Formal Verification and Solutions for Estonian E-Voting

Sevdenur Baloglu
sevdenur.baloglu@uni.lu
University of Luxembourg
Luxembourg

Sergiu Bursuc
sergiu.bursuc@uni.lu
University of Luxembourg
Luxembourg

Sjouke Mauw
sjouke.mauw@uni.lu
University of Luxembourg
Luxembourg

Jun Pang
jun.pang@uni.lu
University of Luxembourg
Luxembourg

ABSTRACT

Estonia has been deploying electronic voting for its government elections since 2005. The underlying e-voting system and protocol have been continuously improved, aiming to fix the vulnerabilities found over the years and to provide election verifiability, which is now the standard way to ensure election integrity despite corrupt infrastructure or parties. Another goal is receipt-freeness, to ensure privacy even if voters are coerced. However, several recent attacks against its verifiability and privacy show the need of rigorous, realistic formal specifications for the protocol and its security, of new solutions to mitigate attacks, and of automated security proofs to ensure all attacks have been covered. In this paper we propose:

- a formal specification of the Estonian E-Voting protocol in a symbolic model suited for automated verification tools;
- a general symbolic model for specifying privacy and receipt-freeness in presence of corrupt parties and infrastructure;
- automated verification of security with respect to an exhaustive set of corruption scenarios, discovering new attacks on verifiability (with Tamarin) and on privacy (with ProVerif).
- new solutions, focused on practical deployment and ease of use, and their automated proofs of security.

CCS CONCEPTS

• **Security and privacy → Formal security models**.

KEYWORDS

Formal verification, E-voting, Verifiability, Privacy

ACM Reference Format:
Sevdenur Baloglu, Sergiu Bursuc, Sjouke Mauw, and Jun Pang. 2024. Formal Verification and Solutions for Estonian E-Voting. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3634737.3657009

ACM AsiaCCS '24, July 01–05, 2024, Singapore

ACM ISBN 979-8-4007-0482-6/24/07
https://doi.org/10.1145/3634737.3657009

1 INTRODUCTION

Estonian e-voting has been enjoying a high voter turnout [2], calling for scrutinous security evaluation. Indeed, an early paper [36] has shown that ballot manipulation attacks are possible by a corrupt voting device. The protocol was later improved [38] to allow individual verifiability, aiming to ensure that, if voters verified their vote, their vote will be correctly counted, even if some of the voting infrastructure is corrupt. Universal verifiability complements the voter checks with procedures performed by external auditors. The improved protocol was used in the local elections of 2013. A technical report [50] subsequently pointed out implementation vulnerabilities and attacks against the individual verifiability mechanisms. The protocol was then further improved for the elections of 2015 [37]. However, a recent attack [46] on individual verifiability shows remaining lacks in the security of the protocol.

A second crucial property of electronic voting is vote privacy. To prevent voter coercion, an even stronger property of receipt-freeness is desirable [33, 42]. This allows voters to cast their private vote even if they are under pressure to vote in presence of the adversary, or to reveal their credentials and any confirmations they have received after voting. To achieve this stronger notion of privacy together with individual verifiability, it is in general necessary to allow revoting - otherwise the adversary can verify the cast ballot to ensure its goals are achieved. Revoting is not sufficient, since there may be ways for the adversary to detect that the voter has cast a new vote against the its instructions. Therefore, the Estonian E-voting protocol (EEV) supports several additional measures that aim to support receipt-freeness: e.g. the bulletin board is not public, a ballot may be verified only within a certain timeframe, voters are allowed to verify any of their cast ballots.

The complex relation between verifiability, receipt-freeness and the corruption abilities of the adversary calls for rigorous formal models and automated verification for EEV. Indeed, in spite of improvements over the years, recent attacks were shown against EEV, both on individual verifiability [46], and on privacy [45]. The attack against individual verifiability in [46] exploits the fact that voters can verify any of their cast ballots, to resist coercion. The privacy attack in [45] exploits cryptographic weaknesses that they propose to fix with zero-knowledge proofs, yet we have found with formal analysis a protocol level attack, in the style of ballot copy attacks from [31]. Even if one removes duplicate ballots, by exploiting revoting the adversary can lead the voter to create two distinct ballots for the same candidate, leading to a privacy violation.

Contributions and related work. We provide formal specifications and automated verification for the most recent version of EEV [3, 37]. We systematically consider all possible corruption scenarios, allowing the adversary to control various parties and infrastructure: registration service, vote collector, communication network, voting device, voters, etc. For the resulting models, we verify end-to-end election verifiability with Tamarin [43], and privacy and receipt-freeness with ProVerif [15]. We find several new attacks and propose practical solutions to provably improve the protocol. We also propose a foundational symbolic framework that allows for the first time to prove vote privacy for an unbounded number of voters in presence of malicious parties or infrastructure.

Formal verification in a symbolic model, also called the Dolev-Yao model, has become essential for ensuring protocol security or finding attacks [28], in particular for e-voting, e.g. [10, 23, 31, 33]. It requires symbolic specifications of the protocol, the adversary and security definitions. Currently, one can only find an informal descriptions or the implementation of EEV [3, 37]; we are the first to provide a detailed formal specification, suitable for automated tools. Our specification also includes some additional timing checks that we have not found in [3, 37], but which we think are necessary to ensure that ballots cannot be reordered when the storage backend is corrupt. We introduce some techniques that may be of independent interest and of more general applicability, allowing for example to express time ordering constraints. We also provide general adversarial models to capture all potential attacks enabled by corrupting each of the protocol parties. We rely on existing definitions to model election verifiability [10, 12]. For privacy and receipt-freeness, we introduce new definitions extending the scope of existing symbolic definitions [33, 34]. As we discuss in Section 5.3, our definition allows to specify more scenarios than the classic definition [33], and allows for a more general way of handling corrupt parties and infrastructure than the recent definition in [34].

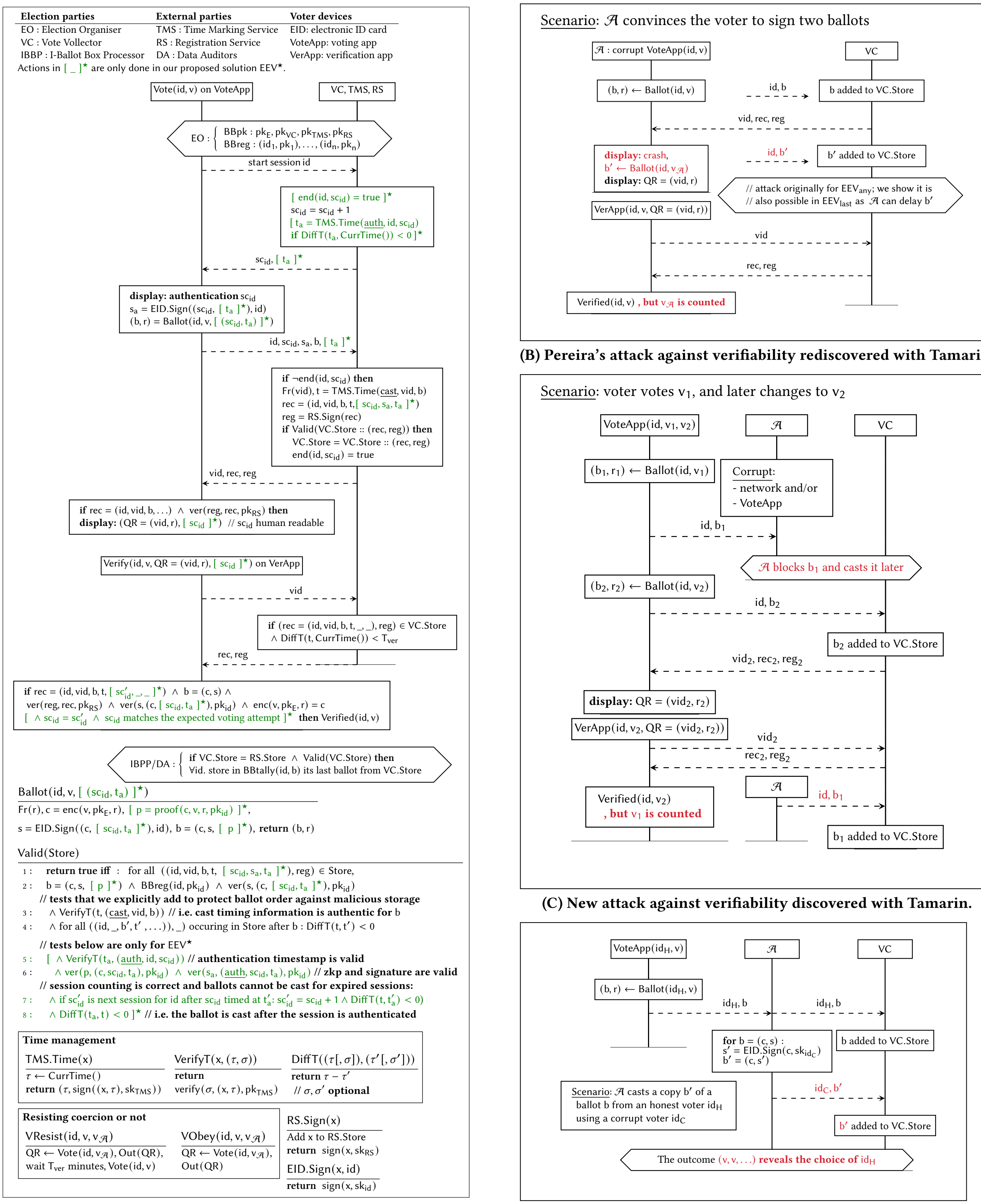
We discover new attacks on verifiability (with Tamarin) and privacy (with ProVerif). These attacks are possible when any party involved in ballot casting is corrupt (i.e., the network, the voting application, or the vote collector). The attack on privacy in addition requires the adversary $\mathcal{A}$ to corrupt voters, and is similar to ballot copy attacks from [31] against Helios, where the ballot of an honest voter is copied and cast in the name of a corrupt voter. Against verifiability, our attacks are similar to the ballot reordering attacks from [10] against Belenios. There are several scenarios for the ballot reordering attack. For example, if a voter votes for a candidate v_1 and later revotes for v_2 (e.g., to resist coercion) and verifies v_2, $\mathcal{A}$ has two ways of reordering the corresponding ballots b_1 and b_2 and make v_1 count while avoiding detection by the voter: i) it can delay b_1 so that it is cast after the voter verified v_2; or ii) it can reorder the ballots before verification, and exploit the fact that voters can verify b_2 even if b_1 is the last ballot cast in their name - which is a feature introduced to offer better receipt-freeness. We also rediscover Pereira's attack [46] against individual verifiability: $\mathcal{A}$ induces the voter to sign two ballots, one of which will contain the vote desired by $\mathcal{A}$, and one for which the voter will perform successful verification. We show that this type of attack is possible in more scenarios than initially described in [46].

Several possible solutions against Pereira's attack are discussed in [46]. We discuss in Section 4.1 why these solutions are not sufficient, and show what additions are needed in order to obtain provable security. Furthermore, we argue that these solutions are not yet readily deployable and affect usability. Therefore, in Section 4.2 we propose a solution based on a better accounting of the time when sessions are started and when corresponding ballots are cast. Concerning privacy, we strengthen the recent improvement proposed in [46] in order to counter our ballot copy attack. Indeed, their attack exploits the malleability of the ballot, and they propose to fix this with a zero-knowledge proof that prevents it. To also prevent ballot copy, we extend this proof with a label that verifiably links the ballot to the corresponding voter, similarly to Belenios [27].

