requested to write down some authentication token, obtained for example through a mobile application. This however means that the solution is no longer purely paper based, which may cause some usability issue.

Alter votes of non verifying voters. The Election Office may always drop some ballots. If the corresponding voters do not check, this will remain undetected. This is indistinguishable from the case where the corresponding voters did not vote. However, in Vote&Check as well as all the other protocols under study, the Election Office may not only drop but also change a vote for another candidate. Here, there could be a room for improvement, for example biding an authenticated secret data to each voting option. The Election Office would not know the secret associated to another vote. This should be carefully design to preserve vote privacy (and usability).

Limitations. As many existing voting schemes, Vote&Check is not *accountable*: when a voter detects an issue, there is no mechanism to identify who misbehaved. Such mechanisms exist in some protocols (e.g., sElect [29], Themis [12]) but, they still have limitations: they only identify which authority misbehaved, assuming the voter behaves honestly. When considering cast-as-intended, a voter may lie (pretend to have voted for B while having voted for A) Given that voters typically cannot apply any cryptographic mechanism (only their possibly corrupted device may perform such operations), it is hard to distinguish between an honest voter who has been attacked and a dishonest voter who wishes to diminish the trust in the election. Such an issue is very challenging and still largely unaddressed, even for Internet voting protocols.

Another limitation of Vote&Check is that it is subject to coercion attacks. Indeed, a coerced voter may be instructed to choose a particular value n to be written on their ballot and to vote for A. Then the coercer expects to see A, n on the Board. Resisting against vote selling is difficult in the context of postal voting. Indeed, a voter may typically provide their voting material to the coercer. One possible direction would be to design a protocol such that the voters can forge voting material, whose corresponding votes will, undetectably, be removed during the tally.

