UNIVERSITÉ DE LORRAINE | BIBLIOTHÈQUES UNIVERSITAIRES

AVERTISSEMENT
DROIT D'AUTEUR

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document. Toute contrefaçon, plagiat ou reproduction illicite encourt une poursuite pénale.

Contact bibliothèque : ddoc-theses-contact@univ-lorraine.fr
(Cette adresse ne permet pas de contacter les auteurs)

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10

UNIVERSITÉ DE LORRAINE
34 COURS LÉOPOLD — CS 25233
54052 NANCY CEDEX
TÉL : 03 54 50 54 00
CONTACT@UNIV-LORRAINE.FR
WWW.UNIV-LORRAINE.FR

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Formal Methods for Internet Voting

THÈSE

soutenance publique le 3. mars 2026

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Florian Moser

Composition du jury

Président :	Emmanuel Thomé	Directeur de recherche, INRIA, France
Rapporteurs :	Ioana Boureanu	Professor, University of Surrey, United Kingdom
	Olivier Pereira	Professor, University of Louvain, Belgium
Examinateurs :	Thomas Haines	Senior Lecturer, Australian National University, Australia
	Elizabeth Quaglia	Reader, University of London, United Kingdom
Invité :	Melanie Volkamer	Professorin, Karlsruher Institut für Technologie, Germany
Encadrants :	Veronique Cortier	Directrice de recherche, CNRS, France
	Alexandre Debant	Chargé de recherche, INRIA, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Abstract

Internet voting refers to elections performed over the internet, where the voters use their own personal devices to cast their ballot. Such a system, while fundamentally different to paper-based elections, needs to provide the same security guarantees as traditional election systems. This includes the verifiability of the correctness of the election, while safeguarding the secrecy of the individual votes.

In the first part of the thesis, we explore which security mechanisms are employed by vendors. To this end, we perform a systematic market study and review academic literature to find systems used in practice, which we then analyse to the extent possible. Our work documents that most industry systems are effectively black-box systems, hence do not provide any meaningful notion of verifiability. However, we also find multiple systems that do indeed implement state-of-the-art mechanisms, thereby demonstrating their practicality.

In the second part of the thesis, we propose a novel mechanism to let voters cast their ballot in secret, even if their own personal device is surveilled. It is based on code voting, where the voter enters a code instead of a candidate. However, we design the mechanism in such a way that the codes are very *short* (typically one or two digits), fit to be entered by hand by the voter. We embed the mechanism in a protocol that is compliant to the highly demanding security requirements set by the Swiss Chancellery for Swiss political elections. Still, the protocol is comparatively simple, and only relies on standard primitives.

In the third part of the thesis, we present a framework to support proofs of internet voting protocols. The framework models an extensible election setting, into which a concrete protocol can be plugged into. ProVerif, which is an automatic prover suitable for cryptographic protocols, can then be used to prove both verifiability and privacy. Our work builds upon a recently proposed framework for verifiability, which we extend towards privacy, and to which we remain backwards-compatible. As part of this work, we also improve the expressiveness of the syntax of lemmas in ProVerif itself, which is of more general interest. We successfully apply the proof framework to several protocols of the literature and industry, including Belenios, Swiss Post, and our own short voting codes proposal.

Résumé

Le vote par internet fait référence aux élections effectuées via internet, où les électeurs utilisent leurs propres appareils personnels pour exprimer leur vote. Un tel système, bien que fondamentalement différent des élections sur papier, doit fournir les mêmes garanties de sécurité que les systèmes électoraux traditionnels. Cela inclut la vérifiabilité de l'intégrité du résultat de l'élection, tout en préservant le secret de vote.

Dans la première partie de la thèse, nous explorons les mécanismes de sécurité employés par l'industrie. À cette fin, nous effectuons une étude systématique du marché et examinons la littérature académique pour identifier les systèmes utilisés, que nous analysons ensuite. Notre travail documente le fait que la plupart des systèmes de l'industrie sont effectivement des systèmes en boîte noire, ne fournissant aucune assurance significative de vérifiabilité. Cependant, nous avons également relevé plusieurs systèmes qui mettent en œuvre des mécanismes à la pointe de la technologie, démontrant ainsi leur praticité.

Dans la deuxième partie de la thèse, nous proposons un nouveau mécanisme permettant aux électeurs de voter en toute confidentialité, même si leur propre appareil est compromis. Notre protocole est basé sur le vote par code, où l'électeur saisi un code au lieu d'un candidat. Nous avons conçu le système de manière à ce que les codes soient très *courts* (typiquement un ou deux chiffres), adaptés à une saisie manuelle par l'électeur. Nous avons intégré ce mécanisme dans un protocole qui respecte les exigences de sécurité très strictes établies par la Chancellerie suisse pour les élections politiques suisses. Néanmoins, le protocole reste simple et repose uniquement sur des primitives standard.

Dans la troisième partie de la thèse, nous présentons un framework permettant d'obtenir des preuves formelles de sécurité pour des protocoles de vote sur internet. Ce framework définit un contexte électoral général, dans lequel un protocole concret peut être intégré. ProVerif, qui est un prouver automatique adapté aux protocoles cryptographiques, peut alors être utilisé pour prouver à la fois la vérifiabilité et le secret de vote. Notre travail s'appuie sur un framework récemment proposé pour la vérifiabilité, que nous étendons au secret du vote, tout en restant compatible avec les versions antérieures. Nous améliorons également l'expressivité de la syntaxe des lemmes dans ProVerif lui-même, ce qui présente un intérêt plus général. Nous avons appliqué avec succès le framework de preuve à plusieurs protocoles de la littérature et de l'industrie, y compris Belenios, Swiss Post, et notre propre proposition de vote par codes courts.

Acknowledgements[1]

First and foremost, I would like to thank my two advisors Veronique Cortier and Alexandre Debant which have spent so many hours with me sharing their knowledge and craftsmanship. It is an honour and a joy to collaborate in such an honest and direct manner. We have broken, proposed and proved internet voting systems, and laid the groundwork for the proofs of the secure systems of tomorrow, roughly in that order. I am also grateful for your flexibility to creative institutional arrangements, allowing me to pursue the thesis in the first place, but also granting space for many personally cherished projects during the thesis. This notably included the verification of real world elections, and the authoring of an internet voting study for the German federal administration, which both was very engaging.

I have had the pleasure to work across domains, institutions and countries. Pierrick Gaudry helped me to explore the foundations of internet voting, as far as to the implementation of elliptic curve algorithms. Vincent Cheval invited me on a trip to the intestines of ProVerif. Johannes Müller and Michael Kirsten contributed greatly to my understanding of cryptographic building blocks, mechanisms and protocols. Melanie Volkamer and Tobias Hilt brought me closer to the human factors perspective, to improve usability and understandability from the user's point of view. Rolf Hänni and Oliver Spycher lightened the spark to even explore internet voting from a scientific perspective, amplified by Kenny Paterson during my Master's Thesis, and fundamentally made possible by the excellent courses offered at ETH Zürich. Altogether, they have given me a toolbox to improve real-world internet voting systems, a persistent vision since I was first asked to develop such a system eight years ago.

Many others have as of yet been unnamed. Scientific exploration (and personal development) needs the environment full of people eager to discuss their and your favorite hyper-fixation. I am very greatful to have had access to this both at Loria as well as at ETH. Further, I needed the stability and habitual escape granted by my partner, family and friends. Thank you for listening to my lengthy rants, flimsy conspiracy theories and machiavellian plans to improve the (internet voting) world, and laughing at my memes and jokes. Many more will follow.

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

Contents

1 Introduction **1**
- 1.1 Properties . . . 2
- 1.2 Mechanisms . . . 4
- 1.3 The Swiss setting . . . 8
- 1.4 Formal methods . . . 9
- 1.5 Contributions . . . 11
  - 1.5.1 Contributions presented in this thesis . . . 11
  - 1.5.2 Contributions not part of this thesis . . . 12
  - 1.5.3 Software contributions . . . 15

2 Mechanisms used in practice **17**
- 2.1 Method . . . 18
  - 2.1.1 Discovery of Systems . . . 18
  - 2.1.2 Extraction of Mechanisms from Systems . . . 22
- 2.2 Analysis . . . 25
  - 2.2.1 Secret ballots . . . 25
  - 2.2.2 Verifiable tally . . . 26
  - 2.2.3 Individual verification . . . 28
  - 2.2.4 Authentication . . . 29
- 2.3 Related work . . . 31
- 2.4 Discussion . . . 32

3 Short Voting Codes **35**
- 3.1 Swiss setting . . . 36
- 3.2 Related Work . . . 37
- 3.3 Voter's view . . . 41
- 3.4 Protocol . . . 42
  - 3.4.1 Setup . . . 43
  - 3.4.2 Voting . . . 46
  - 3.4.3 Tally . . . 47
- 3.5 Security Claims . . . 47
- 3.6 Discussion . . . 49

4 ProVerif Proof Framework **53**
4.1 ProVerif . . . 54
4.2 Running example . . . 61
4.3 Verifiability framework . . . 63
4.3.1 Verifiability notion . . . 63
4.3.2 Model . . . 64
4.3.3 Limitations . . . 67
4.4 Add support for vote privacy . . . 68
4.4.1 The privacy definition . . . 68
4.4.2 Problem statement . . . 69
4.4.3 Model a simple tally . . . 70
4.4.4 Modelling a verifiable decryption shuffle . . . 72
4.4.5 Example . . . 76
4.4.6 Limitations . . . 77
4.5 How to prove privacy? . . . 78
4.5.1 Transformations of the verifiability framework . . . 78
4.5.2 Architecture of the voting framework . . . 79
4.5.3 Ballot swap . . . 80
4.5.4 Balancing precision and non-termination . . . 81
4.5.5 The cellBB lemma . . . 83
4.6 Extending ProVerif . . . 84
4.6.1 A generic cellBB lemma . . . 84
4.6.2 Add destructors and letfun to queries . . . 86
4.7 Case Studies . . . 87
4.7.1 Running example . . . 89
4.7.2 Belenios . . . 90
4.7.3 Swiss Post Protocol . . . 94
4.7.4 Short Voting Codes . . . 96
4.7.5 Vote&Check . . . 96
4.8 Discussion . . . 97