We perform automated verification of verifiability, privacy and receipt-freeness in 6 protocol variants: two variants for the original EEV (allowing individual verification for any ballot, or only for the last ballot cast), and the two same variants for each of the two improved versions described in Section 4. For each verification task, we consider 9 corruption scenarios, showing that the improved versions satisfy the original goals of EEV, and also where there is scope for improvement. For example, none of the considered variants can provide verifiability or receipt-freeness when both the vote collector and the registration service are corrupt. Our specifications and results are presented in Section 5, and the full files are in supplementary material [1].

2 OVERVIEW OF THE EEV PROTOCOL

There have been several iterations of the EEV protocol [36–38]. We present an overview of the most recent version, IVXV [3, 37]. See Figure 1 for a more detailed description of some operations and [1] for our full formal specification. A public key infrastructure provides protocol parties with certified signature key pairs; all messages are signed by senders and verified by receivers. Secret keys of eligible voters (used for authentication and signing) are stored within their personal electronic identity card (EID). As listed at the top of Figure 1, three election parties are responsible for organising the election and computing the final result, and three external parties perform ballot time-stamping, ballot registration, and election data audit. The Election Organiser EO determines the lists of candidates and eligible voters, each holding an EID card with certified signature key pair $(\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$. EO also generates the election key pair $(\mathsf{pk}_{\mathsf{E}}, \mathsf{sk}_{\mathsf{E}})$, makes pk_{E} available to any party in the protocol, and decrypts the final ballots to be tallied. The Vote Collector VC collects ballots from voters via their voting applications during the voting phase. It interacts with the Time Marking Service TMS and the Registration Service RS to acquire a timestamp and registration confirmation for each ballot. The I-Ballot Box Processor IBBP prepares the tally procedure by verifying that all the ballots in the VC ballot box are valid and consistent with the registration information recorded by RS and selects the last ballot for each voter for tally. The Data Auditors DA audit all parties by verifying the list of ballots and the eligibility of corresponding voters, verifying the registration confirmations from RS, checking consistency between VC and RS as done by IBBP, and verifying the proofs of tally correctness.



(A) Voting and individual verification in EEV and EEV*.

(B) Pereira's attack against verifiability rediscovered with Tamarin.

(C) New attack against verifiability discovered with Tamarin.

(D) New attack against privacy discovered with ProVerif.

Figure 1: Procedures and attacks in the Estonian E-Voting Protocol and its variants.

Setup and voting phases. The election public key $\mathsf{pk_E}$ and the list of eligible voters are published by EO. Each voter id holds an EID which has the tuple $(\mathsf{id}, \mathsf{pk_{id}}, \mathsf{sk_{id}})$ inside, where $(\mathsf{id}, \mathsf{pk_{id}})$ is certified. Similarly, the services TMS, RS and VC hold certificates for their public keys. For simplicity of presentation, we assume a bulletin board BB containing all the information that should be available to all parties in the election. Voters use their VoteApp and EID card to cast a ballot $\mathsf{b} = (\mathsf{c}, \mathsf{s})$, where $\mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk_E}, \mathsf{r})$ is the randomised encryption of their choice v and $\mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk_{id}})$ is their signature. The EID card is first used to authenticate the voter to VC and then for signing the ciphertext containing the vote. The VoteApp sends $(\mathsf{id}, \mathsf{b})$ to VC, which verifies the eligibility of the voter and the validity of the certificate acquiring a timestamp on the ballot from TMS, creates a fresh identifier vid for b, and registers $(\mathsf{vid}, \mathsf{b})$ with RS, who replies with a signature reg of $(\mathsf{vid}, \mathsf{b})$. If all checks and registration are successful, VC stores the record in its database and sends $(\mathsf{vid}, \mathsf{reg})$ back to the VoteApp, confirming the receipt of the ballot by VC and its registration by RS. The VoteApp constructs a QR code representing $(\mathsf{vid}, \mathsf{r})$ that can be used for verifying the ballot.

Tally phase and individual verification. IBBP collects ballots from VC, verifying their validity and consistency with ballots registered by RS. If no problem is detected, IBBP retrieves the last ciphertext recorded for each voter. The list of resulting ciphertexts is anonymised by homomorphic combination (or using a re-encryption mixnet), and the resulting combined ciphertext (or list of ciphertexts) is sent to EO for decryption. EO uses the election secret key $\mathsf{sk_E}$ to decrypt the ciphertext(s), publishing the outcome and a proof of correct decryption to be checked by DA. During the voting phase, voters can verify that their ballots reached the VC and encoded their desired votes. For this, they enter the $\mathsf{QR} = (\mathsf{vid}, \mathsf{r})$ code they received into the VerApp, which sends vid to VC to request the ballot recorded for that vid. The VC retrieves the tuple $(\mathsf{id}, \mathsf{b} = (\mathsf{c}, \mathsf{s}), \mathsf{reg})$ corresponding to vid from its database and sends the tuple back. The VerApp first verifies s and reg. Second, using the randomness r from the QR code it can determine whether the ciphertext c recorded by VC encodes a valid vote v', by simply recomputing the encryption algorithm for eligible candidates. In this case, v' is displayed to the voter, who concludes successful verification if v' matches the expected vote v. We note that there are two possible instances of EEV based on the type of individual verification allowed: in one variant, that we call $\mathsf{EEV_{last}}$, only the last recorded ballot can be verified; in the second variant, $\mathsf{EEV_{any}}$, any of the recorded ballots can be verified. We explain the reason for this distinction below.

Measures for receipt-freeness. Voters in the EEV system are particularly vulnerable to coercion due to the verification QR codes that could serve as receipt for the coercer. To counter this, EEV allows individual verification only within a timeframe $\mathsf{T_{ver}}$ specified by the protocol, usually 30 minutes. This feature means that the voter can pretend to follow the coercer instruction and provide them with the corresponding QR code, but after $\mathsf{T_{ver}}$ expires, the QR code becomes useless to learn which vote was cast, and the voter can undetectably revote. Still, there may be a problem if the voter does not have the possibility to wait for $\mathsf{T_{ver}}$ minutes, and has to vote soon after it was coerced. For that situation, the system $\mathsf{EEV_{any}}$ provides a better protection than $\mathsf{EEV_{last}}$: the vote cast by the voter will be counted (as it is the last one cast), while the QR code provided to the coercer will still be valid, since any ballot can be verified.

Timing information and checks. During ballot casting, the VC executes the OCSP [48] protocol with the TMS to attest that the public key certificate of the voter is still valid. In [3, 37], we can see that the OCSP protocol can also be used to get a timestamp t on the ballot. In this case, the TMS will play the role of RS. In other cases, the VC executes the TSP [53] protocol with the RS to register the ballot in addition to the OCSP [48] protocol with the TMS. For clarity of presentation, we assume all the timing information is explicitly recorded with TMS in Figure 1, bearing in mind that RS may also assume some timekeeping duties to distribute trust. Looking at the informal specification in [37] and the implementation at [3], we have not seen the ballot casting timestamps explicitly verified in the auditing checks performed by DA. Furthermore, we expect the DA should also check that the order of the stored ballots is consistent with the order of their casting time, to ensure that ballots have not been reordered in the final store. We add such checks to the Valid(Store) procedure executed by DA. In one of our solutions, we will add an additional timing information $\mathsf{t_a}$ related to the authentication time of the session, and additional timing checks related to both t and $\mathsf{t_a}$. As part of the OCSP/TSP protocol, both the VC and the TMS will check that the recorded time correctly reflects the current time. Therefore, if at least one of these parties is honest, we can trust that the timing information is correct.

3 THREAT MODELS AND ATTACKS

We present the considered threat models and the attacks that we found by automated verification as described in Section 5. In the next section we present our formally proved solutions to these attacks. See Figure 1 for an illustration of attacks and Table 1 for verification results in all cases. Each threat model (nine in total) describes a corruption scenario concerning parties and infrastructure: the adversary $\mathcal{A}$ may control voters, their voting applications, the communication network and some of the election parties. In each scenario, the verification application and the data auditors DA are assumed honest. Since all IBPP actions are audited by DA, it is subsumed by DA in our models. In each scenario, $\mathcal{A}$ has a subset of the following corruption abilities:

- *Corrupt voters:* leak all their voting credentials, ballot casting and signing abilities to $\mathcal{A}$.
- *Corrupt network:* $\mathcal{A}$ may remove, insert, and reorder messages from public channels.
- *Corrupt* VoteApp*:* $\mathcal{A}$ can use the EID card to sign any chosen message; it can see and replace voter's inputs, and supplies corresponding replies.
- *Corrupt* RS*:* $\mathcal{A}$ gets the secret key $\mathsf{sk_{RS}}$ and is allowed to answer any queries meant for RS.
- *Corrupt* VC*:* leaks all its data to $\mathcal{A}$ and lets it cast any ballots.
- *Corrupt* TMS*:* helps $\mathcal{A}$ in manipulating the recorded time.

Verifiability should hold for EEV as soon as one of the VC or RS is honest. The attacks we discover target individual verifiability, which should ensure that for every voter that verified their vote successfully, that vote should be correctly counted in the final tally.

Table 1: Corruption Scenarios and Verification Results in EEV and Its Variants

Corrupt parties	Verifiability		Privacy		RF	
	EEV	EEV^{+}	EEV	EEV^{+}	EEV	EEV^{+}
$\mathcal{A}_1$: voters	✓	✓	✓	✓	✓	✓
$\mathcal{A}_2$: voters, network	✗	✓	✗	✓	✗	✓
$\mathcal{A}_3$: voters, VoteApp	✗	✓*	✗	✗	✗	✗
$\mathcal{A}_4$: voters, network, VoteApp	✗	✓*	✗	✗	✗	✗
$\mathcal{A}_5$: $\mathcal{A}_4$ + RS + TMS	✗	✓*	✗	✗	✗	✗
$\mathcal{A}_6$: $\mathcal{A}_4$ + VC	✗	✓*	✗	✗	✗	✗
$\mathcal{A}_7$: $\mathcal{A}_2$ + RS + TMS	✗	✓	✗	✓	✗	✓
$\mathcal{A}_8$: $\mathcal{A}_2$ + VC	✗	✓	✗	✓	✗	✗
$\mathcal{A}_9$: $\mathcal{A}_2$ + VC + RS + TMS	✗	✓	✗	✓	✗	✗

where EEV^{+} is EEV* or EEV$^{\mathsf{ntfy}}$; ✓* represents weak result integrity: $\mathcal{A}$ can stuff ballots if voters don't verify their votes.

Known attacks. We rediscover Pereira's attack [46] shown in Figure 1.(B). This attack assumes a corrupt VoteApp, and was originally shown for the variant $\mathsf{EEV_{any}}$ of the system. The voter starts a session with the VC and submits a ballot b_1 for the desired vote v_1. Instead of displaying the QR code that the VC sent as confirmation for b_1, the VoteApp pretends the connection was cut and asks for a renewed voting session. In that session, the VoteApp encrypts the adversary's choice v_2 and submits the corresponding b_2. Finally, VoteApp displays the first ballot's QR code for verification and all the verification checks pass, so the voter expects v_1 to be tallied. However, the last ballot cast, that is b_2 corresponding to v_2, is tallied for the respective id. While the original attack as described in [46] works only for $\mathsf{EEV_{any}}$, our results with Tamarin show that individual verifiability is violated also for $\mathsf{EEV_{last}}$. Indeed, $\mathcal{A}$ can delay the delivery of the adversarial ballot b_2 until after the voter verifies their ballot b_1. Furthermore, we find that the scenario described by Pereira, where there is a dubious interaction with the VoteApp, is not the only one where this attack is possible. The attack also works if the voter simply decides to revote for the same candidate, as a result of coercion or lost verification code. In that case, $\mathcal{A}$ could simply feed to the voter the previous verification code and cast the vote that it desires exploiting the second voting attempt.