References

[1] Martin Abadi and Cédric Fournet. 2001. Mobile Values, New Names, and Secure Communication. In *POPL'01*. ACM, 104–115.
[2] Ben Adida. 2008. Helios: Web-based Open-Audit Voting. In *USENIX'08*. 335–348.
[3] Josh Benaloh. 2021. STROBE-Voting: Send Two, Receive One Ballot Encoding. In *E-Vote-ID'21*. Springer, 33–46.
[4] Josh Benaloh, Peter Y. A. Ryan, and Vanessa Teague. 2013. Verifiable Postal Voting. In *Security Protocols XXI*. Springer, 54–65.
[5] David Bernhard, Véronique Cortier, Pierrick Gaudry, Mathieu Turuani, and Bogdan Warinschi. 2018. Verifiability Analysis of CHVote. Cryptology ePrint Archive, Paper 2018/1052. https://eprint.iacr.org/2018/1052
[6] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. 2017. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *S&P'17*. 483–502.
[7] Bruno Blanchet. 2016. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends in Privacy and Security* 1 (2016), 1–135.
[8] Bruno Blanchet, Martín Abadi, and Cédric Fournet. 2005. Automated Verification of Selected Equivalences for Security Protocols. In *LICS'05*. IEEE, 331–340.
[9] Bruno Blanchet, Vincent Cheval, and Véronique Cortier. 2022. Proverif with lemmas, induction, fast subsumption, and much more. In *S&P'22*. IEEE, 69–86.
[10] Bruno Blanchet and Ben Smyth. 2018. Automated reasoning for equivalences in the applied pi calculus with barriers. *Journal of Computer Security* 26, 3 (2018), 367–422.
[11] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. 2023. ProVerif 2.05: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial. (2023).
[12] Mikaël Bougon, Hervé Chabanne, Véronique Cortier, Alexandre Debant, Emmanuelle Dottax, Jannik Dreier, Pierrick Gaudry, and Mathieu Turuani. 2022. Themis: an On-Site Voting System with Systematic Cast-as-intended Verification and Partial Accountability. In *CCS'22*. ACM.
[13] Alessandro Bruni, Eva Drewsen, and Carsten Schürmann. 2017. Towards a Mechanized Proof of Selene Receipt-Freeness and Vote-Privacy. In *E-Vote-ID'17*. Springer, 110–126.
[14] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. 2008. Civitas: Toward a Secure Voting System. In *S&P'08*. IEEE.
[15] Véronique Cortier, Alexandre Debant, and Vincent Cheval. 2023. Election Verifiability with ProVerif. In *CSF'23*. IEEE.
[16] Véronique Cortier, Alexandre Debant, Pierrick Gaudry, and Léo Louistisserand. 2025. Vote&Check: Secure Postal Voting with Reduced Trust Assumptions. In *PoPETs'25*. https://inria.hal.science/hal-04813613
[17] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. 2016. SoK: Verifiability Notions for E-Voting Protocols. In *S&P'16*. IEEE.
[18] Braden L. Crimmins, Marshall Rhea, and J. Alex Halderman. 2022. RemoteVote and SAFE Vote: Towards Usable End-to-End Verification for Vote-by-Mail. In *FC'22*. Springer, 391–406.
[19] Alexandre Debant and Lucca Hirschi. 2023. Reversing, Breaking, and Fixing the French Legislative Election E-Voting Protocol. In *Usenix'23*.
[20] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. 2010. Symbolic bisimulation for the applied pi calculus. *Journal of Computer Security* 18 (2010), 317–377.
[21] Stéphanie Delaune, Mark Ryan, and Ben Smyth. 2008. Automatic verification of privacy properties in the applied pi calculus. In *IFIP International Conference on Trust Management*. Springer, 263–278.
[22] Henri Devillez, Olivier Pereira, and Thomas Peters. 2024. Verifiable and Private Vote-by-Mail. Cryptology ePrint Archive, Paper 2024/926. https://eprint.iacr.org/2024/926.
[23] J. Alex Halderman and Vanessa Teague. 2015. The New South Wales iVote System: Security Failures and Verification Flaws in a Live Online Election. In *VoteID'15*. Springer.
[24] Charlie Jacomme, Elise Klein, Steve Kremer, and Maïwenn Racouchot. 2023. A comprehensive, formal and automated analysis of the EDHOC protocol. In *USENIX'23*. Anaheim, CA, United States.
[25] Ari Juels, Dario Catalano, and Markus Jakobsson. 2005. Coercion-resistant electronic elections. In *WPES'05*. ACM, 61–70.
[26] Christian Killer and Burkhard Stiller. 2019. The Swiss Postal Voting Process and Its System and Security Analysis. In *E-Vote-ID'19*. Springer, 134–149.
[27] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. 2017. Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach. In *EuroS&P'17*. IEEE, 435–450.
[28] Ralf Kuesters, Tomasz Truderung, and Andreas Vogt. 2012. Clash Attacks on the Verifiability of E-Voting Systems. In *S&P'12*. IEEE.
[29] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. 2016. sElect: A Lightweight Verifiable Remote Voting System. In *CSF'16*. IEEE.
[30] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2010. Accountability: definition and relationship to verifiability. In *CCS'10*. ACM, 526–535.
[31] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. 2020. VoteAgain: A scalable coercion-resistant voting system. In *USENIX'20*.
[32] Ülle Madise and Tarvi Martens. 2006. E-voting in Estonia 2005. The first Practice of Country-wide binding Internet Voting in the World. In *EVOTE'06*. GI.
[33] Eleanor McMurtry, Xavier Boyen, Chris Culnane, Kristian Gjøsteen, Thomas Haines, and Vanessa Teague. 2021. Towards Verifiable Remote Voting with Paper Assurance. *CoRR* abs/2111.04210 (2021).
[34] Swiss Post. 2023. https://www.post.ch/de/ueber-uns/aktuell/2023/wie-die-schweiz-zur-briefwahl-nation-wurde. Blog article.
[35] Swiss Post. 2024. Swiss Post symbolic models - Version 1.3. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/master/Symbolic-models.
[36] Swiss Post. 2024. Swiss Post Voting System Specification - Version 1.4.1.1. https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/master/System.
[37] Peter Ryan, Peter Roenne, and Simon Rastikian. 2021. Hyperion: An Enhanced Version of the Selene End-to-End Verifiable Voting Scheme. In *E-Vote-ID'21*.
[38] Peter Ryan, Peter Rønne, and Vincenzo Iovino. 2016. Selene: Voting with Transparent Verifiability and Coercion-Mitigation. In *VOTING'16*. Springer.
[39] Peter Y. A. Ryan, David Bismark, James Heather, Steve A. Schneider, and Zhe Xia. 2009. Prêt à voter: a voter-verifiable voting system. *IEEE Trans. Inf. Forensics Secur.* (2009).
[40] Peter Y. A. Ryan and Vanessa Teague. 2009. Pretty Good Democracy. In *17th International Workshop on Security Protocols XVII*. Springer, 111–130.
[41] Swiss Federal Chancellery. 2022. Federal Chancellery Ordinance on Electronic Voting (OEV). https://www.fedlex.admin.ch/eli/cc/2022/336/en.
[42] MIT Election Lab team. 2021. How we did vote in 2020 – A topical look at the survey of the performance of American elections. https://electionlab.mit.edu/articles/how-we-voted-2020.