5 Conclusion **101**
5.1 Future research . . . 102
5.1.1 Modular systems and proofs . . . 102
5.1.2 Verifiability notions . . . 102
5.1.3 E-Collecting . . . 104

A Contributions **107**
A.1 PHP Bugfixes . . . 107

B Future work **109**
B.1 Example: Fine-grained verifiability notions . . . 109

C Introduction (French) **111**

Bibliography **131**

Chapter 1

Introduction

Internet voting aims to perform elections over the internet. For some parts of the electorate, internet voting has a clear benefit and increases turnout, for example for Australians living in the outback where the next polling place is far away [NSW23], for expats for whom postal mail is too slow [OEAMiEOfDIR08, DMSTW12, Ger21], or people with disabilities who otherwise cannot vote independently [Bun21]. While for voters who already have a practical voting channel available, an improvement in turnout is not observed [GS17, SW24], internet voting may also simply be of long-term strategic importance (e.g., as it is the case for the Swiss administration [Bun13, fK20]), together with a general push to provide digital services to its citizens.

Given the inherent and perceived advantages of internet voting, the current state of active use of internet voting systems is less promising. Besides Estonia [Est25], no country so far provides internet voting to a significant portion of their population. The system has been updated continuously over the years since its introduction [HW14, HMVW16, HKW20], but has been and continues to be perceived critically from an academic perspective [SFD+14, Mül22, SHR23]. The Swiss Post Voting System is currently the most advanced and transparent internet voting system, collaborating tightly with researchers to review and improve their system [Swi24a]. However, also this system has had its share of controversies [LHK19, HLPT20, PT19, Bun19b], which lead to major reworks of the system and the legislative basis [Bun19a, Pos19]. It will continue to undergo major transformations [Ste23, CDE+25], while its use in Switzerland is currently limited mostly to expats [Swi13, Fed25a]. A few other countries have introduced internet voting, but again limit their use to expats (e.g., in France [DH23]), or discontinued the system after only few elections (e.g., Norway [BGS+16] and Australia [Com23]). All of these examples had their own set of controversies about the security of the employed systems [DH23, ØK12, VT19]. Clearly, implementing an internet voting system is not straightforward.

While the target of internet elections — guarding the secrecy of the vote and the correctness of the election result — remain the same as for traditional elections, the trust assumptions fundamentally change. Compared to re-

mote elections per postal mail, for which processing is complex but heavily distributed [KS19], internet elections are implemented as a mostly centralized system (e.g., [Swi24b, Val22]). Compromising these central servers potentially has a much larger impact than compromising parts of the very distributed classical elections, as attacks may scale much better and are harder to detect (e.g., as there is no paper trail to be forged, which might serve as tamper-evidence [SRF21]).

For internet voting, this means that conventional hardening of the systems (e.g., following principles of secure programming, tight access control, ...) is in many settings considered insufficient [oE17, Swi13, Est14], including (lower-stakes) non-political elections [Fre25, Fed25b]. Instead, systems are obligated to make their computations *verifiable*, hence the voter, other system components or even third-party auditors must be able to check that the ballots are proceeded as intended. This of course has to be achieved without breaking the secrecy of the vote. We extend on these properties, verifiability and vote secrecy, in the next section.

Box 1: Internet voting is not online banking

Crucially, digitalizing elections is very different to digitalizing other services, such as online banking. For example, incorrect processing in online banking — whether it be due to fraud or a simple programming error — are easily detected: The account holder has a detailed list of every inflow and outflow from their banking statement, and can pinpoint transactions they did not initiate. Then, they can complain to their bank, which can start an investigation who initiated these transactions, how much money is impacted, and where it went.

This is fundamentally different to electronic elections. First, users likely never notice if the system malfunctions, as they would need to guess only from the officially published accumulated election result whether their individual vote has been changed or dropped. Further, even if a voter complains, this complaint cannot (and should not) be answerable: By construction, the system must guard the secrecy of the vote, hence it should be impossible to trace back how an individuals vote has been processed. Compared to other secure systems, elections therefore need different techniques to be secured.

1.1 Properties

To attest that a voting system functions correctly, its functionality is built to be verifiable. Some of the verifications need to be performed directly by the voters, referred to as individual verification [SK95]. The voter needs to be

able to ensure that their ballot was created correctly (cast-as-intended), and is part of the ballot box (recorded-as-cast) [BRR+15, BBH+17, BCG+18a]. In some settings, abstaining voter may also be able to verify that no ballot of theirs has been included [Swi13, BCG+18a]. Other verifications can be performed by independent components of the system or auditors, referred to as universal verification [SK95]. This includes the tallied-as-recorded notion [BRR+15, BCG+18a], which ensures all recorded ballots are correctly tallied. Further, the system also need to ensure only eligible voters participate, which is referred to as eligibility verification [KRS10, SRKK10].

The composition of these various properties may however still leave gaps. For example, there might be clash attacks [KTV12], where a dishonest voting server provides a receipt for the same ballot to two different voters. For this purpose, E2E-Verifiability has been coined, which defines verifiability over the whole system [KTV10, CGGI14, CGK+16, BBMP20]. The formalization we use in this work attests that the final result is composed out all votes of honest voters who verify, some votes of honest voters who do not verify, plus at most one vote per dishonest voter [CGGI14]. The notion thereby relates the voter's actions on the one end directly to the result of the election on the other end, leaving no potential for gaps. Its formulation accommodates a potential active adversary, while giving it clear bounds. Honest voters who verify are guaranteed to have their vote tallied. Honest voters who skip the verification may have their vote dropped by the adversary (e.g., the adversary could prevent delivery over the internet), but their vote remains unchanged. Further, the adversary cannot insert more votes than dishonest voters it has control over.

Some systems target correctness properties that go further than verifiability of the election result. Accountability [KTV10] ensures that when a deviation from the specification is detected, it can be attributed to some malfunctioning component. Further, dispute resolution [KWC+13] aims to ensure that disputes can be correctly resolved, and dispute freeness prevents disputes from arising in the first place [KY02].

Despite the strong correctness guarantees, the system needs to guard vote secrecy. The adversary needs to be prevented to learn how the voters cast their vote, apart from what the election result leaks anyway [Ben87, DKR09]. This can be formalized as an indistinguishability property, where the attacker is unable to distinguish one world where Alice is voting 1 and Bob is voting 0, from another world where Bob is voting 1 and Alice is voting 0 [DKR09].

In addition to vote secrecy, some systems target stronger privacy guarantees. Receipt freeness captures the notion that the voting system should not hand out proof of how the vote was cast [BT94]. This is to prevent vote selling by the voter to the adversary. Making this notion even stronger, some voting

systems are also designed to be coercion resistant [MN06]: Even if a voter is coerced during parts of the voting phase (e.g., an attacker spies on them, or can even command specific behaviour), the voter should still be able to cast their vote according to their actual preference. Finally, everlasting privacy aims to ensure the secrecy of the votes indefinitely [MN06, ACKR13], even against future threats. For example, an adversary may store encrypted ballots now, and then decrypt them when quantum computers are able to break traditional means of encryptions [Sho94].

Box 2: The contraction between verifiability and privacy

Privacy and verifiability seem to be at odds with each other. For privacy, it would be best to not publish any information except the voting result. However, to verify this election result is indeed correct, the publication of additional data is necessary. This additional data needs to be *somehow* connected to the election content, which intuitively jeopardizes privacy.

This intuition has been formalized, leading to incompatibility results. For example, it has been shown that verifiability and unconditional privacy (i.e., privacy from an *unbounded* adversary) are incompatible [CMFP+10]. Conversely, it has been shown that privacy needs individual verifiability, i.e., that the votes of honest voters are actually tallied [CL18]. Otherwise, an adversary may simply drop all ballots except one, and trivially learn from the voting result the vote it contains.

1.2 Mechanisms

Reaching the somewhat contradicting properties verifiability and privacy at the same time is challenging. In in-person elections, this is achieved using physical constraints. Only authenticated voters are permitted to enter the voting area and cast a single paper ballot into a ballot box. After the voting period ends, the ballot box is shuffled and then the ballots are counted. This whole process can be verified, and in case of doubt, counting can even be repeated. Still, the privacy of the voters is preserved: They create their ballot in private in a voting booth and drop it into the ballot box, where it will be shuffled together with all other ballots before counting.

In (postal mail) remote voting, these guarantees weaken compared to the in-person elections: The voter may be observed or even coerced when creating their ballot (e.g., by a family patriarch, a party representative or others), and they have no way to know whether their ballot indeed reaches the ballot box at the election office. While at the election office some basic privacy and verifiability guarantees establish — by checking authentication information against the electoral roll, and then separating ballots from their authentica-

tion information before opening them — the process is complex and relies on honest behaviour of many individuals [KS19]. Indeed, fraud in postal voting has already been documented. For example, a dishonest municipality employee added additional paper ballots, which could only be singled out as they did not inhibit the usual folding patterns [SRF21]. In internet elections, all security mechanisms based on paper — and generally physical processes — are however inapplicable, and therefore other means of securing the election need to be deployed.