New attack on verifiability, Figure 1(C). We discover a ballot reordering attack when the network (or VoteApp, or VC) is corrupt, which applies to both $\mathsf{EEV_{any}}$ and $\mathsf{EEV_{last}}$. The attack applies in the scenario when the voter revotes and changes the desired vote, e.g. from v_1 to v_2. Note that this is the prescribed behaviour in EEV when the voter is coerced, so it is a situation that is anticipated by design in EEV. Nevertheless, we find that in this case the adversary can reorder the ballots and cast the vote for v_1 - even if the voter verifies the ballot for v_2 and expects it to be cast. The attack is depicted in Figure 1: the voter id casts two successive ballots b_1 and b_2, corresponding to v_1 and v_2; $\mathcal{A}$ controls the network, blocks the first ballot b_1, and submits the second ballot b_2 directly. The VC processes b_2 and sends its confirmation to the voter. At some later point, $\mathcal{A}$ silently submits b_1. Finally, the voter verifies v_2 and expects it to be tallied, whereas the ballot b_1 corresponding to v_1 is tallied instead, which violates individual verifiability. This attack is similar to the one in [10] found for Belenios.

New attack on privacy, Figure 1(D). We discover a ballot copy attack against privacy, which belongs to a class first described in [31] against Helios. In its simplest version, the attack consist in copying the ballot cast by one honest voter and recasting it in the name of a dishonest voter. This creates a bias in the outcome that allows the adversary to infer the vote of the honest voter. For example, imagine a simple scenario with three voters, two honest and one corrupt. If the two honest voters vote differently, the adversary should not be able to tell how each of them voted. However, if the adversary manages to copy and make the corrupt voter cast the same vote as one of the honest voters, then the winner will reveal the choice of that voter. As shown in [44], this type of attack leads to quantifiable privacy violations in more general scenarios.

Practical impact. Pereira's attack clearly breaks verifiability under the current trust assumptions of EEV, where the voting application is assumed to be malicious. Malware on the voting platform is a real threat, as noticed in the analyses of earlier variants of EEV [50], and the point of individual verification is, among others, to protect against this threat. Concerning the ballot reordering and the ballot copy attacks, one may note that they should not be possible if a secure channel is implemented between the voting application and the vote collector. We note, however, that ballot reordering is possible even if the network is assumed honest: it is sufficient for either the voting application or the vote collector to be malicious. A minimum standard for verifiability is that it should not fail if any single party is corrupt or fails, e.g. due to an implementation error. Ballot reordering is also stealthier than Pereira's attack, since there is no noticeable change in the system behaviour. Similarly, while the voting application has to be trusted in EEV for privacy, the vote collector or the network should not be trusted, yet $\mathcal{A}$ can mount a ballot copy attack in EEV by corrupting them. We will aim to obtain verifiability even if the voting application, the network and one of the VC or RS are corrupt.

Attacks currently outside the scope of our model. The symbolic model that we consider abstracts away some cryptographic details like the type of the encryption scheme being used. In particular, we treat a ciphertext as a black box that cannot be modified by $\mathcal{A}$, except through the equations we allow. Two recent attacks, one against verifiability [51] and one against privacy [45], have been found by exploiting additional properties of ElGamal ciphertexts used in the implementation of EEV. The attack in [51] allows the voting application to discard or change the vote in a ballot, even if the voter verified it. It uses the fact the implementation of the VC does not send back to the verification application the complete ElGamal ciphertext of the vote $\mathsf{enc}(v, pk, r) = (g^{r}, pk^{r} \cdot v)$, but only $pk^{r} \cdot v$. This allows the malicious voting application to choose a different randomness r' when displaying the QR code. When combined with $pk^{r} \cdot v$, this results in the encryption of another vote $(g^{r'}, pk^{r} \cdot v) = (g^{r'}, pk^{r'} \cdot v') = \mathsf{enc}(v', pk, r')$, for some v'. The attack on privacy described in [45] uses the fact that the ElGamal encryption scheme is malleable. Specifically, it uses its homomorphic properties to modify the vote inside the ballot of a voter and make it equal to a value that, when published in the outcome, will be an outlier that will give a hint about the original value of the vote, or will leak the votes of other voters. These two attacks can be brought within the scope of symbolic models by extending the equational theory to capture the relevant algebraic properties of the encryption scheme. However, automation for such homomorphic or re-randomisation properties of the encryption scheme is currently outside the scope of Tamarin and ProVerif. The problem may be simpler to automate for a bounded number of sessions, but current tools targeted for this case, like e.g. DeepSec [20], still cannot handle it.

4 SOLUTIONS

We first consider the possible solutions that have been proposed in [46] (for verifiability) and [45] (for privacy) to counter the attacks described in these papers. We show why these proposals are not sufficient and propose improvements to obtain the desired properties. This comes with significant changes to the voting infrastructure and reduced usability. We therefore develop a new solution that relies on existing infrastructure with minimal impact on the voting and verification experience.

4.1 Improving existing solutions within $\mathsf{EEV}^{\mathsf{ntfy}}$

4.1.1 Achieving verifiability. Four approaches are proposed in [46] to mitigate Pereira's attack. We can discard two of them, since they would violate receipt-freeness in EEV: providing a public bulletin board (the coercer can see a disobeying revote) or accepting at most one ballot per voter (one cannot revote to resist coercion). Another approach suggested in [46] is to use $\mathsf{EEV}_{\mathsf{last}}$, since they observed their attack only against $\mathsf{EEV}_{\mathsf{any}}$. However, we have seen in Section 3 that the attack still applies in this case. We are left with the final variant suggested in [46]: the use of an additional feedback channel (e.g. a mobile phone) whereby the voter is notified each time a ballot is cast in their name. This notification should contain the identifier vid_c of the ballot cast, and the voter should confirm that it matches the vid_c they received from the VoteApp. Intuitively, this should prevent Pereira's attack because the malicious VoteApp cannot hide from the voter that a ballot is cast in their name.

A first challenge for this solution is choosing the party that notifies the voter. If we assign this role to VC, then an adversary corrupting it can still mount the attacks, especially if it also corrupts the VoteApp or the network channel. A second challenge is usability. Indeed, to counter the ballot reordering attacks, not only do the voters need to perform the additional test $\mathsf{vid}_\mathsf{c} = \mathsf{vid}_\mathsf{p}$, but they also need to detect any notification received after they performed verification, since the adversary may cast the desired ballot afterwards. Finally, there is the question of deployment and of new risks associated with the feedback channel, as discussed in [35]. Our contribution in this context is to show what are the minimally required additions in order to make the notification-based extension of EEV provably secure, even if all parties except the DA and one of VC and RS are corrupt.

Our main additions are a bulletin board shared between VC and RS and additional voter checks. The VC and RS update the bulletin board whenever a new ballot is cast, and a trusted party TP monitors the bulletin board and sends a notification to the respective voter when a ballot is cast. The voters need to ensure they receive no further notification after they verified their desired ballot. Formally, for the given vid from the VoteApp, they should ensure that: **(a)** a notification has been received on the out-of-band channel matching vid; **(b)** no further notification is received, unless the voter has revoted and expects the first ballot to be overwritten.

4.1.2 Achieving privacy and receipt-freeness. The privacy attacks from [45] are based on corrupting the ballot of an honest voter, such that the vote encoded inside can reveal information about the initial vote cast. The solution proposed in [45] to mitigate this attack is to add a zero-knowledge proof that ensures the ballot cannot be modified without detection, preventing the vote v inside the ciphertext constructed by an honest voter from being modified. However, this is not sufficient to prevent the ballot copy attack against privacy, since the adversary can take a ballot, extract the ciphertext and the zero-knowledge proof, and reuse them without modification to cast a vote on behalf of a malicious voter. To prevent this from happening, we extend the zero-knowledge proof proposed in [45] to ensure that no ciphertext from a ballot cast by an honest voter can be cast by another voter. Specifically, we rely on labeled encryption, as used in Belenios [27] to prevent ballot copy attacks like in [31]. A *labeled encryption scheme* is one that has two additional algorithms, one for labelling a ciphertext and one for verifying the label. The labelling algorithm zkp allows the creator of a ciphertext $\mathsf{c} = \mathsf{enc}(\mathsf{x}, \mathsf{pk}, \mathsf{r})$, who knows the randomness r, to attach a label ℓ to c obtaining a proof p such that, when the label verification algorithm ver is applied with arguments $\mathsf{p}, \mathsf{c}, \mathsf{pk}, \ell'$, it will return true iff $\ell = \ell'$. This can be modelled by the equation (3) from Figure 2. In Belenios, and our proposed enhancement to EEV, the label attached to ciphertext is the public key of the voter that constructs the ballot. The ballot is now $\mathsf{b} = (\mathsf{c}, \mathsf{s}, \mathsf{p})$, where $\mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}_\mathsf{E}, \mathsf{r})$, $\mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{sk}_{\mathsf{id}})$, and $\mathsf{p} = \mathsf{zkp}(\mathsf{enc}(\mathsf{v}, \mathsf{pk}_\mathsf{E}, \mathsf{r}), \mathsf{v}, \mathsf{r}, \mathsf{pk}_{\mathsf{id}}))$. The labelling proof p is verified through the test $\mathsf{ver}(\mathsf{p}, \mathsf{c}, \mathsf{pk}_\mathsf{E}, \mathsf{pk}_{\mathsf{id}}) = \mathsf{true}$, which must be performed by the VC and the DA.

We denote by $\mathsf{EEV}^{\mathsf{ntfy}}$ this version of the protocol. We prove that verifiability holds for $\mathsf{EEV}^{\mathsf{ntfy}}$ if (VC or RS) and TP are trusted. Apart from the organisational challenges (who should play the role of TP?) and the usability challenges involved in this solution, designing and implementing a secure and distributed bulletin board for voting with minimal trust assumptions is a long-standing and still actively researched problem [39, 52], thus we think this solution is not easily deployable. On the other hand, extending the zero-knowledge proof with a label to protect privacy does not present any deployment challenge. It is already implemented, for example, in Belenios.

4.2 New solution $\mathsf{EEV}^{\star}$

We notice that both in Pereira's and in the ballot reordering attacks, the adversary is helped by the power to delay the delivery of a ballot, and sometimes cast it after another session started. In Pereira's attack, it also helps that $\mathcal{A}$ can start a second voting session while the voter thinks it is still participating in the first session. To prevent attacks based on such vulnerabilities, we propose a solution that allows to ensure that each ballot is cast within the session where it was created, and that the voter cannot confuse the session for which it verifies the ballot and the session for which the ballot is cast. Recall that ballot casting in EEV requires to first start a session for a given voter id, after which the VoteApp creates a ballot and asks the voter to sign it. Our additions for $\mathsf{EEV}^\star$ are shown in Figure 1, denoted by $[\ \_\ ]^\star$. The main idea is to add verifiable timing information when the user starts each new session. We link this timing information with a corresponding session counter. This session counter will help voters for individual verifiability, enabling them to verify ballots for the correct voting session, without having to track the time. In more detail, at every voting session for a given id, the VC increases a corresponding session counter $\mathsf{sc_{id}}$ and registers with the TMS that a new authentication session is created for id and $\mathsf{sc_{id}}$, obtaining a timestamp $\mathsf{t_a}$. It sends $\mathsf{sc_{id}}$ and $\mathsf{t_a}$ to the VoteApp for display to the voter. Then VoteApp includes $\mathsf{t_a}$ in the signature $\mathsf{s_a}$ that authenticates the voter. When the voter signs the ballot, the VoteApp puts $\mathsf{sc_{id}}$ and $\mathsf{t_a}$ together with the ciphertext c encoding the vote, to get a signature $\mathsf{sign}((\mathsf{c}, \mathsf{sc_{id}}, \mathsf{t_a}), \mathsf{sk_{id}})$. The ballot also contains the zero-knowledge proofs described in Section 4.1, in order to obtain privacy and receipt-freeness. The VC checks that the ballot casting time is greater than the time when the session has started, i.e. the time recorded in t is greater than $\mathsf{t_a}$. The VC ends any old session if the voter starts a new voting session: no ballot from the older session can be received anymore, and there is at most one ballot per session.