Acknowledgments

This work benefited from funding managed by the French National Research Agency under the France 2030 programme with the reference ANR-22-PECY-0006.

A Recommended size for the data in Vote&Check

To ensure security, some data must be large enough, but others don't need to be, and other can't be, because they involve human interactions.

- The role of the a_i is to be a pseudonym in order to provide vote secrecy with respect to the Election Office. The only requirement is that they are all distinct. They could, in principle be taken as integers in the interval $[1, \#\mathcal{V}]$.
- The c_i's and the t_i's must be hard to guess. Furthermore, all the $\mathrm{hash}(c_i, t_i)$ must be distinct (with high probability), and this can not be checked during setup. Taking them as uniformly random bit strings of 128 bits makes the result of the hash function also uniformly random as soon as the Tracker Server and the Printer are not both dishonest, in the random oracle model. The probability of collision is then negligible.
- The nonce n_i is chosen and written by the voter and must therefore be short. On the other hand, it must be large enough to ensure that a clash attack will go undetected with only a small (but non-negligible) probability. We suggest n_i to be a 3-digit number.

This is summarized in Table 2.

B Soundness of the axiom

In this section we formally state and prove the soundness of the axiom, informally mentioned in Section 4.4, and used in the ProVerif

Notation	Role	Created by	Size (bits)
a_i	pseudonym	Tracker Server	$\log(\#\mathcal{V})$
c_i	credential	Printer	128
t_i	tracker	Tracker Server	128
n_i	voter's nonce	Voter	10

Table 2: Elements involved in a ballot and its verification.

models to prevent the tool from deriving false attacks by over-approximating `else` branches. We first recall parts of ProVerif theory needed to establish our result.

While ProVerif defines an operational semantics between configurations, for sake of simplicity and understandability, we only consider abstract traces made of events and predicates that highlight specific actions. Mapping executions in ProVerif semantics to our abstract traces shall be straightforward. Interested readers can refer to [7] for a comprehensive description of ProVerif theory.

B.1 Terms

Messages are modelled with *terms* that can either be a variable $x \in \mathcal{X}$, a name $n \in \mathcal{N}$, or a function symbol applied to terms, i.e. $f(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are terms and $f/n \in \Sigma$ a function symbol of arity n. The set of function symbols Σ is split in two disjoint subsets: Σ_c that contains *constructor* symbols, and Σ_d the *destructor* symbols. Without loss of generality, we assume that $\{\mathsf{true}, \mathsf{false}\} \subseteq \Sigma_c$ and that they represent the two boolean values.