There is a plethora of cryptographic mechanisms proposed to (digitally) bridge the gap between the privacy and verifiability guarantees [BBH+17, FDL07, JMP13]. These mechanisms are then composed into end-to-end secure systems (e.g., [Ben06, CGG19, Swi24c, Val22]), with the aim of choosing an appropriate balance of verifiability and privacy, while relying on realistic trust assumptions. A simple internet voting system (e.g., [Ben06]), achieving basic privacy and verifiability guarantees, encrypts the ballot towards an election key managed by trustees, and then sends the ballot to the server over an authenticated channel. Then, after the voting phase finishes, the server verifiably shuffles[1] all received ballots, to separate them from their casting voters. Finally, the trustees decrypt the shuffled ballots.

State-of-the-art systems employ mechanisms that avoid trust in single components, e.g., the voting server or a single trustee. Using distributed key generation [CGGI13] (e.g., as implemented in Belenios [CGG19]), the election key can be generated in such a way that ballots encrypted under this key need a quorum of holders of the decryption key to decrypt. Authenticating the ballot can be made verifiable to third parties by digital signatures (e.g., as done in Estonia [Val22]), possibly combined with multi-factor authentication involving multiple authorities (e.g., as implemented in Electa [Ass23] or Belenios [CGG19]). To safeguard ballots until the tally phase, and guarantee they are not dropped or changed, they are stored at multiple components simultaneously (e.g., as implemented in the Swiss Post Voting System [Swi24c] and Estonia [Val22]) or published on (public) bulletin boards (e.g., [HL08, KKL+18, HSB21]).

During the tally phase, trust can again be distributed. Many different ways of verifying a shuffle have been proposed [SK95, JJR02, TW10, BG12], with the most established choice using a zero-knowledge proof of shuffle for its verifiability (e.g., the Bayer-Groth construction [BG12] used in the Swiss Post Protocol [Swi24c], or the Terelius-Wikström proposal [TW10] imple-

[1] The idea of *shuffling* is, given a list of ballots, to change the representation of each individual ballot and randomize their order in the list. Thereby, each ballot in the output list cannot be linked to the corresponding ballot in the input list. This shuffling is made *verifiable* by giving auditors convincing proof that the list of output ballots represents the same votes as the list of input ballots (i.e., no ballot has been dropped, added or changed). Of course, this proof must not again reveal the link between input and output ballots.

mented in Verificatum [Dou] and CHVote [HKLD17]). Instead of using verifiable shuffles, another approach homomorphically aggregates the ciphertext [CF85, Ben87] (e.g., as implemented in ElectionGuard [BNPW24]), where the resulting ciphertext is the (multiplicative or additive) combination of all the input ciphertext, i.e., the election result. This avoids the trusted component necessary for privacy of verifiable shuffles[2], however implementing arbitrary voting rules using homomorphic aggregation (including the necessary ballot validity proofs) is challenging [CGY22, HKLR24]. Finally, either the single result ciphertext from the homomorphic aggregation, or the shuffled and therefore anonymized ciphertext from the verifiable shuffle, are decrypted by the trustees (described, e.g., as part of Belenios [CGG19]).

There are many more mechanisms available to further extend the privacy and verifiability guarantees of a system. To avoid receipts with which the voter could proof to the adversary how they voted, re-randomizable signatures [CCFG16, CFL19] have been proposed. These remain valid over a randomized representation of the ballot, hence making this re-randomized ballot safe to publish while guarding strong authentication properties. To improve on coersion resistance, fake credentials allow voters to cast a ballot that will not be counted [JCJ05, CCM08]. Further, some systems allow revoting (e.g., [Val22]), where voters can replace their ballot at any time. To combine revoting with a public bulletin board, *deniable* vote updating [LQT20, MPP24, KTV15] is proposed, where the adversary does not learn whether a vote has been updated. To reach a practical notion of everlasting privacy, the use of unconditionally hiding commitments are proposed (e.g., Pederson [Ped92]), embedded in a suitable internet voting protocol (e.g., [CPP13]). The commitments will continue to hide the voter's selection even if cryptography can be broken at some future point in time.

Included in the effort to reduce trust in the system is in the remote voting setting also the (personal) voting device used by the voter. Notably in political elections these devices are often not assumed fully trustworthy (e.g., in Estonia [Est14] and Switzerland [Swi13]). Therefore, various methods have been proposed to let the voter check that their vote has been cast correctly by their own device. In the audit-or-cast approach (also called the Benaloh challenge [Ben06]), the voter device commits to a ballot, and then the voter chooses to either audit or cast this ballot. This is notably implemented in Helios [Adi08], while this challenge-based approach seems to be hard to use for voters [MKRV18]. In the cast-then-audit approach, voters are offered the possibility to audit the cast ballot. This is used in the Estonian setting [HW14] or, improving on receipt freeness through the use of an inter-

[2]While the input ballots are ensured to correspond to the output ballots using the verification mechanisms, a dishonest component may still pick a predictable ordering of ballots (e.g., the identity function), and therefore no privacy guarantees would establish.

active ZKP, in POLYAS [Tru23, MT23].[3] For settings where no second device can be assumed, but a secure channel is available from the system to the voter, return codes are proposed [AC11]. A voting sheet with a unique return code per candidate is delivered to the voter prior to the election. Then, after the voter cast their vote, they check that the return codes displayed by the voting device match the return codes received over the secure channel. This approach is chosen notably in the Swiss setting [Swi24c, HKLD17], besides others [AC12, Com19a]. Alternatively, when verification in the tally phase is acceptable, tracking codes can be used [KMST16, RRI16, DRRR24], where all counted votes are published in plain together with a tracking code. The voter then checks that their vote, identifiable by their personal tracking code, is correct.

While these verification methods ensure that the voting device (and possibly other parts of the system) did not modify the selected candidate, they do not protect privacy against a malicious voting device. For this purpose, voting codes [Cha01] have been proposed. The voter does not enter their plain choice anymore, but instead enters a code that represents this choice, with the attribution unknown to the voting device. Some approaches use voting codes which can be entered by hand [Cha01, CZZ+16], but as they can be also easily guessed by a malicious device, these approaches need to be combined with a cast-as-intended mechanism. Other proposals use longer codes that cannot be guessed by the device [RT09, HKD23, CFL19], with the trade-off that the voter needs to scan a QR-code (or similar) to cast the corresponding ballot.

Given all these different approaches and systems, it is noteworthy to highlight that not a single best system exists. The precise properties to aim for (e.g., the notion of privacy), and the trust assumptions that can be deemed acceptable, depend heavily on the election context. This leads to very different system designs. For example, the Estonian [Val22] and the Swiss Post [Swi24c] systems are both built for national political elections. However, already the assumptions concerning the voter are quite different: In Estonia [Val22], the voters are assumed to be in the (sole) possession of a personal electronic id, and to have access to a trustworthy second device. Hence, the system expects the voters to sign their ballot using their electronic id, and verify their ballot using the cast-then-audit approach on their second device. In contrast, in the Swiss setting, there is no electronic id yet, but trusted setup component and a private channel (via postal channel) is deemed acceptable. Hence, the voter receives per postal mail a code sheet with an authentication code to cast their ballot, and with return codes to verify their ballot [Swi24c].

[3] The audit in POLYAS is performed over a reencryption of the cast ballot. This audit is convincing to the voter, as an interactive ZKP is performed with the server attesting that the reencryption is correct, but the transcript of the ZKP is not convincing to a third party.

Box 3: It is trivial to fulfil only privacy or only verifiability

Given all these mechanisms with their inherent complexity, which additionally need to be combined in a sensible fashion to form a complete system, one might ask whether there are no simpler solutions. Indeed, it is trivial to build systems that provide only verifiability or only privacy. For example, a system can publish in plain who voted for which candidates, or conversely, publish nothing at all except the election result.

However, reaching acceptable notions for both privacy and verifiability remains a challenge, and requires use of a combination of some of the described mechanisms. This complexity is also reflected in real-world systems, which need many 100s of pages to fully document their approach in detail, e.g., see the Swiss Post Voting System [Swi24c].

1.3 The Swiss setting

The Swiss Federal Chancellery [Swi13] is highly demanding in terms of security for internet voting. In this setting, voting sheets are distributed per postal mail to voters, with those sheets generated by a trusted setup component, in collaboration with control components. When a voter intends to vote , they enter their candidate into the voting device, which encrypts the vote and sends the ciphertext to the control components. The control components intuitively play the role of a distributed voting server, and collaboratively compute the return codes corresponding to the specific vote. These codes are sent back to the voter, who will use the voting sheet to verify that they are correct, and then confirm their vote. As defined by the Swiss Chancellery, verifiability should hold even if the attacker controls the network, all the control components but one, and the voting device. For vote privacy, the trust assumptions are similar except that the voting device is now additionally trusted.

Besides defending against such a strong adversary, the Swiss context distinguishes through their transparency requirements and close collaboration with research. The specification needs to be fully public, as well as the code of the system. Public scrutiny is encouraged through bug bounty programs. Further, the Swiss Federal Chancellery commissions international experts to regularly review the system in use, and these reports are also public [Swi24a]. As part of these efforts, various issues have been uncovered and fixed [HLPT20, CDG22, HR25], and there is long-term planning on how to improve the system over the upcoming years [Ste23].