Universal verification is done by DA in procedure Valid(Store). It ensures that timestamp, signature and zkp in b are valid (*line 5, 6*); no ballot is cast for an older session if a new one started *(line 7)*; the session started before the ballot is cast *(line 8)*. *Individual verification*: for a given vid, the VerApp displays to the voter the session counter $\mathsf{sc'_{id}}$ stored by VC for vid; the voter checks that the displayed session counter $\mathsf{sc'_{id}}$ matches their number of voting attempts $\mathsf{sc_{id}}$, obtained through their interaction with VoteApp. The session counter $\mathsf{sc_{id}}$ is added to the receipt of the ballot in addition to the QR code, in order to help voters track their number of voting attempts. This number is displayed at ballot casting time and, if the VoteApp attempts to display an incorrect counter, the assumption is that the voter will spot this, either while casting the vote, or later when verifying it on the VerApp. Note that $\mathsf{sc_{id}}$ is the number of successfully established voting sessions; the number of cast ballots for that voter may be smaller. We prove formally with Tamarin that $\mathsf{EEV}^\star$ is secure according to threat models and results in Table 1. Verifiability holds even if the VoteApp, the communication network and one of the VC or RS are corrupt. Note that, while the TMS is responsible for the timing of sessions and ballots, the VC will also check that the timing information is correct, as part of the OCSP/TSP [48, 53] protocol, as we explain in Section 2. This means that, if the TMS is corrupt, we can still obtain security when the VC is honest.

How attacks are countered. The general pattern for Pereira's attack is that the voter creates a first ballot $\mathsf{b_1}$ containing the desired vote $\mathsf{v_1}$, that is cast and is assigned a $\mathsf{vid_1}$. Then the voter constructs a second ballot $\mathsf{b_2}$, which $\mathcal{A}$ manipulates to contain $\mathcal{A}$'s desired vote $\mathsf{v_2}$. For verification of this second session, which from the voter's perspective should still represent a vote for $\mathsf{v_1}$, $\mathcal{A}$ presents the voter with $(\mathsf{vid_1}, \mathsf{r_1})$ from the earlier session. In $\mathsf{EEV}^\star$, the verifiable backend will record the session counter for the each session. Let's say that these will be $\mathsf{sc^1_{id}}$ for the first session, and $\mathsf{sc^2_{id}}$ for the second session. Then, when the voter submits $(\mathsf{vid_1}, \mathsf{r_1})$ for individual verification, the VerApp will be able to determine and display the correct session number $\mathsf{sc^1_{id}}$. On the other hand, from the voter's perspective, these numbers should satisfy $\mathsf{sc^2_{id}} > \mathsf{sc^1_{id}}$, so the inconsistency can be detected by the voter.

In a ballot reordering attack, we have the following pattern:

Voter perspective		Adversary $\mathcal{A}$	
$\mathsf{sess_1}$ @$\mathsf{t_1}$,	cast $\mathsf{b_1}$ @$\mathsf{t_2}$	// **corrupts** VoteApp**,** VC **or network**	
$\mathsf{sess_2}$ @$\mathsf{t_3}$,	cast $\mathsf{b_2}$ @$\mathsf{t_4}$	$\mathsf{cast}(\mathsf{b_2})$ @$\mathsf{t_5}$,	$\mathsf{cast}(\mathsf{b_1})$ @$\mathsf{t_6}$

Time ordering: $\mathsf{t_1} < \mathsf{t_2} < \mathsf{t_3} < \mathsf{t_4} < \mathsf{t_5} < \mathsf{t_6}$

In $\mathsf{EEV}^\star$, once $\mathsf{sess_2}$ started, a ballot cannot be accepted for $\mathsf{sess_1}$. So, for the attack to succeed, the adversary needs the voter to sign both ballots $\mathsf{b_1}, \mathsf{b_2}$ before starting the second session. On the voter side, the voter is instructed to sign at most one ballot per session. Still, the adversary could try to manipulate the session such that to the voter it looks like the two ballots are signed for two different sessions. That is why we add the tuple $(\mathsf{sc_{id}}, \mathsf{t_a})$ in the ballot information signed by the voter. $\mathcal{A}$ has to timestamp the session before the voter signs the ballot, and it cannot timestamp a second session without a second authentication attempt by the voter. This will be checked by DA so $\mathcal{A}$ cannot cheat on it even if it corrupts all of VoteApp, VC and the network, but not the TMS.

We adopt the extended zero-knowledge proof described in Section 4.1 and we formally prove with ProVerif that privacy holds even if the network and (VC or RS) are corrupt. Intuitively, the additional session constraints ensured in $\mathsf{EEV}^\star$ improve privacy, because they simply restrict the set of traces available for analysis by the adversary, and this restriction does not depend on the votes inside the ballot. Concerning receipt-freeness, we can still achieve it through revoting by using a trusted environment, since there is no constraint that prevents voters from authenticating and casting a new ballot after they were coerced.

5 SPECIFICATION AND VERIFICATION

We use Tamarin [7, 43, 49] for verifying election verifiability (modelled as a trace property) and ProVerif [8, 15] for verifying privacy-type properties (modelled as equivalence). These are both state-of-the-art tools for automated protocol verification, and for our models Tamarin is more expressive for trace properties, while ProVerif works better for equivalence. To simplify presentation, we adopt a generic specification language that borrows constructs from both Tamarin and ProVerif. The semantics of these tools is based on a notion of *execution traces with events* and, as shown in [19], a more general and practical syntax can incorporate the two. Our syntax can be seen as a simplified version of SAPIC [19] and part of our future work is a unified specification code from which all the proofs can be derived, e.g. with SAPIC. We refer to the supplementary material [1] for details of our specifications in Tamarin and ProVerif, and to [19] for a presentation of distinctive features of these tools.

5.1 Specification language

Messages (also called terms) are built from a set of variables x, y, z, ..., constants and function symbols, endowed with a set of equations for modelling the cryptographic primitives. A tuple $(\mathsf{t_1}, \ldots, \mathsf{t_n})$ of

(1)	$\mathsf{dec}(\mathsf{enc}(x, \mathsf{pk}(y), z), y)$	=	x
(2)	$\mathsf{verify}(\mathsf{sign}(x, y), x, \mathsf{pk}(y))$	=	true
(3)	$\mathsf{ver}(\mathsf{zkp}(\mathsf{enc}(x, y, z), x, z, \ell), \mathsf{enc}(x, y, z), y, \ell)$	=	true

$\mathsf{In}(t), P \quad \mathsf{Out}(t), P$	// **input and output on public channel**
$\mathsf{Fr}(x), P \quad x = t, P$	// x **can be fresh or assigned a term** t
evstore $T(t_1, \ldots, t_n), P$	// **event declaration and/or table storage**
get $T(x_1, \ldots, x_n), P$	// **reading from table**
if Φ **then** P **else** P'	// Φ **is a formula without implications**
$P \mid P' \quad \Phi \mid \Phi' \quad P \mid \Phi$	// **process = actions restricted by formulas**
$!P$	// **unbounded number of instances**
// **formula atoms:**	
$T(t_1, \ldots, t_n)[\ @i]$	// **event occurred [at timepoint** i**]**
$t = t' \quad i = i' \quad i < i'$	// **term equality and timepoint ordering**
// **trace formulas, with formula atoms as base cases:**	
$\Phi \wedge \Phi', \Phi \vee \Phi', \neg\Phi, \Phi \Rightarrow \Phi'$	

Example process

$$\begin{aligned}
\mathsf{Dec}(x) &:= \mathsf{Key}(k), y = \mathsf{dec}(x, k).\\
\mathsf{PStore} &:= \mathsf{In}(x); y \leftarrow \mathsf{Dec}(x); \mathsf{evstore}\ S(y)\\
&\quad \mid \big(S(f(x))\ @i \Rightarrow S(x)\ @j\ \wedge\ j < i\big).\\
\mathsf{POut}(x, k) &:= \textbf{if}\ \neg S(x) \wedge x \neq k\ \textbf{then}\ \mathsf{Out}(\mathsf{enc}(x, \mathsf{pk}(k)))\\
\mathsf{System} &:= !\mathsf{PStore} \mid\ !\mathsf{In}(x), \mathsf{Key}(k), \mathsf{POut}(x, k).
\end{aligned}$$

Figure 2: Equational theory, processes and formulas.

terms is also a term. For EEV protocol, we use public key encryption, signature schemes and zero-knowledge proofs, modelled by the equations from Figure 2, where the equation for the zero-knowledge proof ensures that verification will succeed only if the label ℓ is the one originally associated with the ciphertext, as explained in Section 4.1. We consider an additional set of symbols to represent event names for defining the security property or table names where we can store data during the execution of a protocol. Such symbols are generically represented by the letter T in the following grammar description. Processes P, P' and formulas Φ, Φ' are built according to the grammar from Figure 2. A process is formed of actions that can be put in sequence, in parallel, and also restricted by formulas (that we also denote by a parallel composition). The constructs Fr, In and Out are in the style of Tamarin, while event declaration and table-based operations are in the style of ProVerif. We have conflated the constructs event and store from ProVerif into evstore to avoid repetitions, since often we need to store data in a table, while at the same time declaring a corresponding event. An *execution trace* of the process will record each event declared with evstore along with the corresponding index in the trace. The indices of the trace will be called *timepoints*, as they represent the temporal ordering of events during the execution of the protocol.

The atom $T(t_1, \ldots, t_n)$ is true if there exists an index i in the trace such that an event $T(t'_1, \ldots, t'_n)$ is recorded at timepoint i and we have $(t_1\sigma, \ldots, t_n\sigma) = (t'_1, \ldots, t'_n)$, for some substitution σ that assigns terms to variables. We can explicitly refer to such a timepoint by using the notation $@i$. In general, variables and timepoints that appear at the left of an implication $\Rightarrow$ are implicitly universally quantified, otherwise they are existentially quantified. For example, for some events F, G, constants a, b and variables x, y, the formula $F(a, x) \Rightarrow G(b, y)$ stands for $\forall x, i.\ F(a, x)\ @i \Rightarrow \exists y, j.\ G(b, y)\ @j$. In the latter formula, we can add a timepoint ordering $j < i$ to the right of the implication if we want the corresponding occurrence of $G(b, y)$ to happen strictly earlier than that of $F(a, x)$. We denote by $(y_1, \ldots, y_n) \leftarrow P(x_1, \ldots, x_m)$ the execution of a process P that assumes some instantiated variables $x_1, \ldots, x_m$ and assigns values to some variables $y_1, \ldots, y_m$. Branching execution on condition Φ is encoded in Tamarin with restrictions. The direct use of events/tables in Φ can be encoded in ProVerif with get construction with pattern matching and branching. In the example from Figure 2, $\mathsf{Dec}(x)$ models the decryption of x with a stored key and assigning the result to y; PStore models that we store plaintexts in a table A, under the restriction that any term of the form $f(t)$ should be preceded by t; POut models that we can output encryptions of any term different from the key, if it was not stored.