The set of messages is denoted $\mathcal{T}(\Sigma, \mathcal{X} \cup \mathcal{N})$. A term is said *destructor-free* if it is does not contain destructor symbols. A *ground term* t is a term that does not contain variables, i.e that belongs to $\mathcal{T}(\Sigma, \mathcal{N})$. We denote $vars(t)$ the set of variables occurring in t ($vars(t) = \emptyset$ when t is a ground-term). Terms can be instantiated thanks to *substitutions*: a substitution σ is a mapping from variables to terms, denoted $\sigma = \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$. Given a term t and a substitution σ, if $vars(t\sigma) = \emptyset$, we say that σ is a *grounding substitution* for t. We say that a substitution σ' extends σ if $\sigma' = \sigma \cup \{y_1 \mapsto v_1, \ldots, y_m \mapsto v_m\}$ and y_is are distinct from x_is.

The term algebra is equipped with a finite set $\mathcal{R}$ of rewriting rules. A term t can be rewritten in a term t' if there exists a position p in t, a rewriting rule $l \to r$, and a substitution σ such that $t_{|p} = u$, $u = l\sigma$, and t' is equal to t in which the term u at position p has been replaced by the term $r\sigma$.

We only consider sets of rewriting rules that yield a convergent system. Given a term t, we thus note $t\Downarrow$ the destructor-free term obtained after the application of the rewriting rules on t. When no such term exist, we denote $t \Downarrow = \mathsf{fail}$.

B.2 Protocol

A protocol is defined by its (infinite) set of execution traces. We note $\mathcal{E}$ a specific set of function symbols used to define events, and $\mathcal{T}bl$ a specific set of function symbols used to define tables. For our reasoning, we only need to consider some particular actions in a protocol execution.

More precisely, a *trace* is a finite sequence of actions of the following form:

$$
\begin{array}{rcll}
tr & := & 0 & \text{empty trace} \\
 & | & \mathsf{event}(E(e(u_1, \ldots, u_n))).tr & \text{event} \\
 & | & \mathsf{insert}(tbl(u_1, \ldots, u_n)).tr & \text{table insertion} \\
 & | & \mathsf{getSucc}(D, tbl(u_1, \ldots, u_n)).tr & \text{table get success} \\
 & | & \mathsf{getFail}(D, tbl).tr & \text{table get failure}
\end{array}
$$

where $u_1, \ldots, u_n \in \mathcal{T}(\Sigma_c, \mathcal{N})$, $e \in \mathcal{E}$ is an event symbol, $tbl \in \mathcal{T}bl$ is a table symbol, and $D \in \mathcal{T}(\Sigma, \mathcal{X} \cup \mathcal{N})$ is a term.

Intuitively, the term D represents the condition which appears in the ProVerif command `get tbl(x) suchthat D in P else Q`. Hence, D is a term that is expected to reduce to a boolean value. This is formally defined now.

An *execution trace* is a trace $tr = tr_1.\ldots.tr_n$ such that for all $i \in \{1, \ldots, n\}$ we have:

- tr_i is an event; or
- tr_i is a table insertion; or
- $tr_i = \mathsf{getSucc}(D, tbl(u_1, \ldots, u_n))$ and there exists an index $j < i$ such that $tr_j = \mathsf{insert}(tbl(u_1, \ldots, u_n))$ and $D\{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}\Downarrow = \mathsf{true}$ where $\{x_1, \ldots, x_n\} = vars(D)$; or
- $tr_i = \mathsf{getFail}(D, tbl)$ and for all indices $j < i$ if we have $tr_j = \mathsf{insert}(tbl(u_1, \ldots, u_n))$ then $D\{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}\Downarrow = \mathsf{fail}$ where $\{x_1, \ldots, x_n\} = vars(D)$.

B.3 Queries

Even if ProVerif supports complex queries we only consider simple ones in our model. These last are expressive enough to state and prove our soundness result.