In this setting, two fully specified systems are proposed. The Swiss Post Voting System [Swi24c, Swi24b] is currently the only fully developed system compliant in this ambitious setting. The more research-focused CHVote

provides a specification and an implementation of the cryptographic primitives (but e.g. no user-interface) [HKLD17]. The main difference in between the system is how they generate the return codes in a privacy-preserving way. Intuitively, the Swiss Post Voting System hashes the (deterministically) encrypted vote, and then uses a hash table to recover the corresponding return code [Swi24c]. The CHVote system uses k-out-of-n oblivious transfer, to recover the k return codes of the selected candidates out of the n candidates in total [HKLD17]. Both protocols are of similar high complexity, and rely on custom cryptographic primitives. While it is hard to quantify complexity, both specifications span more than 300 pages to detail the cryptography used and the messages exchanged between the various components [HKLD17, Swi24c, Swi24d, Swi25a]. The complexity makes it hard to intuitively analyse the system and the different states it might be in (and possibly should not be in), and therefore more advanced methods to analyse these kind of systems need to be used.

1.4 Formal methods

Formal methods aims to analyse the concrete security guarantees of systems. Given a specification, the targeted security properties and a threat model, formal methods aims to answer the question whether the specification indeed fulfils these properties. Even for "simple" protocols, this precise analysis may produce a surprising amount of nuance (e.g. handshakes to agree on a shared key may achieve many different flavours of precise security properties [GHS+20]). For complex protocols, such as those necessary for state-of-the-art internet voting systems, it allows precise reasoning that is vital to understand the exact properties despite the complexity. In the Swiss setting, any complaint system must apply formal methods to demonstrate the security of the protocol [Swi13].

The general procedure of applying formal methods involves translating the specification and the properties to a corresponding model. In this model, sound reasoning is then possible, with the precision depending on accuracy of the model. There are two main formal modelling approaches, the computational and the symbolic model. Computational models assume a computationally bounded attacker, and reduce the security of a concrete system to the concrete cryptographic assumptions, such as the discrete log assumption.[4] This is typically done using pen & paper proofs (e.g., [Swi25c, HKK+22, BCG+18a]), while there are some semi-automated tools available such as EasyCrypt [BDG+12], CryptoVerif [Bla07] and Squirrel [BDJ+24]. Symbolic models are more abstract, and assume cryptography to be perfect (e.g., if the adversary does not know the decryption key, they cannot

[4]The discrete log assumption states that it hard to derive x for some given $y = g^x$. y is an element in a large finite multiplicative cyclic group, with g its generator.

decrypt). In turn, more complex systems are provable due to improved automation, and mature symbolic verification tools such as ProVerif [Bla16, BCC22] and Tamarin [MSCB13] have been successfully applied to analyze industrial scale protocols such as TLS [BBK17, CHSvdM16, CHH+17], 5G [BDH+18], Signal [KBB17] or the EMV payment protocol [BHSTP24]. In particular, ProVerif has been used to analyse voting protocols [Swi25d, CCD23, CDGL24], and is also the tool we focus on in this work.

Protocols are modelled in ProVerif using the applied pi calculus [AF01]. Properties can be expressed as reachability (e.g., whether the adversary learns some value) or correspondence queries (e.g., whether some relation between protocol states holds). Further, a flavour of equivalence can be proven, which is useful for privacy properties. ProVerif models a Dolev-Yao attacker [DY03], i.e., an attacker that can create, drop or change messages, and derive any value from its knowledge, but who cannot break cryptography. To perform a proof, ProVerif evaluates whether the attacker can break the properties in the protocol model. Under the hood, ProVerif transforms the protocol and the attacker model into a set of Horn clauses $\mathbb{C}$. Then, the clauses are saturated into $\mathbb{C}_{sat}$, i.e. clauses are combined with each other to find other implied clauses. This is continued until no more (semantically) new clauses emerge. Finally, it is verified in $\mathbb{C}_{sat}$ whether the properties hold. The output of ProVerif is proven to be sound, hence if it does not find an attack in the model, there is none. However, ProVerif might find false attacks, or may not terminate. For these cases, ProVerif provides utilities to (soundly) guide its execution, which notably has been extended recently to support lemmas [BCC22]. Lemmas specify intermediate properties that ProVerif is able to use during its execution, but which are nonetheless proven true by ProVerif beforehand (i.e., unsound lemmas lead to a failing proof).

Formal proofs about the security of internet voting protocols pushes the available tools to their limits. E2E-Verifiability relates the votes contained in the result to the votes the individual voters had cast, which intuitively requires *counting* the votes. This is hard in general for formal verification tools, but even more so in the complex electronic voting setting, where a model aims to be unbounded in elections, voters, ballots, and even revotes. Recently, a ProVerif model has been proposed to aid verifiability proofs [CCD23], formulating E2E-verifiability as a correspondence property. This framework encodes an extensible (unbounded) election setting into which concrete protocol models can be cast into. It further provides generic lemmas to aid with termination and precision. The framework has then been successfully used to prove verifiability of various case studies, notably Belenios [CGG19], Swiss Post Voting System [Swi24c] and CHVote [HKLD17]. Besides verifiability, internet voting protocols need also a privacy proof. Privacy needs to be modelled as an equivalence property, which is notoriously harder to verify than correspondence properties [BS16, BDS15]. Existing pri-

vacy proofs are often modelled separately to verifiability [BCG+18a, HC19], and may include additional simplifications (e.g., omitting the voter verification [HC19]).

1.5 Contributions

1.5.1 Contributions presented in this thesis

In this thesis, we present the following contributions.

Systematization of knowledge: Mechanisms used in practice While many mechanisms for privacy-preserving and verifiable internet elections have been proposed, we are interested in which security mechanisms are currently actually used by vendors. We perform a systematic market study and review academic literature to find systems used in practice. This leads to the collection of 82 candidate systems, and from those we isolate 29 internet voting systems that are both in active use and claim to employ some form of verifiability. Thereof, we characterize and systematize the 18 systems that provide sufficient information to extract their security mechanisms.

Our work documents that most systems which provide internet voting are effectively black-box systems of which their inner workings are unknown, which notably includes systems that claim some form of verifiability. Further, the few more transparent systems sometimes exhibit a significant gap in what mechanisms they use, and what can be considered the state-of-the-art (e.g., distributed key generation is used by only around half of the systems). Out of the 18 systems analysed in detail, we consider 8 to be well-documented, and these systems also overall implement state-of-the-art mechanisms. We detail our method and our results in Chapter 2. This work has been published at E-Vote-ID 2024 as joint work with Michael Kirsten and Felix Dörre [MKD24].

Short Voting Codes The Swiss Chancellery has developed highly demanding requirements in terms of security for electronic voting. Interestingly, the trust assumptions concerning the voting device are somewhat inconsistent: While for verifiability, the voting device cannot be assumed trustworthy, it is trusted for privacy. We intend to get rid of this trust assumption, and therefore safeguard vote privacy even against a dishonest voting device.

We achieve this using code voting, while we employ the mechanisms in such a way that the codes are very short (typically one or two digits), fit to be entered by hand by the voter. Moreover, the protocol remains simple — notably only relies on standard primitives — especially when compared to other protocols in the same setting [HKD23, HKLD17, Swi24c]. We detail the protocol in Chapter 3, and prove it secure in Chapter 4. This work has

been published at ESORICS 2024 as joint work with Veronique Cortier and Alexandre Debant [CDM24].

Proof Framework To prove verifiability, recent work has modelled an extensible election setting into which a concrete protocol can be plugged into [CCD23]. This framework further provides utilities for a successful proof, appropriately balancing termination and precision. However, it does not support proving privacy, which needs a fundamentally different proof approach (equivalence instead of reachability), and usually relies on different modelling techniques.

We build upon this work, and propose a framework for both verifiability and privacy. It models an election setting with an unbounded number of concurrent elections, voters and votes. Further, it provides a library of lemmas to balance termination and precision of the model. Crucially, the framework allows to reuse the same protocol model for both privacy and verifiability proofs. Further, it is backwards-compatible to the models developed in the verifiability framework. We apply the proof framework to several protocols of the literature and industry, including Belenios, CHVote, Swiss Post Voting System, and the Short Voting Codes proposal. We detail the framework and the case studies in Chapter 4. This work has been published at CSF 2026 as joint work with Vincent Cheval, Veronique Cortier and Alexandre Debant [CCDM26].

1.5.2 Contributions not part of this thesis

Other contributions we chose not to present in this manuscript are the following.

BSI StuVe Commissioned by the BSI, the Federal Office for Information Security of Germany, the study explores the current state-of-the-art of mechanisms used in verifiable internet voting. Based on a literature and market study[5], eight mechanisms used in verifiably internet voting were selected by the BSI to be evaluated. The selected mechanisms include malleable public-key encryption, malleable commitments, homomorphic aggregation, verifiable mixing networks, digital signatures, audit-or-cast, cast-then-audit and return codes.

The study allows to compare these mechanisms to each other, based on clearly defined criteria across secrecy, verifiability, usability and practicality. We present variants of the mechanisms if they reach different trade-offs between these properties. For secrecy and verifiability, we gather trust

[5]The market study has been integrated into the systemization of knowledge of mechanisms used in practice which is presented as part of this thesis in Chapter 2.

assumptions necessary for the guarantees to establish. For secrecy, we additionally evaluate whether the mechanism has some interesting property towards receipt freeness or everlasting privacy. To evaluate usability, we perform an expert evaluation of whether mechanisms are easy to perform for the voter, which we consider the case if the mechanism is both *efficient* (few steps need to be executed) and *effective* (these steps are easy to perform). To evaluate practicality, we highlight *implementability* (difficulty of development) and *efficiency* (computational and communication costs).