The adversary $\mathcal{A}$ is allowed to control any inputs and outputs specified by In and Out and apply the function symbols from the equational theory to obtain new messages. As we describe in Section 5.2, to specify the abilities of the adversary resulting from corrupting protocol parties, we can also add new processes, modify honest processes, or omit actions and restrictions. For instance, in the example from Figure 2 we can add a process get $\mathsf{Key}(k); \mathsf{Out}(k)$ to express that the key holder is compromised, or remove the restriction, to specify that the storage is compromised. A *trace property* is a trace formula defined as above. A process P satisfies a property Φ if all its traces satisfy it, denoted by $P \models \Phi$. Examples of trace properties are secrecy, authentication and election verifiability - the adversary should not reach a given state described by Φ without a particular condition Φ' being satisfied, i.e. $\Phi \Rightarrow \Phi'$ should be satisfied. An *equivalence property* specifies that a pair of processes P and Q is indistinguishable for the adversary $\mathcal{A}$, denoted by $P \approx Q$. To define the indistinguishability notion $\approx$, one first models the ability of $\mathcal{A}$ to observe relations between messages it can build by performing equality tests. Then, informally, the property requires that any trace of P can be matched by a trace of Q where $\mathcal{A}$ can perform the same observations. There are various ways of formally defining the equivalence relation, e.g. trace equivalence, bisimulation, etc [9, 21]. We use ProVerif, which relies on one of the strongest notions, named diff-equivalence: the two processes must have the same structure and the matching traces of P and Q should come from the same execution branch [16].

5.2 E-voting protocols and the adversary

We assume that an e-voting specification is given by a process that has the structure $!\mathsf{Vote} \mid !\mathsf{Verify} \mid \mathsf{System} \mid \mathsf{Tally} \mid \mathcal{A}$, where:

- Vote and Verify model the voting and verification actions of voters and their platforms;
- System models the actions of all other election parties and components;
- Tally decrypts the ballots that determine the final outcome;
- $\mathcal{A}$ models any additional power of the adversary that may result from corrupting components or voters.

Sometimes we model the power of $\mathcal{A}$ directly in System, for example getting inputs on the voting platform from a public channel without doing any prescribed verification checks. In addition, to define verifiability along the lines of [10, 12, 26], we will assume the following events:

// **VOTING: generic voting process** Vote**, followed by instantiations of** Ballot **and** Cast **for** EEV **and** $\mathsf{EEV}^{\star}$

$\mathsf{Vote} := \mathsf{In}(id, v),\ w \leftarrow \mathsf{Auth}(id),\ (b, x) \leftarrow \mathsf{Ballot}(id, v, w),\ y \leftarrow \mathsf{Cast}(id, b, w),\ \text{evstore}\ \mathsf{CastB}(id, b),\ \text{evstore}\ \mathsf{Voted}(id, v), \text{evstore}\ \mathsf{Ver}(id, v, x, y).$

// **ballot creation for an honest voting platform; we have** $w = (sk_{id}, pk_{id}, [\ sc_{id}, t_a\ ]^{\star})$ **from** Auth **process**

$\mathsf{Ballot}(id, v, w) := \text{get}\ \mathsf{BBpk}(pk_E),\ \mathsf{Fr}(r),\ c = \mathsf{enc}(v, pk_E, r),\ [\ p = \mathsf{zkp}(c, v, r, pk_{id}),\ ]^{\star},$

$\quad s = \mathsf{sign}((c,\ [\ sc_{id}, t_a\ ]^{\star}), sk_{id}),\ b = (c, s,\ [\ p\ ]^{\star}),\ x = r.$

// **ballot creation for corrupt voting platform: we let $\mathcal{A}$ choose all ballot and session data, but we trust** EID **card for signing**

$\mathsf{Ballot}(id, v, w) := \mathsf{In}(c, r, p),\ s = \mathsf{sign}((c,\ [\ sc_{id}, t_a\ ]^{\star}), sk_{id}),\ b = (c, s,\ [\ p\ ]^{\star}),\ x = r.$

// **ballot casting for corrupt network and** VC**: we output the ballot and let $\mathcal{A}$ control how it is cast and provide the return value** vid

$\mathsf{Cast}(id, b, w) := \mathsf{Out}(b),\ \mathsf{In}(vid), y = (vid, [\ sc_{id}\ ]^{\star}).$

// **authentication events: timing events are justified because one of** TMS **or** VC **is honest; counting events are justified because** DA **is honest**

$[\ \mathsf{AuthVC} := \mathsf{In}(id), \mathsf{In}(sc_{id}), \mathsf{Fr}(t_a), \text{evstore}\ \mathsf{Time}(t_a, (\underline{auth}, id, sc_{id})),\ \text{evstore}\ \mathsf{Count}(((\underline{auth}, id), sc_{id}), \mathsf{Out}(sc_{id}, t_a),\ \mathsf{In}(s_a), \text{evstore}\ \mathsf{Sess}(id, sc_{id}, s_a, t_a)\ ]^{\star}.$

// **ballot registration and storage, complemented by restrictions checking the validity of the ballot; we let $\mathcal{A}$ choose** vid

$\mathsf{StoreVC} := \mathsf{In}(id, b, vid),\ [\ \mathsf{In}(sc_{id}), \text{get}\ \mathsf{Sess}(id, sc_{id}, s_a, t_a), \mathsf{Fr}(t), \text{evstore}\ \mathsf{Time}(t, (\underline{cast}, vid, b))\ ]^{\star}, rec = (id, vid, b,\ [\ sc_{id}, s_a, t_a\ ]^{\star}),$

$\quad \mathsf{In}(rec, reg), \text{evstore}\ \mathsf{Store}(rec, reg).$ // **complemented by restrictions checking that** reg **is a valid signature of** RS **on** rec

// **INDIVIDUAL VERIFICATION: the vote collector checks that the verification time is not expired before returning the record**

$\mathsf{VerifyVC} := \mathsf{In}(vid); \text{get}\ \mathsf{Store}(rec, reg); \text{if}\ rec = (\_, vid, \ldots)\ \wedge\ \neg\mathsf{Expired}(vid)\ \textbf{then}\ \mathsf{Out}(rec, reg)$

// **INDIVIDUAL VERIFICATION: checks performed by the voter and the verification app**

$\mathsf{Verify} := \text{get}\ \mathsf{Ver}(id, v, r, vid,\ [\ sc_{id}\ ]^{\star}), \mathsf{Out}(vid), \mathsf{In}(rec, reg), \text{get}\ \mathsf{BBpk}(pk_{RS}),\ [\ \mathsf{Count}(id, sc_{id})\ ]^{\star},$

$\quad \textbf{if}\ \mathsf{ver}(reg, rec, pk_{RS}) = true\ \wedge\ rec = (id, vid, b, t,\ [\ sc', \_, \_\ ]^{\star})\ \wedge\ b = (c, \ldots)\ \wedge\ c = \mathsf{enc}(v, pk_E, r)\ [\ \wedge\ sc_{id} = sc'\ ]^{\star}\ \textbf{then}\ \text{evstore}\ \mathsf{Verified}(id, v)$

// **UNIVERSAL VERIFICATION: the last ballot is selected for tally and the ballot order is consistent with timestamps**

$\mathsf{Store}((id, vid, b, t, \ldots), \_)\ @i\ \wedge\ \mathsf{Store}((id, vid', b', t', \ldots), \_)\ @i'\ \wedge\ i < i' \Rightarrow \mathsf{Earlier}(t, t')\ \wedge\ (b \neq b' \Rightarrow \neg\mathsf{BBtally}(id, b))$

// **UNIVERSAL VERIFICATION: restrictions checking that signatures, proofs and timestamps are valid**

$\mathsf{Store}(rec, reg)\ @i\ \wedge\ rec = (id, vid, b, t,\ [\ sc_{id}, s_a, t_a\ ]^{\star})\ \wedge\ \mathsf{BBreg}(id, pk_{id})\ \wedge\ \mathsf{BBpk}(pk_E)$

$\quad \Rightarrow b = (c, s,\ [\ p\ ]^{\star})\ \wedge\ \mathsf{ver}(s, (c,\ [\ sc_{id}, t_a\ ]^{\star}), pk_{id}) = true\ [\ \wedge\ \mathsf{ver}(p, c, pk_E, pk_{id}) = true\ \wedge\ \mathsf{ver}(s_a, (sc_{id}, t_a), pk_{id}) = true\ ]^{\star}$

$\quad\quad \wedge\ \mathsf{Time}(t, (\underline{cast}, vid, b))\ [\ \wedge\ \mathsf{Time}(t_a, (\underline{auth}, id, sc_{id}))\ \wedge\ \mathsf{Earlier}(t_a, t)$ // **the ballot is cast after the session is authenticated** $]^{\star}$

// **session counting is correct and ballots cannot be cast for expired sessions**

$\Big[\ \wedge\ \Big(\mathsf{Store}(rec', \_)\ @i'\ \wedge\ rec' = (id, \_, \_, \_, sc'_{id}, \_, t'_a)\ \wedge\ i' > i \Rightarrow \mathsf{Count}((\underline{auth}, id), sc_{id})\ @j\ \wedge\ \mathsf{Count}((\underline{auth}, id), sc'_{id})\ @j'\ \wedge\ j < j'\ \wedge\ \mathsf{Earlier}(t, t'_a)\Big)\Big]^{\star}$

// **because of the test** VC.Store = RS.Store **and the fact that one of** VC **or** RS **is honest, in the specification we can assume one single** Store **table**

// **NATURAL ORDERING, COUNTING AND TIMING CONSTRAINTS**

$\mathsf{Counting} := \Big(!\mathsf{In}(z); \text{evstore}\ \mathsf{Nat}(z)\Big)\ |\ \Big(\mathsf{Count}(x, z_1)\ @i_1\ \wedge\ \mathsf{Count}(x, z_2)\ @i_2\ \wedge\ i_1 < i_2 \Rightarrow \mathsf{Nat}(z_1)\ @j_1\ \wedge\ \mathsf{Nat}(z_2)\ @j_2\ \wedge\ j_1 < j_2\ \wedge\ z_1 \neq z_2\Big)$

$\mathsf{Timing} := \Big(\mathsf{Earlier}(x, x')\ \wedge\ \mathsf{Time}(x, y)\ @i\ \wedge\ \mathsf{Time}(x', y')\ @i' \Rightarrow i < i'\Big)$

$\mathsf{Expire} := \Big(\text{get}\ \mathsf{Store}(\_, vid, \ldots); \text{evstore}\ \mathsf{Expired}(vid)\Big)\ |$ // **ballot verification period expiration is consistent with ballot casting time**

$\quad \Big(\mathsf{Time}(t, (\underline{cast}, vid, b)) \wedge \mathsf{Time}(t', (\underline{cast}, vid', b'))\ \wedge\ \mathsf{Earlier}(t', t)\ \wedge\ \mathsf{Expired}(vid)\ @i \Rightarrow \mathsf{Expired}(vid')\ @i'\ \wedge\ i' < i\Big)$

Figure 3: A selection of processes and restrictions for the specifications of EEV **and** $\mathsf{EEV}^{\star}$ **(in** $[\ \_\ ]^{\star}$**).**

- $\mathsf{Voted}(id, v)$ to record that a voter with that id cast a vote v, typically at the end of the Vote process;
- $\mathsf{Verified}(id, v)$ - a voter with that id verified a vote v, typically at the end of the Verify process;
- $\mathsf{Reg}(id, cr)$ - id is registered with public credential cr; typically cr is equal to the public key of the voter, e.g. in EEV and Belenios;
- $\mathsf{Corr}(id)$ to record that id is corrupt;
- $\mathsf{BBtally}(cr, b)$ to record that b is to be tallied for cr;
- $v = \mathsf{open}(b)$ to represent that v is the vote encoded by b.