In what follows, we consider only six types of predicates:

- $\mathsf{p{-}event}(e(t_1, \ldots, t_n))$ where $e \in \mathcal{E}$ and $t_1, \ldots, t_n \in \mathcal{T}(\Sigma_c, \mathcal{X})$,
- $\mathsf{p{-}insert}(tbl(t_1, \ldots, t_n))$ where $tbl \in \mathcal{T}bl$ and $t_1, \ldots, t_n \in \mathcal{T}(\Sigma_c, \mathcal{X})$.
- $\mathsf{p{-}getSucc}(tbl(t_1, \ldots, t_n))$ where $tbl \in \mathcal{T}bl$ and $t_1, \ldots, t_n \in \mathcal{T}(\Sigma_c, \mathcal{X})$.
- $\mathsf{p{-}getFail}(D, tbl)$ where $tbl \in \mathcal{T}bl$ and $D \in \mathcal{T}(\Sigma, \mathcal{X})$.
- $u = v$ where $u, v \in \mathcal{T}(\Sigma, \mathcal{X})$.
- $i > j$ where $i, j \in \mathcal{V}_t$, $\mathcal{V}_t$ being a specific set of variables used to model timing annotations[1].

We consider queries of the form $F_1@i_i \wedge \ldots \wedge F_n@i_n \Rightarrow F@i$ where $F, F_1 \ldots, F_n$ are predicates, and $i, i_1, \ldots, i_n \in \mathcal{V}_t$. Timing annotations apply to the four first predicates only. When not applicable or not necessary, timing annotations i_j are be omitted. Moreover, we assume that $vars(F) \subseteq \bigcup_{j=1\ldots n} vars(F_j)$.

We define the satisfaction relation $\vdash$ as follows: given an execution trace $tr = tr_1.\ldots.tr_p$ and a grounding substitution σ, $tr, \sigma \vdash \mathsf{p{-}event}(e(t_1, \ldots, t_n))@i$ if $tr_{i\sigma} = \mathsf{event}(e(t_1\sigma, \ldots, t_n\sigma))$. The satisfaction relation is defined similarly for $\mathsf{p{-}insert}(\cdot)$, $\mathsf{p{-}getSucc}(\cdot)$, and $\mathsf{p{-}getFail}(\cdot, \cdot)$.

Note that the $\mathsf{p{-}getSucc}(\cdot)$ predicate records only the table element used to pass the get. In contrast, to ease the formal development, the $\mathsf{p{-}getFail}(\cdot, \cdot)$ keeps track of the condition D.

[1] It is important to note that these variables cannot appear in other predicates/events/...

The satisfaction relation is then extended as expected to conjunction of predicates and implications following the usual interpretation of logical operators. More specifically, a protocol $\mathcal{P}$ (i.e. a set of execution traces) satisfies a query $\rho = (F_1@i_1 \wedge \ldots \wedge F_n@i_n \Rightarrow F@i)$, noted $\mathcal{P} \vdash \rho$ if for all execution trace $tr \in \mathcal{P}$ and for any grounding substitution σ such that $tr, \sigma \vdash F_1@i_1 \wedge \ldots \wedge F_n@i_n$, there exists a grounding substitution σ' extending σ such that $tr, \sigma' \vdash F@i$ and $i\sigma' \leq max_{j=1\ldots n}(i_j\sigma)$.

Encoding from ProVerif models. If events are a standard command in ProVerif, it is not the case for the specific events that we have introduced. However, any ProVerif model can be easily annotated to correspond to the model described just before. To do so, any command `insert tbl(u1,...,un)` is preceded by a specific `event Insert(tbl(u1,...,un))` used to record it in the trace in ProVerif labelled operational semantics. Similarly, an event getSucc() is placed at the very beginning of the positive branch of a `get` command to record the action. Similarly, a specific event getFail() is placed at the very beginning of the negative branch of a `get` command. However, unlike $\mathsf{p{-}getSucc}(\cdot)$ that only record the entry used to pass the test, $\mathsf{p{-}getFail}(\cdot,\cdot)$ predicate records the table but also the test applied to enter in the else branch. Unfortunately, ProVerif does not allow to encode such an element into an event. For sake on simplicity, we thus simply assume that the specific event is tagged with a fresh name to uniquely identify the `get` command that failed. This extra annotation is easy to implement.

B.4 Soundness result

Intuitively, we state that if two events F_1 and F_2 guarantee that a tem t has been successfully retrieved from a table tbl satisfying a condition D, then once F_1 and F_2 are executed in a trace, it is no longer possible to take the else branch of a command of the form `get tbl(x) suchthat D in P else Q`.