The study concludes that according to the current state of the art, it is practical to implement verifiable and secrecy-preserving tallying, coupled with a verification of the voter's device. Further, it highlights that when evaluating a voting system and its trust assumptions, the system must be evaluated as a whole, not just the mechanisms in isolation. Lastly, it documents that research already identified approaches to preserve secrecy during the verifiable tallying procedure against quantum adversaries.

The study has been published by the BSI in 2024 [MMC+24], and is joint work with Johannes Müller, Véronique Cortier, Alexandre Debant, Pierrick Gaudry, Anselme Goetschmann, Ralf Küsters and Melanie Volkamer. Further, a short paper referring to this study has been published at E-Vote ID 2025, to grant the study more visibility while highlighting its strengths and weaknesses [CDK+25], as joint work with Veronique Cortier, Alexandre Debant, Ralf Küsters, Johannes Müller and Melanie Volkamer. The study also lead to the publication of a guide for election organizers on how to organize verifiable elections, authored and published by the BSI [Fed25b], but with the initial draft prepared as part of the BSI StuVe project. The project that involved the study and the draft of the guide has been co-lead by me and Johannes Müller.

Recommendations for Implementing Independent Individual Verifiability The cast-then-audit mechanism lets the voter audit the ballots cast by their voting device. This audit needs to be performed on another device, independent of the potentially manipulated voter's device. Further trust is gained by executing the audit using independent implementations, in order to defend against a dishonest primary system operator. I have implemented such an independent implementation, published at famoser/polyas-verification, for the elections of the Gesellschaft für Informatik e.V. that uses the POLYAS system [Tru23]. This implementation has been provided to voters in the yearly elections since 2023, besides other implementations. To our knowledge, this has been the first project of its kind that provides independent cast-as-intended verification to voters.

Based on the experiences made in 2023, we formulate recommendations for future projects that intend to implement independent verification tools

[MGH+24]. The work recalls the context of the project, and then formulates concrete recommendations across implementation, testing, quality and responsibility. Each recommendation is contextualized with corresponding experiences made during the project. These recommendations have been published at E-Vote ID 2023, as joint work with Rüdiger Grimm, Tobias Hilt, Michael Kirsten, Christoph Niederbudde and Melanie Volkamer.

Development and Expert Evaluation of an Informative Video The continued use of unverifiable black-box systems suggests that election organizers may be unaware of the security challenges in internet voting and the mitigation strategies that have been developed. To address this gap, we developed an informative video on the topic for election organizers who are non-experts in internet voting. In this video, we explain the general challenges and solutions, while also emphasizing that there is no single best approach, with each having their individual advantages and disadvantages. As a running example, we use a code voting system designed to be simple to explain and illustrate. We simplify the topic in such a way that it is understandable to non-experts, while ensuring that this simplification does not lead to misunderstandings or even incorrect statements.

We document the method to create the video, and let 19 German-speaking internet voting experts evaluate the video [HMMV25]. Based on their feedback, we consider improvements to the video to enhance its correctness, clarity, and completeness. Further, developing the video and then performing the expert evaluation provided valuable lessons learned, for example that much of the (but not all) feedback could have already been resolved based on only the textual script. This work has been published at E-Vote-ID 2025, receiving the Distinguished Paper award, as a joint work together with Tobias Hilt and Melanie Volkamer.

Swiss Post Voting System concurrency bug The Swiss Post Voting System is a system developed in the Swiss setting. The code of the system is public, and Swiss Post invites reviews and responsible disclosure of vulnerabilities as part of a bug bounty program [Swi25e].

Upon inspection of the code, we found a bug that allowed to run algorithms more often than they were specified to run, e.g. to avoid storing multiple ballots per voter or brute-forcing secrets. Concretely, for each request received over the internet, the internet-facing (untrusted) voting server prepares a command with the content of the request, and sends them to the control components [Swi25b] (of which one is assumed trusted). The control component then checks whether some semantically identical command has already run, and only if not, executes it. However, the check whether some command has already run included a value set by the (untrusted) voting server, which could therefore make an honest control component run a

semantically equivalent command arbitrarily often. We confirmed the bug by implementing a corresponding unit test in the code base of Swiss Post.

When investigating the impact of the bug, from six algorithms that were potentially impacted, we explored two concrete potential exploits in detail, which however both did not lead to a successful attack:

- `partialDecryptPCC`: This function stores the voter's ballot, and starts the return code generation. Invoking it multiple times has the potential to store multiple ballots per voter. The bug could not be abused due to a primary key constraint in the database, which prevented storing multiple ballots per voter.
- `createLVCCShare`: This function checks the confirmation code for correctness, with an upper limit of how many times this process can be repeated per voter. Invoking it multiple times has the potential to try more confirmation codes for correctness than configured. The number of attempts is stored in the database, with versioning managed by the database abstraction layer (`@VersionId` attribute managed by hibernate [Hib25]). The versioning prevented the exploit, as when storing (i.e., overwriting) the number of attempts from concurrent executions, the implementation threw a `StaleStateException` in all but the first execution to finish.

The bug has been responsibly disclosed, and was classified as severity high. It was fixed within a week, and Swiss Post payed out a bounty of EUR 10'000. More details about the bug and the potential exploits can be found on gitlab.com:swisspost-evoting/e-voting/e-voting#9.

1.5.3 Software contributions

Further, our work lead to some practical results. We have used ProVerif extensively, and proposed and implemented some improvements to it or to tooling around it. Further, while implementing the individual verification for the POLYAS system, we discovered and resolved issues in PHP.

ProVerif improvements Some superficial improvements to the grammar make it more permissive (#aa34a4c9, #3e9e6af7, #f03b9136), and thereby remove some annoyances where semicolons and dots are allowed (or not) in the models. Further, we have added a command line flag *-parse-only* to parse a model, but not directly start executing it (#a6a4f059, #d2776e8a).[6] As a less superficial change, we helped to develop an improvement to debug equivalence-type proofs: whenever ProVerif is not able to proof equivalence, it now outputs additional information about why the proof fails

[6] Used notably in the VSCode language server, described in the next paragraph.

(#f03b9136).[7] Finally, we have implemented an improvement which makes lemmas more expressive, which we will detail in Chapter 4. All these improvements are already on the master branch, hence included in the next releases of ProVerif.

ProVerif tooling While working on ProVerif, we developed tooling to make our workflow more efficient and safe. A major effort was the development of a ProVerif language server for VSCode, which adds semantic highlighting, navigation and safe renames (see ProVerif/vscode-proverif-language-service). The extension is generally available [Pro25a, Pro25b]. To diagnose non-termination issues, we developed famoser/proverif-terminator which is able to diagnose simple non-termination issues, and selectively print the ProVerif output. As part of the framework, we further developed a testing framework, which proved to be useful both to efficiently test iterations of the framework, as well as to document the model runs.

PHP bugfixes As part of implementing the individual verification for the elections of the Gesellschaft für Informatik e.V. using the POLYAS system [Tru23], bugs in PHP were uncovered, the language chosen for implementation. We actively helped in diagnosing and resolving the bugs, as well as adding proper unit testing to PHP to prevent similar issues from re-occurring. We describe the bugs and our work in Section A.1. PHP is the most widely used server-side language.[8]

[7] Under the hood, ProVerif reasons about clauses. If some clause leads to a breaking equivalence proof, the change ensures that ProVerif outputs how it has derived that clause.

[8] See e.g. the statistics of w3tech.

Chapter 2

Mechanisms used in practice

In the context of internet voting, many mechanisms to construct secure systems have been proposed (e.g., see surveys [BBH+17, FDL07, JMP13]). They guard ballots secret (e.g., using asymmetric encryption or commitments), guarantee that the received ballots are privacy-preserving and correctly tallied (e.g., homomorphic aggregation [CGS97] and verifiable shuffles [TW10, BG12]) and ensure ballots have been formed correctly (e.g., audit-or-cast [Ben06], cast-then-audit [Val22, MT23], and return codes [Swi24c, HKLD17]). A full system then composes appropriately these mechanisms for security and usability in its specific election context.

While there are some systems provided directly out of academia and that are free to use (e.g., Belenios [Bel24b] and Helios [Hel24]), most elections are run on commercial industry systems, which are able to offer support contracts and other services necessary for most real-world elections. Our primary interest in this chapter is to find out which security mechanisms these current industry systems actually employ, to shed light on potential gaps in between industry and academia.

Contributions Our contribution consists in a systematization of knowledge of the current implementations of verifiable internet voting systems used in practice, by classifying their mechanisms based on state-of-the-art criteria from academia. Specifically, our contribution is the following:

1. a market study and an academic literature study identifying 82 different internet voting systems used in practice,
2. a systematization of security mechanisms suitable to characterize state-of-the-art internet voting systems,
3. a mapping of our systematization to the 18 voting systems sufficiently documented for our assessment.

Related publication This work has been published at E-Vote-ID 2024 as joint work with Michael Kirsten and Felix Dörre [MKD24]. This work was done in parallel to the BSI study [MMC+24], and the works have influenced each other.

2.1 Method

For this work, we aim to study verifiable internet voting systems in active use. In a first step, we discover candidate systems, trading recall for precision. In a second step, we reduce the list to systems that meet more precise selection criteria, as described in Section 2.1.1. Using only publicly available material, we review each system in detail and assess the implemented mechanisms, as described in Section 2.1.2.

2.1.1 Discovery of Systems

To discover practical systems in active use, we emulate an organization who wishes to provide verifiable internet voting to their electorate. We assume that the organization has no specific election competencies, however, we assume some intrinsic motivation to do well by spending dedicated effort to learn about internet voting for choosing a system that meets appropriate security criteria. For this matter, we assume a high-level understanding of internet voting, i.e., knowledge about some relevant security requirements, e.g., "end-to-end verifiability" and related mechanisms, e.g., "mixnets". However, we do not assume any corresponding technical expertise or prior knowledge about existing systems. Assuming this coarse-grained understanding of secure internet voting, we perform a market study, with the (hypothetical) aim of choosing a secure state-of-the-art system.