For privacy, we will specify two processes that differ in ballots cast by honest voters, and ask for them to be indistinguishable for the adversary. To define the two worlds generically, we assume the Vote process has a part Auth for authrnitcation, a part Ballot that creates the ballot and a part Cast that casts it. We typically have the structure from Figure 3, with small changes depending on the protocol. The terms x, y represent data obtained from procedures Ballot and Cast that can be used for verification. We let $\mathcal{A}$ determine the identity of the voter and its vote, so that $\mathcal{A}$ can setup any scenario it would like to attack. The table Cred stores voter credentials, where w is a tuple representing public and private credentials. For example, in EEV we have $w = (sk_{id}, pk_{id})$ and $x = r$, $y = vid$ form the voter QR code (vid, r). The process Cast for EEV from Figure 3 is simple because we assume a corrupt network, corrupt authentication and a corrupt VC: we output the ballot to the adversary $\mathcal{A}$ and accept any vid as response. $\mathcal{A}$ can choose to treat and cast the ballot in any way it wants, as soon as it passes the verification tests performed by the voter and data auditors. Similarly, when we assume a corrupt VoteApp, we allow $\mathcal{A}$ to choose the ciphertext c in the Ballot process. This allows $\mathcal{A}$ to modify the vote v or to copy the ciphertext from another voter.

Modelling time and counters. Both ProVerif and Tamarin have recently introduced features that allow to model counters [17, 32]. We have attempted to use this feature in some of our models in order to model the counting of voter sessions and the flow of time. However, we have encountered termination problems, especially for checking equivalence in ProVerif and for advanced corruption scenarios in Tamarin, showing that these features need to be further improved in order to be applicable in general. Where counters don't pose problems, we have used them. Otherwise, in most of the cases, we model time and counting using the natural ordering that is provided by the trace execution timepoints in ProVerif or Tamarin. As explained in Section 5.1, every event from the specification of the protocol can occur several times in an execution trace, and we can associate a timepoint for each instance. For example, for an event $\mathsf{E}(x, y)$, we can have:

$$\begin{array}{ccccccc} \mathsf{E}(t_1, u_1) & & \mathsf{E}(t_2, u_2) & \ldots & \mathsf{E}(t_n, u_n) & & \mathsf{E}(t_{n+1}, u_{n+1}) \\ @i_1 & < & @i_2 & < & @i_n & < & @i_{n+1} \end{array}$$

for various terms t_j, u_j substituted for x, y at each step. If we add a restriction to make sure all terms t_i are distinct, we can naturally extend the total ordering on timepoints i_j to a total ordering on the corresponding terms t_j and use t_j as a label to mark that the event E occurred with second argument u_j at the *timepoint* t_j. We show how this idea can be applied to model timestamps and counters.

Timestamp ordering. To timestamp a term u at a given point in the specification, we add the instruction $\mathsf{Fr}(t)$, evstore $\mathsf{Time}(t, u)$, modeling that a time server signed the current time paired with the term u. To verify that a timestamp t has been recorded for u by the time when we perform an action A, we add a restriction $A \Rightarrow \mathsf{Time}(t, u)$. Finally, to ensure that a time t occurs earlier than t', we add an event $\mathsf{Earlier}(t, t')$ with an associated constraint:

$$\mathsf{Earlier}(x, x') \ \wedge \ \mathsf{Time}(x, y) \ @i \ \wedge \ \mathsf{Time}(x', y') \ @i' \Rightarrow i < i'$$

As shown in Figure 3, we use this model to ensure the correct ordering of ballots according to their session and casting time. A more realistic symbolic model of time in security protocols was recently proposed in [13]. They allow for example specifying that a certain cryptographic computations (e.g. the opening of a timed commitment) will take at least a given amount of time, or adding real-time constraints for the execution of protocol rules. However, on the one hand, the support for automation provided in [13] is quite limited: they perform a manual translation of their models into Tamarin to obtain proofs or attacks for some simple examples. On the other hand, we don't need to capture the complex interplay between cryptographic algorithms and their cost in time, as is aimed in [13]. Since our goal is to enforce the ordering of certain actions in time, the abstraction of the real time by ordered execution timepoints is sufficient, and can be expressed directly in Tamarin.

Verification time. To obtain receipt-freeness, we need to restrict the ballot verification functionality to a certain time. After that, the ballot cannot be verified anymore, and the coerced voter can revote for the desired candidate. For this feature, we use again the timepoint ordering: it is sufficient to ensure that cast ballots will eventually expire, in the same order as they were cast. As shown in Figure 3, a respective event Expired can be recorded by an ExpireVid process, and the VC can check expiration upon each individual verification request.

Counter ordering. For modelling counters, we will similarly rely on a series of events $\mathsf{Nat}(u_1)\ @i_1, \ldots, \mathsf{Nat}(u_n)\ @i_n, \ldots$, where $i_1 < \ldots < i_n < \ldots$ and $u_1, \ldots, u_n, \ldots$ as numbers occurring in the natural order. We can refer to them in events $\mathsf{Count}(w, u_j)$, with a respective restriction Counting in Figure 3, modelling the natural counting order. The meaning of $\mathsf{Count}(w, u)$ is that a party in the protocol interprets u as a member of the set of counters defined by the events Nat and w as the occurrence of a specific event it wants to count. For example, for $\mathsf{EEV}^\star$ in Figure 3 we use $\mathsf{Count}(id, sc)$ to represent that the voter with that id counts sc as the current session number. Then, the Counting restriction will ensure that the order of counters on the voter side is the same as the order of counters defined by the Nat events, which is the same as the order ensured by the DA by universal verifiability on the vote collector side, as we add the event $\mathsf{Count}((\underline{auth}, id), sc)$ to which the same restriction applies.

5.3 Definitions for security properties

Election verifiability. We consider the symbolic definition of election verifiability from [12], extending earlier definitions of [10, 26], which is a set of trace formulas ensuring two main properties:

- *Individual verifiability:* if a voter successfully verifies the vote, it will be correctly counted for the final tally.
- *Result integrity:* the tallied vote for each credential should correspond to a vote cast by the corresponding voter, unless that voter is corrupt, i.e. $\mathsf{BBtally}(cr, b) \Rightarrow \mathsf{Reg}(id, cr) \ \wedge (\mathsf{Vote}(id, v) \wedge v = \mathsf{open}(b) \vee \mathsf{Corr}(id))$.

Ballot integrity. Recent works show a connection between privacy and verifiability, e.g. claiming that individual verifiability is needed for privacy [29], or that we should consider various levels of privacy to match various levels of bulletin board corruption [30]. In this paper we take a simpler approach, showing that one single property of ballot integrity is sufficient to define privacy in any corruption model. Intuitively, ballot integrity (defined as Φ^{bi} in Figure 4) is a stronger version of result integrity ensuring that ballots tallied for honest voters have been actually cast by them. In modern systems like e.g. Belenios [5], EEV [3] and the one from SwissPost [6], ballot integrity has emerged as one of the fundamental requirements. It is sometimes called eligibility verifiability [41] and is typically guaranteed by signing the ballots that the voters cast. In some other systems the property does not hold if honest voters don't verify their votes, for example in Helios [4]. Although individual verifiability can help in ensuring ballot integrity (e.g. in Helios), it is just one of the available means.

Privacy definition and related notions. In symbolic models privacy is typically defined only for a simple scenario where two honest voters swap their votes [33]. As shown in [14], this setting cannot capture certain types of voting schemes and scenarios. It also assumes an honest infrastructure that correctly counts the ballots of the two voters. Definitions in the computational model are more flexible and can be extended to handle corrupt infrastructure [30]. The computational definition that has emerged as the most expressive and amenable to mechanised proofs is BPRIV - ballot privacy [14, 25, 30]. It allows $\mathcal{A}$ to setup an experiment whereby it interacts with the protocol in one of the two worlds: the ballots in the left world include the real vote of honest voters, while the ballots in the right world include arbitrary votes chosen by the adversary. The goal is to show that $\mathcal{A}$ cannot distinguish between the two worlds, which implies the privacy of honest votes.

Specification $\mathcal{S}$ = !Vote | !Verify | System | Tally | $\mathcal{A}$

Vote : contains procedures Ballot and Cast to create and cast a ballot; it generates events CastB(id, b).
Verify : models voter verification; generates events Verified(id, v).
System : generates events Reg(id, cr) and BBtally(cr, b).
Tally : for all b s.t. BBtally(cr, b) , does TallyB(b) , where TallyB(b) opens and publishes the vote from b.

Adversary $\mathcal{A}$: corrupts parties and infrastructure.

Voters = !Vote | !Verify

Vote

In(id, v)
w ← Auth(id)
(b, x) ← Ballot(id, v, w)
Cast(id, b, w, x)
evstore CastB(id, b)

Voters_X = $!\text{Vote}_X$ | !Verify

Vote_X for $X \in \{L, R\}$

$\text{In}(id, v_L, v_R)$, Honest(id),
w ← Auth(id)
$(b_L, x_L) \leftarrow \text{Ballot}(id, v_L, w)$
$(b_R, x_R) \leftarrow \text{Ballot}(id, v_R, w)$
$\text{Cast}(id, b_X, w, x_X)$
evstore $\text{CastB}(id, b_L, b_R)$

Tally_R := for all (b, id) s.t. BBtally(cr, b),
let id be s.t. Reg(id, cr) , // **well-defined by ballot integrity**
if $\neg$Corr(id) $\wedge$ CastB(id, b_L, b) **then** TallyB(b_L) **else** TallyB(b).
Tally_L := ∀b s.t. BBtally(cr, b), do TallyB(b) // **note:** Tally_L = Tally

Property specifications for integrity, privacy, and receipt-freeness

Left and right specifications:

$$\mathcal{S}_L = !\text{Voters}_L \mid \text{System} \mid \text{Tally}_L \mid \mathcal{A}$$
$$\mathcal{S}_R = !\text{Voters}_R \mid \text{System} \mid \text{Tally}_R \mid \mathcal{A}$$

Simple ballot integrity : $\mathcal{S} \models \Phi^{bi}$, which implies $\mathcal{S}_R \models \Phi^{bi}_R$, where

$\Phi^{bi} = \text{BBtally}(cr, b) \Rightarrow \text{Reg}(id, cr) \wedge (\text{CastB}(id, b) \vee \text{Corr}(id))$

$\Phi^{bi}_R = \text{BBtally}(cr, b) \Rightarrow \text{Reg}(id, cr) \wedge (\text{CastB}(id, b_L, b) \vee \text{Corr}(id))$

Static corruption: for $X \in \{L, R\}$,

$$\mathcal{S}_X \models \text{Honest}(id) \wedge \text{Corr}(id) \Rightarrow \text{false}$$

Ballot privacy : $\mathcal{S}_L \approx \mathcal{S}_R$

Receipt-freenes : $!\text{Voter}^{rf}_L \mid !\text{Voters}_L \mid \text{System} \mid \text{Tally}_L \mid \mathcal{A} \approx$
$!\text{Voter}^{rf}_R \mid !\text{Voters}_R \mid \text{System} \mid \text{Tally}_R \mid \mathcal{A}$

Voter^{rf}_X for $X \in \{L, R\}$

$\text{In}(id, v_L, v_R)$, **if** X = L **then** $\text{VResist}(id, v_L, v_R)$ **else** $\text{VObey}(id, v_R)$
where
VResist : is the strategy for voting v_L while coerced to vote v_R
VObey : the voter votes v_R and forwards all data to $\mathcal{A}$

Figure 4: E-voting specifications for privacy.

In order to avoid trivial attacks based on differences in the outcome introduced by the experiment, the tally function computes the real outcome for honest voters in both worlds. As shown in [30], when infrastructure is corrupt, determining this outcome correctly requires the definition and proof of additional ballot box integrity properties, and corresponding recovery functions to determine the ballots to be tallied to obtain the real outcome.