Let $\mathcal{P}$ be a protocol, i.e. a set of execution traces.
Let $\rho_1 = F_1@i_1 \wedge F_2@i_2 \Rightarrow \mathsf{p{-}getSucc}(tbl(u_1,\ldots,u_n))@i_s$.
Let $\rho_2 = \mathsf{p{-}getSucc}(tbl(y_1,\ldots,y_n)) \Rightarrow D\{x_1 \mapsto y_1,\ldots,x_n \mapsto y_n\} \Downarrow = \mathsf{true}$.
Let $\rho_3 = F_1@i_1 \wedge F_2@i_2 \Rightarrow i_1 > i_2$.
Let $\rho_4 = F_1@i_1 \wedge F_2 \wedge \mathsf{p{-}getFail}(D, tbl)@i_f \Rightarrow i_1 < i_f$.
Let $ax = F_1@i_1 \wedge F_2 \wedge \mathsf{p{-}getFail}(D, tbl)@i_f \Rightarrow \mathsf{false}$.

Proposition B.1. *If $\mathcal{P} \vdash \rho_1 \wedge \rho_2 \wedge \rho_3 \wedge \rho_4$ then $\mathcal{P} \vdash ax$.*

Proof. Let $tr \in \mathcal{P}$ be an execution trace and σ a grounding substitution such that $tr, \sigma \vdash F_1@i_1 \wedge F_2 \wedge \mathsf{p{-}getFail}(D, tbl)@i_f$.

Since $\mathcal{P} \vdash \rho_1$ we know that there exists a grounding substitution σ' extending σ_α (σ_α is equal to σ up to an α-renaming of variables occurring in predicates and timing annotations) such that $tr, \sigma' \vdash \mathsf{p{-}getSucc}(tbl(u_1,\ldots,u_n))@i_s$. By definition of the satisfaction of the predicate, we have that $tr_{i_s\sigma'} = \mathsf{getSucc}(\_, tbl(u_1\sigma',\ldots,u_n\sigma'))$ and $i_s\sigma' \leq max(i_1^1\sigma', i_2^1\sigma') = max(i_i\sigma, i_2\sigma)$ (i_1^1 and i_2^1 being the timing annotation occurring in ρ_1). By definition of an execution trace, we know that there exists $i_{\mathsf{insert}} \leq i_s\sigma'$ such that $tr_{i_{\mathsf{insert}}} = \mathsf{insert}(tbl(u_1\sigma',\ldots,u_n\sigma'))$.

Moreover, because $\mathcal{P} \vdash \rho_2$ we have that $tr, \sigma'_\alpha \vdash \rho_2$ (where σ'_α is equal to σ', up to, as before, an α-renaming of variables occurring in predicates), and thus $D\{x_1 \mapsto u_1\sigma',\ldots,x_n \mapsto u_n\sigma'\} \Downarrow = \mathsf{true}$.

Let us show that $i_f\sigma' < i_1\sigma$. Applying the definition of the satisfaction relation to $tr, \sigma \vdash \mathsf{p{-}getFail}(D, tbl)@i_f$ and considering the table element $tbl(u_1\sigma',\ldots,u_n\sigma')$ we deduce that

$$\begin{array}{lcll} i_f\sigma' & < & i_{\mathsf{insert}} & (def.\ \mathsf{getFail}(\cdot)) \\ & < & i_s\sigma' & (def.\ \mathsf{getSucc}(\cdot)) \\ & < & max(i_1^1\sigma', i_2^1\sigma') & (def.\ \Rightarrow) \\ & < & max(i_1\sigma, i_2\sigma) & (def\ of\ \sigma') \\ & < & i_1\sigma & (max(i_1\sigma, i_2\sigma) = i_1\sigma\ by\ \rho_3) \end{array}$$

We hence derive a contradiction with ρ_4.
We conclude that $tr, \sigma \vdash ax$, hence $\mathcal{P} \vdash ax$. □