To replicate this setting, we let a business consultancy perform the market study, with experience in market studies, but without prior knowledge in internet voting, thereby ensuring an unbiased perspective. The consultancy received a brief (less than one hour) introduction on the general process of holding (verifiable) elections, guarantees generally desired in such systems, and an overview of some of the mechanisms usually employed to reach these guarantees. We consider this sufficient to give a coarse-grained understanding, but not more than what we would expect from motivated but non-technical election organizers.

To verify the quality of the market study, and minimize chances of ignoring relevant systems, we complement the market study by gathering systems referenced in academic literature (e.g., previous comparisons or system proposals). Further, we cross-check the resulting list of systems with two other market analyzes and one list of systems maintained by the Inria Nancy research team PESTO. Throughout this process, we prioritize recall over prevision, which however leads to discovering many products that are not relevant to our work. Hence, in a final step, we filter the *longlist* of discovered systems for a *shortlist* of systems that meet our more precise criterion of being actively used for verifiable and private internet voting. In Table 2.1, we give an overview on how many systems were added by which source.

Source	Longlist	Shortlist
Our Market Study	60	17
Our Literature Study	+12	+5
Analysis on French Market	+6	+5
Analysis on German Market	+2	+1
PESTO (INRIA Nancy) List	+2	+1
Total	**82**	**29**

Table 2.1: We report how many systems on the *longlist* and the *shortlist* were found by our Market Study, and how many were added using additional sources. We found a total of 82 systems broadly related to verifiable internet voting. Filtering for our criteria, i.e., internet voting, end-to-end verifiability and privacy, and in recent use, leaves a *shortlist* of 29 systems for further analysis.

Market study To discover products on the market, the business consultancy searched for products that are broadly related to online voting, thereby aiming for breadth over depth. They conducted the search primarily in January 2024 by using Google, Bing, and Yandex, as well as relevant Wikipedia articles and LinkedIn, with native language proficiency in English and German. In their search, the consultancy used keywords related to the general topic (e.g., "election", "voting"), mechanisms (e.g., "zero-knowledge proofs", "mixnet"), and security properties (e.g., "end-to-end verifiability", "privacy"). For cross-checking intermediate results, they consulted ChatGPT 4.0 with a dataset from around January 2022. In total, the consultancy identified between 250 and 350 products that fit one or more of the keywords. Within the discovery phase, the consultancy employed superficial judgement of the providers' marketing and immediately discarded products not directly related to *verifiable* internet voting. We included the remaining 60 products on our *longlist*.

Literature study To discover products referenced in academic literature, we ran a manual search, primarily in October 2023, of online-accessible databases and webpages for all proceedings (including relevant satellite workshops) since 2016 of the most relevant academic venues.[1] Moreover, we included the preprint server *IACR Cryptology ePrint Archive* on which many academics publish technical reports or preprint versions of their submissions in cryptography- or security-related domains. In our search, we used the general keywords "voting" and "election", and then scanned the titles for any of the keywords "voting system", "voting protocol", "voting

[1]These venues are the *International Joint Conference on Electronic Voting* (E-Vote-ID), the *ACM Conference on Computer and Communications Security* (CCS), the *IEEE Computer Security Foundations Symposium* (CSF), the *European Symposium on Research in Computer Security* (ESORICS), the *International Conference on Financial Cryptography and Data Security* (FC), and the *IEEE Symposium on Security and Privacy* (SP).

scheme", "internet voting", "remote voting", "electronic voting", "e-voting", "electronic election", "verifiable", "verifiability", "review", "study", or "survey". We then did a semantic scan by manually reading all titles and abstracts for whether the publication mentions any verifiable internet voting system, which comprises papers presenting specific systems of that nature, as well as surveys, evaluations, analyzes or employments. Moreover, we extended our search by scanning the papers' references from their introduction, related work section, and evaluation sections, for the same criteria.

Our semantic scan aimed to identify systems with recent (i.e., within the last two years) usage in real-world internet elections, i.e., we excluded any kind of in-place systems. Our criterion of real-world comprises any election with decision power on the level of national governments, states or cantons, districts, municipalities, companies, associations or organizations, institutions (e.g., universities), or political parties. We disregarded systems used purely for demonstration purposes or scientific papers, as well as clearly abandoned systems. Collecting all systems proposed in academia would include many more systems, but is out of scope for our purpose. In total, we found 22 systems and, after deduplication and consultation of the systems' webpages and manuals, we added 12 systems to our longlist.

Cross-check with other system collections. To ensure that our list of systems does not contain significant gaps, we did a cross-check with the following two market studies and one list of systems:[2]

- market analysis focused on French market leaders in internet voting (9 systems, 2023)
- market analysis focused on the German market of internet voting (63 systems, 2022)
- list of systems maintained by the Inria Nancy research team PESTO focused on end-to-end verifiable systems (12 systems, continuously updated)

The added systems from the first analysis only provide information in French, and were hence hard to discover in our market study performed in English and German. From the second analysis, we directly discarded systems which provide only simple polls or conference voting, as this is not in the scope of our work. In total, after deduplication, we added 6 systems from the analysis on the French market, 2 systems from the analysis on the German market, and 1 system from the list of PESTO (Inria Nancy) (see Table 2.1).

[2]Made internally available as part of a study on E2E-verifiability for the Federal Office for Information Security.

Result The discovery phase resulted in a longlist with 82 systems (see Table 2.1), for which we collected general facts, i.e., vendor, location and webpage. We further constructed a *shortlist*, for which we reduced the longlist's systems in order to fulfill the following criteria:

- Real internet voting systems. The system allows casting votes over the internet. This excludes, e.g., election management software, or systems which require a paper trail where the voter sends a receipt of their vote to the election authorities after voting.
- Aim for end-to-end verifiability and some form of privacy. We welcome different jargon, as long as the overall goal promises some form of end-to-end verifiability and privacy. Further, we admit claimed compliance to some electronic-voting-related certification (e.g., to the French CNIL [Com19b, Com19c] or the Common Criteria Protection Profiles published by the German BSI [VV08, IT 23]), while we deem more generic certifications such as ISO 27001 as insufficient.
- Active or recent use. We check the proxy criterion that some official communication (e.g., on LinkedIn), code or documentation was produced within the last two years.

Based on these criteria, we found that 9 systems do not provide internet voting, 15 systems already vanished (e.g., dead startups), and 29 systems have no claims related to verifiability. This resulted in a shortlist that leaves 29 systems which fulfill all our criteria, and which we aim to analyze in detail. We give an overview of which source contributed how many systems to the longlist and the shortlist in Table 2.1. The full longlist with all systems and their respective criteria evaluation is publicly available.[3]

Discussion We observe that the market study alone already covers a large share of the systems on both lists (see Table 2.1). We therefore expect our findings to represent the market well, by having found the most relevant systems used in practice. Further, more than 80 percent of the systems in the market study's longlist differ from the systems in the literature study. This indicates that the precise inner workings of many systems used in practice are not published academically, and that these systems are also not considered when comparing systems in scientific publications.

For transparency, we report on indicators of gaps on our lists in the following. In our hypothetical scenario, we do not consider hard-to-find systems as explicit gaps, as the organization emulated in our market study would also fail to learn of the system. Yet, we expect gaps for systems that are neither advertised in English nor in German, as we observed in our cross-check

[3]The list is available as supplementary material on inria.hal.science.

for the French market. Further, our market study may exclude companies not explicitly advertising verifiability, but who only use it under the hood.

2.1.2 Extraction of Mechanisms from Systems

All the systems on our shortlist (see Section 2.1.1) claim some level of verifiability, and as verifiability aims to make a system observable to outsiders, this directly requires that the system's inner workings are communicated clearly and openly. We would therefore expect all systems to publish at least a detailed system specification, possibly also a reference implementation, security proofs, and expert reviews of their system.

Our goal consists in assessing which mechanisms are used in practice, not in assessing the systems themselves. We can hence relax the thoroughness of our reviews and require less documentation than would be necessary for full system reviews. Our classification targets the most positive and secure interpretation of the described mechanisms, where we favorably fill gaps with educated guesses, and generally trust the claims made by the vendors. Often a high-level overview of the system (e.g., a whitepaper, or explicit claims on the vendor's webpage) is sufficient, if that allows to clearly identify which mechanisms are in use.

However, even in this relaxed setting, many systems do not publicly provide sufficient information for extracting the mechanisms in use. The following list contains all systems on our shortlist for which we could not find sufficient information and which we therefore excluded from further analysis:

- *Electis* (France) [Ele24b]. The code is open source [Ele24a] and seems to integrate with ElectionGuard [BNPW24] and the Tezos blockchain, but it is hard to extract the specific security mechanisms of the currently implemented solution. The product claims to follow the CNIL recommendations for electronic voting [Com19c].
- *Eligo* (Italy) [Eli24]. The published documentation does not disclose security details apart from support for two-factor authentication.
- *Genolive* (Germany) [Con24]. While the product claims to be developed in compliance with the BSI Common Criteria Protection Profile for internet voting [VV08], it did not actually undergo the compliance audit [FOfIS25], and the employed security mechanisms are not described.
- *Kercia Solutions* (France) [Ker24], *LegaVote* (France) [Leg24], *Neovote* (France) [Neo24], *WebVote* by Gedivote (France) [Ged24], and *WeeChooz* (France) [WeC24]. We could not find documentation about the security mechanisms implemented by these systems. All products claim to follow the CNIL recommendations for electronic voting [Com19c].