A symbolic version of the BPRIV property has recently been introduced in [34]. We will propose a definition that is similar to theirs, but that allows for a more general structure of the vote and election processes. For technical reasons related to their proof methods, they assume for example that each voter sends their ballot on a separate channel. They also have a special version of the definition for when revoting is allowed, since it is assumed also that each new voting session happens on a separate channel. A more fundamental difference between our definition and theirs is the way in which corrupt infrastructure is handled. In their definition, while the ballot is sent to $\mathcal{A}$ after it is created, it is also directly added to the ballot box on the voter side; there is no (honest or corrupt) server functionality for casting the ballot. This hardcodes in the specification the fact that the ballots are honestly cast (except they may be blocked by blocking the voter process), and mirrors earlier computational BPRIV definitions [14]. Indeed, the conclusion in [34] mentions a more advanced model of a malicious ballot box in the style of [30] as future work.

Our symbolic definition more directly matches the protocol specification for each component, and explicitly allow $\mathcal{A}$ to control the functionality of each party if it corrupts it. As in earlier BPRIV definitions, we will set up a left versus right world security experiment. In order to compute the expected outcome for the right world in case of malicious components, we will rely on the property of ballot integrity. Formally this will have the same effect as in [34], allowing only honestly generated ballots to be tallied for honest voters. The difference is that this property is outside the protocol specification and it is something that we will actually prove. This approach can be seen as a particular instance of using a recovery function in the style of [30], translated to the symbolic model. To apply [30], users have to define a ballot box integrity property and an associated recovery function for each protocol. The guaranteed level of privacy then depends on the strength or weakness of the corresponding property of ballot box integrity. We think that our proposal to use ballot integrity is currently the most usable way to reconcile theory and practice: it allows to define and prove privacy, while also being a property satisfied by most current e-voting systems. Both properties, of privacy and integrity, can be directly given as input to automated tools like ProVerif or Tamarin.

Definition 5.1. A voting specification $\mathcal{S}$ satisfies *ballot privacy* if:

- Vote and Tally processes from $\mathcal{S}$ are as prescribed in Fig. 4,
- for $\mathcal{S}_L$ and $\mathcal{S}_R$ are defined as in Figure 4, $\mathcal{S}_L \approx \mathcal{S}_R$, and
- $\mathcal{S}_R$ satisfies ballot integrity, and $\mathcal{S}_L, \mathcal{S}_R$ satisfy static corruption - as defined in Figure 4.

Receipt-freeness. In the privacy definition, the votes in the left world can be interpreted as the voting intentions of honest voters, while those on the right as the ones expected by the adversary. When the voter is coerced and has to provide to the adversary $\mathcal{A}$ any data that it obtained from the voting process, the indistinguishability of the two worlds may not hold directly. Taking EEV as example, if the voter forwards the QR code to $\mathcal{A}$, then the verification procedure allows $\mathcal{A}$ to derive the vote. For such cases, some systems that attempt to achieve receipt-freeness, like EEV, JCJ/Civitas [22, 40] or Selene [47] define a coercion-resistance strategy (although the notions of receipt-freeness and coercion-resistance are sometimes considered separately [33], they belong to the same spectrum where $\mathcal{A}$ is allowed to exert influence and interact with the voter). To define receipt-freeness, we assume that:

$\mathsf{VResist}(\mathsf{id}, \mathsf{v_L}, \mathsf{v_R})$ for $\mathsf{Honest}(\mathsf{id})$

$\mathsf{w} \leftarrow \mathsf{Auth}(\mathsf{id}), (\mathsf{b_1}, \mathsf{x_1}) \leftarrow \mathsf{Ballot}(\mathsf{id}, \mathsf{v_R}, \mathsf{w}), \mathsf{Cast}(\mathsf{id}, \mathsf{b_1}, \mathsf{w}, \mathsf{x_1}), \mathsf{Out}(\mathsf{t_{leak}}),$
$\mathsf{WaitOK}, (\mathsf{b_2}, \mathsf{x_2}) \leftarrow \mathsf{Ballot}(\mathsf{id}, \mathsf{v_L}, \mathsf{w}), \mathsf{Cast}(\mathsf{id}, \mathsf{b_2}, \mathsf{w}, \mathsf{x_2}).$

$\mathsf{VObey}(\mathsf{id}, \mathsf{v_R})$

$\mathsf{w} \leftarrow \mathsf{Auth}(\mathsf{id}), (\mathsf{b_1}, \mathsf{x_1}) \leftarrow \mathsf{Ballot}(\mathsf{id}, \mathsf{v_R}, \mathsf{w}), \mathsf{Cast}(\mathsf{id}, \mathsf{b_1}, \mathsf{w}, \mathsf{x_1}), \mathsf{Out}(\mathsf{t_{leak}}).$

Figure 5: Processes for receipt-freeness by revoting

- **VResist:** on the left, coerced honest voters apply the coercion-resistance strategy to cast their intended vote.
- **VObey:** on the right, voters obey the coercer instructions.

and extend the indistinguishability experiment with these two types of processes. The actual definition of VResist and VObey, as we show below, will depend on the protocol specification and on the assumed adversarial influence.

Definition 5.2. Let $\mathcal{S}$ be a specification that satisfies Definition 5.1. In addition, assume we have processes $\mathsf{Voters}^{\mathsf{rf}}_{\mathsf{L}}, \mathsf{Voters}^{\mathsf{rf}}_{\mathsf{R}}$ as in Figure 4. Then $\mathcal{S}$ satisfies *receipt-freeness* iff

$$!\,\mathsf{Voters}^{\mathsf{rf}}_{\mathsf{L}} \mid \mathcal{S}_{\mathsf{L}} \approx !\,\mathsf{Voters}^{\mathsf{rf}}_{\mathsf{R}} \mid \mathcal{S}_{\mathsf{R}}$$

The coercion scenario considered in $\mathsf{EEV}^+ \in \{\mathsf{EEV}^\star, \mathsf{EEV}^{\mathsf{ntfy}}\}$ is that $\mathcal{A}$ is able to personally influence the voter for example by over-the-shoulder coercion or by asking the voter to forward it any information it received while casting the vote. We call this the coerced voting session. However, at some later time while the election is still open, the voter is assumed to no longer be under the influence of $\mathcal{A}$ and be able to cast a vote on a trusted device. Then, we have the following coercion-resistance strategies:

- in $\mathsf{EEV}^+_{\mathsf{any}}$: the voter is instructed to revote at any later time when it is no longer controlled by $\mathcal{A}$.
- in $\mathsf{EEV}^+_{\mathsf{last}}$: the voter has to revote, but only after the vote verification window for the coerced session expired.

We say that an e-voting specification $\mathcal{S}$ satisfies *receipt-freeness by revoting* if it satisfies Definition 5.2 using the VResist and VObey processes from Figure 5, where $\mathsf{t_{leak}}$ models data from the voting session that the voter can leak to $\mathcal{A}$, while WaitOK models the conditions when revoting is prescribed. In EEV^+, we assume $\mathsf{t_{leak}}$ is the QR code that the voter obtained from the first session, that is equal to $(\mathsf{vid}, \mathsf{r}, [\mathsf{sc_{id}}]^\star)$. Note that $\mathcal{A}$ can then obtain b from VC by requesting ballot verification for the corresponding vid. In $\mathsf{EEV}^+_{\mathsf{any}}$, WaitOK is empty, while in $\mathsf{EEV}^+_{\mathsf{last}}$ it says the revote should happen only after the verification time for the previous ballot has passed. Based on the Expired table we introduced for VC, we can model this by $\mathsf{WaitOK}; \mathsf{P} \equiv \mathtt{get\ Expired}(\mathsf{vid}),$ **if** $\mathsf{vid} = \mathsf{vid_1}$ **then** P, for any process P, where $\mathsf{vid_1}$ is obtained from first session.

5.4 Verification results and discussion

Verification results for all cases are presented in Table 1, showing that the current version of EEV is secure only in the basic scenario $\mathcal{A}_1$ where all the parties are honest, except the voters. In other cases we obtain attacks. Apart from the positive results for EEV^+, we should also note some limitations. First, we can see that receipt-freeness does not hold if one of the backend parties VC, RS or TMS is corrupt. This is because they can see the ballots cast by voters, so voters cannot revote without being detected by $\mathcal{A}$. We can hide this information from RS or the TMS by asking them to sign hashed data, instead of information in the clear. Yet, the VC has to authenticate voters and check the signatures on ballots, so a fundamental change is required to obtain receipt-freeness when the VC is corrupt. Interestingly, a similar trust assumption is also needed in BeleniosRF [18], even if it relies on more complex cryptographic constructions. BeleniosRF also satisfies a weaker notion of verifiability, since the voter cannot directly verify the vote encoded inside the ballot, but has to trust the voting application, or perform a Benaloh challenges in the style of Helios.

Another limitation of EEV^+ is that, when the voting application is corrupt, it satisfies only weak result integrity, according to the terminology in [10, 12]. This means that, if the voting application is corrupt and the voter does not verify the ballot, $\mathcal{A}$ can do ballot stuffing, i.e. cast any ballot in the name of that voter. This, however, is a more general limitation of current e-voting systems. Although systems like Belenios are proved to satisfy a strong form of result integrity and prevent ballot stuffing [11, 12, 24], this holds only under the assumption that the voting platform is trusted. Under this assumption, we also prove strong result integrity for EEV^+. Another trust assumption in EEV^+ is shown by the negative results for $\mathcal{A}_9$: at least one of the VC and RS should be honest in order to achieve verifiability. While this is the original goal of EEV, we think achieving verifiability when both are corrupt should be possible.

6 CONCLUSION AND FUTURE WORK

We perform a first thorough formal security analysis of the Estonian E-voting protocol, relying on ProVerif and Tamarin. We discover new attacks against individual verifiability and vote privacy, and rediscover some recent attacks. In the light of this analysis, we propose solutions to improve privacy and verifiability of the protocol, that we aim to further improve in future work. For example: all current EEV variants can suffer from ballot stuffing if the voting application is corrupt and voters do not verify their ballots; no variant satisfies receipt-freeness if the vote collector is corrupt. It would be interesting to see if weaker trust assumptions could be achieved in practice without a loss of usability.

We propose the first definition and automated proofs that allow to derive privacy guarantees even when any number of parties and infrastructure can be corrupt. For our framework to be applicable, it is only needed to prove that ballot integrity holds, which is a general and natural property. In future work we aim to further extend the scope of our framework to consider result integrity instead of ballot integrity, to cover, for example, BeleniosRF that relies on ballot re-randomisation. Automation needs to be improved to cover the equational theories of re-randomisation and homomorphic encryption, and attacks like in [45], currently outside the scope of automated tools. An end goal would be an unified model a la SAPIC [19] to automatically prove all the desired properties from a single specification file.

ACKNOWLEDGMENTS

This work was supported by the Luxembourg National Research Fund (FNR) under the grant agreement C22/IS/17238244/AVVA.

REFERENCES

[1] a1 [n. d.]. Additional material: code files for the specifications. https://github.com/sbaloglu/eev-codes

[2] a2 [n. d.]. Statistics about internet voting in Estonia. https://www.valimised.ee/en/archive/statistics-about-internet-voting-estonia

[3] a3 [n. d.]. Estonian online voting system. https://github.com/valimised/ivxv https://github.com/valimised/ivxv.

[4] a4 [n. d.]. Helios - Verifiable Online Elections. https://heliosvoting.org/ https://heliosvoting.org/.

[5] a5 [n. d.]. Belenios - Verifiable Online Voting System. https://belenios.org/ https://belenios.org/.

[6] a6 [n. d.]. SwissPost e-voting system. https://gitlab.com/swisspost-evoting https://gitlab.com/swisspost-evoting.

[7] a7 [n. d.]. Tamarin Prover. https://tamarin-prover.github.io

[8] a8 [n. d.]. ProVerif: Cryptographic protocol verifier in the formal model. https://bblanche.gitlabpages.inria.fr/proverif/

[9] Kushal Babel, Vincent Cheval, and Steve Kremer. 2020. On the semantics of communications when verifying equivalence properties. *J. Comput. Secur.* 28, 1 (2020), 71–127. https://doi.org/10.3233/JCS-191366

[10] Sevdenur Baloglu, Sergiu Bursuc, Sjouke Mauw, and Jun Pang. 2021. Election Verifiability Revisited: Automated Security Proofs and Attacks on Helios and Belenios. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*. IEEE, 1–15. https://doi.org/10.1109/CSF51468.2021.00019

[11] Sevdenur Baloglu, Sergiu Bursuc, Sjouke Mauw, and Jun Pang. 2021. Provably Improving Election Verifiability in Belenios. In *Electronic Voting*, Robert Krimmer, Melanie Volkamer, David Duenas-Cid, Oksana Kulyk, Peter Rønne, Mihkel Solvak, and Micha Germann (Eds.). Springer International Publishing, Cham, 1–16.

[12] Sevdenur Baloglu, Sergiu Bursuc, Sjouke Mauw, and Jun Pang. 2023. Election Verifiability in Receipt-Free Voting Protocols. In *36th IEEE Computer Security Foundations Symposium, CSF 2023, Dubrovnik, Croatia, July 10-14, 2023*. IEEE, 59–74. https://doi.org/10.1109/CSF57540.2023.00005

[13] Gilles Barthe, Ugo Dal Lago, Giulio Malavolta, and Itsaka Rakotonirina. 2022. Tidy: Symbolic Verification of Timed Cryptographic Protocols. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM, 263–276. https://doi.org/10.1145/3548606.3559343

[14] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. 2015. SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 499–516. https://doi.org/10.1109/SP.2015.37

[15] Bruno Blanchet. 2016. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends in Privacy and Security* 1, 1-2 (2016), 1–135. https://doi.org/10.1561/3300000004

[16] Bruno Blanchet, Martín Abadi, and Cédric Fournet. 2008. Automated verification of selected equivalences for security protocols. *J. Log. Algebraic Methods Program.* 75, 1 (2008), 3–51. https://doi.org/10.1016/J.JLAP.2007.06.002

[17] Bruno Blanchet, Vincent Cheval, and Véronique Cortier. 2022. ProVerif with Lemmas, Induction, Fast Subsumption, and Much More. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 69–86. https://doi.org/10.1109/SP46214.2022.9833653

[18] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. 2016. BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme. In *23rd ACM Conference on Computer and Communications Security (CCS'16)*. ACM, Vienna, Austria, 1614–1625. https://doi.org/10.1145/2976749.2978337

[19] Vincent Cheval, Charlie Jacomme, Steve Kremer, and Robert Künnemann. 2022. SAPIC+: protocol verifiers of the world, unite!. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 3935–3952. https://www.usenix.org/conference/usenixsecurity22/presentation/cheval

[20] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. 2018. DEEPSEC: Deciding Equivalence Properties in Security Protocols Theory and Practice. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 529–546. https://doi.org/10.1109/SP.2018.00033

[21] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. 2020. The Hitchhiker's Guide to Decidability and Complexity of Equivalence Properties in Security Protocols. In *Logic, Language, and Security - Essays Dedicated to Andre Scedrov on the Occasion of His 65th Birthday (Lecture Notes in Computer Science, Vol. 12300)*, Vivek Nigam, Tajana Ban Kirigin, Carolyn L. Talcott, Joshua D. Guttman, Stepan L. Kuznetsov, Boon Thau Loo, and Mitsuhiro Okada (Eds.). Springer, 127–145. https://doi.org/10.1007/978-3-030-62077-6_10

[22] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. 2008. Civitas: Toward a Secure Voting System. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*. 354–368. https://doi.org/10.1109/SP.2008.32

[23] Véronique Cortier. 2017. Electronic Voting: How Logic Can Help. In *Implementation and Application of Automata - 22nd International Conference, CIAA 2017, Marne-la-Vallée, France, June 27-30, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10329)*, Arnaud Carayol and Cyril Nicaud (Eds.). Springer, xi–xii. https://link.springer.com/content/pdf/bfm%3A978-3-319-60134-2%2F1.pdf

[24] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, and Bogdan Warinschi. 2018. Machine-Checked Proofs for Electronic Voting: Privacy and Verifiability for Belenios. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 298–312. https://doi.org/10.1109/CSF.2018.00029

[25] Véronique Cortier, Constantin Cătălin Drăgan, François Dupressoir, Benedikt Schmidt, Pierre-Yves Strub, and Bogdan Warinschi. 2017. Machine-Checked Proofs of Privacy for Electronic Voting Protocols. In *IEEE Symposium on Security and Privacy*. 993–1008. https://doi.org/10.1109/SP.2017.28

[26] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. 2019. BeleniosVS: Secrecy and Verifiability Against a Corrupted Voting Device. In *32nd IEEE Computer Security Foundations Symposium, Hoboken, NJ, USA, June 25-28, 2019*. 367–381. https://doi.org/10.1109/CSF.2019.00032

[27] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. 2019. Belenios: A Simple Private and Verifiable Electronic Voting System. In *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows (Lecture Notes in Computer Science, Vol. 11565)*, Joshua D. Guttman, Carl E. Landwehr, José Meseguer, and Dusko Pavlovic (Eds.). Springer, 214–238. https://doi.org/10.1007/978-3-030-19052-1_14

[28] Véronique Cortier and Steve Kremer (Eds.). 2011. *Formal Models and Techniques for Analyzing Security Protocols*. Cryptology and Information Security Series, Vol. 5. IOS Press. http://www.iospress.nl/loadtop/load.php?isbn=9781607507130

[29] Véronique Cortier and Joseph Lallemand. 2018. Voting: You Can't Have Privacy without Individual Verifiability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 53–66. https://doi.org/10.1145/3243734.3243762

[30] Véronique Cortier, Joseph Lallemand, and Bogdan Warinschi. 2020. Fifty Shades of Ballot Privacy: Privacy against a Malicious Board. In *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*. IEEE, 17–32. https://doi.org/10.1109/CSF49147.2020.00010

[31] Véronique Cortier and Ben Smyth. 2011. Attacking and Fixing Helios: An Analysis of Ballot Secrecy. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*. IEEE Computer Society, 297–311. https://doi.org/10.1109/CSF.2011.27

[32] Cas Cremers, Charlie Jacomme, and Philip Lukert. 2023. Subterm-Based Proof Techniques for Improving the Automation and Scope of Security Protocol Analysis. In *36th IEEE Computer Security Foundations Symposium, CSF 2023, Dubrovnik, Croatia, July 10-14, 2023*. IEEE, 200–213. https://doi.org/10.1109/CSF57540.2023.00001

[33] Stéphanie Delaune, Steve Kremer, and Mark Ryan. 2009. Verifying privacy-type properties of electronic voting protocols. *J. Comput. Secur.* 17, 4 (2009), 435–487. https://doi.org/10.3233/JCS-2009-0340

[34] Stéphanie Delaune and Joseph Lallemand. 2022. One Vote Is Enough for Analysing Privacy. In *Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13554)*, Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng (Eds.). Springer, 173–194. https://doi.org/10.1007/978-3-031-17140-6_9

[35] Sven Heiberg, Kristjan Krips, and Jan Willemson. 2020. Planning the next steps for Estonian Internet voting. *Proceedings of the E-Vote-ID* (2020), 82.

[36] Sven Heiberg, Peeter Laud, and Jan Willemson. 2011. The Application of I-Voting for Estonian Parliamentary Elections of 2011. In *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 7187)*, Aggelos Kiayias and Helger Lipmaa (Eds.). Springer, 208–223. https://doi.org/10.1007/978-3-642-32747-6_13

[37] Sven Heiberg, Tarvi Martens, Priit Vinkel, and Jan Willemson. 2016. Improving the Verifiability of the Estonian Internet Voting Scheme. In *Electronic Voting - First International Joint Conference, E-Vote-ID 2016, Bregenz, Austria, October 18-21, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 10141)*, Robert Krimmer, Melanie Volkamer, Jordi Barrat, Josh Benaloh, Nicole J. Goodman, Peter Y. A. Ryan, and Vanessa Teague (Eds.). Springer, 92–107. https://doi.org/10.1007/978-3-319-52240-1_6

[38] Sven Heiberg and Jan Willemson. 2014. Verifiable internet voting in Estonia. In *6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014, Lochau / Bregenz, Austria, October 29-31, 2014*, Robert Krimmer and Melanie Volkamer (Eds.). IEEE, 1–8. https://doi.org/10.1109/EVOTE.2014.7001135

[39] Lucca Hirschi, Lara Schmid, and David A. Basin. 2021. Fixing the Achilles Heel of E-Voting: The Bulletin Board. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*. IEEE, 1–17. https://doi.org/10.1109/CSF51468.2021.00016

[40] Ari Juels, Dario Catalano, and Markus Jakobsson. 2005. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*. 61–70. https://doi.org/10.1145/1102199.1102213

[41] Steve Kremer, Mark Ryan, and Ben Smyth. 2010. Election Verifiability in Electronic Voting Protocols. In *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6345)*, Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou (Eds.). Springer, 389–404. https://doi.org/10.1007/978-3-642-15497-3_24

[42] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2011. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. In *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*. 538–553. https://doi.org/10.1109/SP.2011.21

[43] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *25th International Conference on Computer Aided Verification (Lecture Notes in Computer Science, Vol. 8044)*. Springer. https://doi.org/10.1007/978-3-642-39799-8_48

[44] David Mestel, Johannes Mueller, and Pascal Reisert. 2022. How Efficient are Replay Attacks against Vote Privacy? A Formal Quantitative Analysis. In *35th IEEE Computer Security Foundations Symposium, CSF*.

[45] Johannes Müller. 2022. Breaking and Fixing Vote Privacy of the Estonian E-Voting Protocol IVXV. In *7th Workshop on Advances in Secure Electronic Voting, FC22*. https://orbilu.uni.lu/handle/10993/49442

[46] Olivier Pereira. 2022. Individual Verifiability and Revoting in the Estonian Internet Voting System. In *7th Workshop on Advances in Secure Electronic Voting, FC22*. https://eprint.iacr.org/2021/1098

[47] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. 2016. Selene: Voting with Transparent Verifiability and Coercion-Mitigation. In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 9604)*. Springer, 176–192. https://doi.org/10.1007/978-3-662-53357-4_12

[48] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Dr. Carlisle Adams. 2013. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960. https://doi.org/10.17487/RFC6960

[49] Benedikt Schmidt, Simon Meier, Cas Cremers, and David A. Basin. 2012. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, Stephen Chong (Ed.). IEEE Computer Society, 78–94. https://doi.org/10.1109/CSF.2012.25

[50] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. 2014. Security Analysis of the Estonian Internet Voting System. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*. ACM.

[51] Anggrio Sutopo, Thomas Haines, and Peter Roenne. 2023. On the Auditability of the Estonian IVXV System and an Attack on Individual Verifiability. In *Workshop on Advances in Secure Electronic Voting*.

[52] Misni Suwito, Bayu Tama, Bagus Santoso, Sabyasachi Dutta, Haowen Tan, Ueshige Yoshifumi, and Kouichi Sakurai. 2022. A Systematic Study of Bulletin Board and Its Application. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security* (Nagasaki, Japan) *(ASIACCS 2022)*. Association for Computing Machinery, New York, NY, USA, 1213–1215. https://doi.org/10.1145/3488932.3527280

[53] Robert Zuccherato, Patrick Cain, Dr. Carlisle Adams, and Denis Pinkas. 2001. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). RFC 3161. https://doi.org/10.17487/RFC3161

Received 7 December 2023; accepted 13 March 2024