- *Neuvote* (Canada) [Neu24]. The webpage does not disclose any security details besides support for end-to-end encryption and two-factor authentication.
- *Nvotes* (Spain) [nVo24]. The webpage does not contain any concrete details about their system. However, the technical leadership seems to have moved to Sequent, which we analyze as part of this work (see below).
- *Onlz* (Belgium) [Onl24]. The webpage does not contain details about the system. While it mentions to employ client-side encryption, some blockchain and ring signatures, this is insufficient for our purposes.

For the following systems, we are able to extract the mechanisms from published specifications:

- *Belenios* (France) [Glo24]. Besides the specification, there is a high-level academic publication which discusses some of the design decisions [CGG19]. Further, the code is open source [Bel24a], and there are further supporting documents, including academically reviewed formal proofs [Bel24b].
- *Electa* by AssemblyVoting (Denmark) [Ass23]. Besides the specification, some of the code is public [Ass24].
- *Helios* [Adi08]. Besides the initial academic publication and the specification, which was available until April 2024 [Adi12], the code is open source [Hel24]. Recent development in the repository seems to focus on dependency updates, minor usability features, and authentication mechanisms.
- *IVXV* by Cybernetica (Estonia) [Val22]. Besides the specification, the code is open source [Val24].
- *Polyas Core-3* (Germany) [Tru23]. Besides the system specification, there is an academic paper and a specification for their cast-and-audit approach [MT23, Tru24]. Further, there are some non-technical descriptions on the webpage [POL24].
- *Sequent* (USA) [Seq24a]. There is a webpage with a system overview, documentation [Seq24c] including usage demos, explanations, the high-level cryptographic protocol [Rue21], and system architecture [Rob22], as well as open-source code [Seq24b].
- *Swiss Post Voting System* (Switzerland) [Swi24c, Swi24d]. Besides the specification, extensive documentation of the system, formal proofs, as well as the code is public [Swi24e]. The system is provided for political elections in the Swiss cantons [Swi24b], and is hencecompliant to the corresponding Swiss law [Swi13]. Further, numerous system reviews by independent experts are publicly accessible [Swi24a].

- *uniWAHL* by ElectricPaper (Germany) [Ele24c]. The webpage focuses on explanations and demos and is mostly aimed at election organizers. Under the hood, they use the security mechanisms provided by Sequent [Seq22].

Not all systems we analysed in this work published detailed specification such as the systems mentioned before. As long as we managed to assess the mechanisms used, this was however not a reason for exclusion. Concretely, for the following systems, we use informal partial specifications, whitepapers, code, or webpages:

- *BigPulse Voting* (UK) [Big24b]. The verification is sketched on the webpage [Big24a]. We did not find details about authentication or encryption of the votes.
- *DecentraVote* (Germany) [Dec24a]. Besides the whitepaper [Faz20], the code is licensed as free software [Dec24b].
- *EVoters* by EVoting (Chile) [EVo24a]. The used encryption, as well as the tally mechanism, are described on their webpage [EVo24b]. We could not find details about how votes are verified by the voter or how the votes are stored, and found only very high-level information on the employed authentication.
- *Followmyvote* (USA) [Fol24a]. The webpage provides informal descriptions and some system code is public [Fol24b].[4]
- *Invote* by Scytl (Spain) [Scy24]. The specification is not public, but the mechanisms are sketched in an academic survey paper [FCC24].
- *Simply Voting* (Canada) [Sim24b]. Some of their used mechanisms are documented [Sim24a].
- *V8te* (France) [V8T24]. The privacy policy describes some mechanisms [Cé23].
- *Voatz* (USA) [Voa24]. The webpage features a video which describes some of the security mechanisms and links to a high-level whitepaper [Moo20]. It remains unclear how ballots are encrypted and authenticated.
- *Voxaly* (France) [Vox24]. Other than compliance to the recommendations of CNIL, the webpage does not provide a closer description of the solution. However, a partial specification [Che23] (e.g. excluding authentication) of the system used for the legislative elections in France is public and described as an adaptation of the Belenios system, which we also analyze as part of this work (see above).

[4]The provided code includes smart contracts and some UI code, but notably no code employed by the registrar.

- *zkVoting* (Korea) [PCKO24]. An academic whitepaper describes cryptographic details.

As for these less documented systems, the assessment is less precise, we group them separately in the tables.

Systemization design To design this systematization, we first performed two independent coarse-grained reviews of some of the chosen systems. These initial reviews resulted each in a systematization proposal which captures their respective subset of systems. We then unified the two systematization proposals into our final systematization, in order to capture the full diversity of the systems.

The systematization aims to consider only mechanisms which are fundamentally different. Hence, we ignore implementation details, and collapse slight variations of a mechanism (small tweaks to achieve, e.g., improved verifiability or secrecy) into a single mechanism. Thereby, we avoid noise in our evaluation, while still distinguishing clearly separate approaches.

2.2 Analysis

We organize the mechanisms into *Secret ballots*, *Verifiable tally*, *Individual verification* and *Authentication*. We organize under *Secret ballots* the encryptions used to form the ballot, and how the corresponding keys are managed. Further, under *Verifiable tally* we track the mechanisms used in the tally phase, to cleanse the list of ballots (e.g., of revotes), anonymize and finally decrypt them. Some systems further provide *Individual verification* to allow the voter to verify that their ballot was formed correctly, and received as such by the system. Lastly, we track *Authentication* mechanisms, which authenticate the voter to the system, or artifacts emitted by the system.

2.2.1 Secret ballots

In the voting phase, the voter forms their ballot. We track the employed encryption and the management of the corresponding (decryption) keys. An overview of the identified mechanisms and whether the systems implement them is provided in Table 2.2.

Encryption We distinguish whether the ballot hides the vote using *symmetric encryption*, *asymmetric encryption* or *committments*. In the internet voting setting, asymmetric encryption and committments are desirable, as they are a precondition to employ any of the privacy-preserving tally methods we identified (see Section 2.2.2).

Keys We place special attention to how encryption keys are handled. Storing the decryption key in a *distributed storage* requires multiple trustees to reconstruct the decryption key. Further, to prevent that a single trustee learns the decryption key when it is generated, we track support for *distributed generation*.

	Belenios	Electa	uniWAHL	Helios	IVXV	Polyas	Sequent	Swiss Post	BigPulse	DecentraVote	followmyvote	Invote	SimplyVoting	V8te	Voatz	EVoters	Voxaly	zkVoting
Encryption																		
symmetric										●								●
asymmetric	●	●	●	●	●	●	●	●				●		●		●	●	
committments																		
Keys																		
distr. storage	●	●	●	●	●	●	●	●				●		●		●	●	
distr. generation	●	●	●	●		●	●	●										

Table 2.2: Comparison of voting systems concerning their mechanism used for encryption and keys, well-documented systems to the left of the dashed line. DecentraVote uses a hash to hide the plain vote, taken from the plain vote concatenated with secrets, with the secrets published in the tally phase to count the votes. zkVoting uses a hybrid encryption scheme, with the ballot itself encrypted under a symmetric key, and that symmetric key under the public election key. For BigPulse, followvoting, Simply Voting and Voatz, we did not find claims about the used encryption.

2.2.2 Verifiable tally

After the voting phase closes, the system will calculate the result. First, the system may perform a cleansing to, e.g., filter out revotes. Then, ballots are anonymized, i.e., the link to the submitting user is removed, and decrypted. We track in Table 2.3 the identified mechanisms and the systems that implement them.

Cleansing Some systems support *revote* policies, where voters may vote multiple times and the latest ballot overrides the previous ones. Further, voters may be entitled to *revoke ballots*, e.g., in case a voter later-on decides to nullify their online ballot to cast in person instead. Further, voters may also cast *fake ballots*, that is ballots cast with invalid credentials that do not enter the tally. This may enable a voter to pretend towards a coercer they have voted in a certain fashion.

	Belenios	Electa	uniWAHL	Helios	IVXV	Polyas	Sequent	Swiss Post	BigPulse	DecentraVote	followmyvote	Invote	SimplyVoting	V8te	Voatz	EVoters	Voxaly	zkVoting
Cleansing																		
revotes	●	●	●	●	●		●		●						●			
revoked ballots					●	●												
fake ballots																		●
Anonymization																		
homomorphic aggr.	●			●								●		●		●	●	◐
verifiable shuffle	●	●	●		●	●	●	●				●						
Decryption																		
verifiable	●	●	●	●	●	●	●	●		●		●					●	●
distributed	●	●	●	●		●	●	●										

Table 2.3: Comparison of voting systems on their tally mechanism, well-documented systems to the left of the dashed line. In zkVoting, a single authority decrypts the (hybridly-encrypted) ballots, sums up the plaintexts, and then publishes the sum with a proof that it corresponds to the authenticated ballots.

Anonymization Before decryption, the ballots need to be anonymize, i.e., the link to the submitting voter needs to be removed. *Homomorphic aggregation* allows aggregating all ciphertexts into a single ciphertext which sums up the chosen candidates from all its contributing ciphertext [CF85, Ben87]. Then, only this resulting ciphertext is decrypted. However, homomorphic aggregation needs to be adapted to each counting function, which is hard or inefficient for some counting functions [HKLR24]. This is one of the reasons a system may use a *verifiable shuffle* (e.g., [BG12, TW10]). The shuffle preserves an individual ciphertext per ballot, while avoiding attribution to the original cast ciphertext using verifiable re-encryption and secret shuffling. After decryption of all the individual ciphertext, the counting function can operate on plain votes, as in classic elections.

Decryption Decryption may be *verifiable*, hence the trustee performing the decryption proves that the plaintext resulting from their decryption corresponds to the ciphertext. Additionally, decryption may be *distributed*, where multiple trustees never share their (partial) decryption keys among the other trustees. This allows each trustee to remain in control over which ciphertext is decrypted, e.g., to enforce in case of that homomorphic decryption is used that only the single resulting ciphertext is decrypted.

	Belenios	Electa	uniWAHL	Helios	IVXV	Polyas	Sequent	Swiss Post	BigPulse	DecentraVote	followmyvote	Invote	SimplyVoting	V8te	Voatz	EVoters	Voxaly	zkVoting
audit-or-cast		●	●	●			●											
cast-then-audit					●	●						●						●
return codes								●										
clear-text receipts													●		●			
tracking codes						●			●				●		●			

Table 2.4: Comparison of voting systems on their cast verification mechanism, well-documented systems to the left of the dashed line. Some systems do not employ cast verification, notably Belenios, and the Voxaly system which adapted Belenios. Audit codes are sometimes combined with other mechanisms.

2.2.3 Individual verification

When casting the vote, systems often allow verifying whether the ballot has been formed correctly. These verification mechanisms may deliver either probabilistic or definite guarantees. In the former, the voter gets probabilistic guarantees over whether their ballot is correct, which get stronger when the voter repeats the verification multiple times. In the latter, if the verification succeeds, the ballot represents only with negligible probability something different. In Table 2.4, the identified mechanisms and the corresponding systems are visulaized.

Verification We observe the probabilistic *audit-or-cast* approach in actual use, where after the ballot is formed, the voter is given the choice to either audit or cast the ballot. If the voter chooses the former, they may verify whether the ballot was formed correctly, but must restart the cast procedure until they finally choose to cast the ballot. If the voter chooses to cast the ballot, they need to check that the ballot reached the voting system exactly as formed.

Some systems allow auditing the cast vote, which we name *cast-then-audit* approach. Such systems may take precautions attempting to avoid that the voter gets a trivial receipt, e.g., limiting the time during which the voter can audit or keeping the bulletin board private. We name the mechanism *clear-text receipt* if the voter gets some confirmation which includes the plaintext vote. If the voter receives opaque codes that correspond to their vote in the voting phase, we name it *return codes*. We name it *tracking codes* when the voter may look up the content of their anonymized vote using a code.

2.2.4 Authentication

Most systems document that the voter needs to pass one or multiple logins, similar to how the voter would interact with other online systems. We are however interested in whether and how the system employs credentials, which we understand as cryptographic keys with a private and a public part, necessary for state-of-the-art cryptographic ballot authentication.

Further, once the authenticated vote is cast, the system may provide an artifact to the user that the ballot has really been stored. This artifact may additionally be authenticated to avoid disputes over the artifacts' validity, and chained, to harden against later changes to the system state.

2.2.4.1 Voter Authentication

We document in Table 2.5 which system uses which mechanisms concerning voter authentication.

	Belenios	Electa	uniWAHL	Helios	IVXV	Polyas	Sequent	Swiss Post	BigPulse	DecentraVote	followmyvote	Invote	SimplyVoting	V8te	Voatz	EVoters	Voxaly	zkVoting
Cred. Generation																		
centralized	●					●		●				●						
distributed		●																
on voter's device		●								●	●							●
elec.-independent					●													
Cred. Interface																		
directly entered	●	●				●		●				●						
trusted hardware					●													
Cred. Properties																		
cryptogr. bound	●	●			●	●		●		●		●						●
distributed auth.		●																

Table 2.5: Comparison of voting systems on how credentials are generated and used, well-documented systems to the left of the dashed line. Electa supports credential-based authentication, where credentials are distributed by credential authorities and entered by the voter, and identity-based authentication, where the voter's device generates the credential, which the voter authenticates using identity providers.

Credential generation Credentials may be generated *centralized*, where a single authority learns all credentials, or *distributed* by multiple authorities. Some systems may generate the credentials *on the voter's device*, and others rely on *election-independent* credentials established outside the specific election context, e.g, for other uses such as electronic IDs.

Credential interface For credentials that were not already generated on the voter's device, we distinguish the interface for the credentials' usage. Credentials may be explicitly delivered to the voter on a dedicated channel, where the credential is then *directly entered* by the voter into the voter's device, possibly looking similar to entering a long password. Another approach stores the credential on *trusted hardware*, such as an electronic ID card, to ensure only the hardware holder is able to use the credential.

Credential properties Most systems use credentials that are *cryptographically bound* to the ballot, to harden against a dishonest voter attempting to (re)use the authentication for a different ballot. Systems may further use *distributed authentication*, where they rely on multiple authorities who authenticate the public part of the credential.

2.2.4.2 Receipt Authentication

We document in Table 2.6 which system uses which approach concerning the receipt provided to the voter.

	Belenios	Electa	uniWAHL	Helios	IVXV	Polyas	Sequent	Swiss Post	BigPulse	DecentraVote	followmyvote	Invote	SimplyVoting	V8te	Voatz	EVoters	Voxaly	zkVoting
User Artifact																		
storage ref		●							●	●	●	●		●	●			●
ballot hash	●		●	●	●	●	●										●	
plain vote													●					
return code								●										
Artifact Properties																		
signed		●			●	●			●								●	
chained										●	●				●			●

Table 2.6: Comparison of voting systems on their receipt authentication mechanism, well-documented systems to the left of the dashed line.

User artifact The voter may receive a *storage reference*, which upon querying the system reveals the stored ballot (e.g., a transaction ID in a blockchain). Otherwise, the voter may receive the *ballot hash*, or even the *plain vote* itself as artifact. Some systems also provide a *return code*, which asserts that some, possibly previously verified, ballot has been stored.

Artifact properties The artifact that the voter receives may be authenticated, i.e., *signed* using a digital signature, which allows proving that the artifact really originated from the system. This makes the system accountable for the artifacts that it issues. Further, the artifact may cryptographically reference the ballots already cast by other voters, which we then name *chained*. This forces the system to commit to the ballots it already stores at the time of issuing the artifact, which hardens against later tampering with that state.

2.3 Related work

Comparing voting systems based on requirements from academia is challenging. There is no generally agreed-upon list of required properties with fixed definitions, nor what acceptable trust assumptions for these properties are. Li et al. [LKZ14] systematize 2014's state of the art for 14 electronic voting systems, 9 of them suitable for internet voting, both for cryptographic mechanisms and for satisfied properties. However, they do not report on how they choose the systems, and by now, many are not in active use anymore. More recently, Finogina et al. [FCC24] select twelve internet voting systems that they deem well-known. Then, they compare the properties they reach (and under which trust assumptions), and track core functionality (e.g., voting channel flexibility, support for digital governmental identity). While proposing a unified convention and systematization, the selection of systems is somewhat arbitrary, and it is unclear how their trust assumptions and property interpretations can be generalized to apply to a wider selection of systems.

Compared to this work, our selection process is clearly defined, and happens to discover many more systems: Our initial list comprises as 82 internet voting systems that are broadly related to verifiable internet voting. We then aim to analyse as many systems as possible, and choose a rather superficial evaluation approach that focuses on the employed mechanisms. With this approach, and with the generous interpretation bound to believe the claims of the vendors, 18 systems then provided sufficient documentation to be included in this analysis.

Limitations Our chosen approach has some limitations, which we want to discuss here. We do not verify whether the mechanisms are actually

implemented correctly, nor whether their composition reaches any tangible security guarantees. Further, we do not consider variants of the mechanisms, which may establish different trade-offs between verifiability and privacy.[5] This is a direct consequence of the rather superficial evaluation approach chosen. To overcome these limitations, additional documentation from all systems would be needed, which would however constrain the analysis to even fewer systems.

2.4 Discussion

We have performed a systematic market study and reviewed academic literature to find systems used in practice. Out of 82 candidate systems, we distil a list of 29 internet voting systems that are in active use, and are presented by the vendors as privacy-preserving verifiable internet voting systems. We then characterized and systematized the mechanisms used by the 18 systems that provide sufficient information to do so. Our work documents that most systems are effectively black-box systems, which notably includes systems that claim some form of verifiability. Further, from the systems we were able to analyse, many exhibit a significant gap in what mechanisms they use, and what can be considered the state-of-the-art (e.g., distributed key generation is used by only around half of the systems). However, we have also found 8 systems we consider well-documented, and which overall implement state-of-the-art mechanisms.

We now sketch avenues for future research.

Investigate why many systems do not aim for verifiability Almost all systems use security as a sales argument, largely independently of the actually taken (and publicly documented) security measures. However, many systems do not even claim to implement verifiability in their products, and out of those that do, most systems document their solution poorly. As an example, from the systems claiming to implement the CNIL e-voting recommendations [Com19c], six out of nine systems did not publish concrete documentation, and two published only a very partial description. Already for other secure systems, such a lack of documentation would be surprising, considering the security principle of *open design* [SJT08]. In the context of internet voting, however, verifiability without transparency is not achievable. Transparency is therefore a precondition for any system that can be deemed verifiable, and verifiability is a precondition for any secure election system.

[5] For example, our evaluation treats the Estonian [Val22] cast-then-audit approach equal to the POLYAS [POL24] approach using CAISED [MT23], while the later aims to improve towards receipt-freeness.

Future work could try to investigate this gap in between what academia considers a basic precondition for secure systems, and why many vendors do not provide it. Are vendors simply not aware of it, or are they deliberately choosing not to provide it? We observe in our initial list many small and local products that may not have dedicated expertise in internet voting in-house. However, also many products that seem to have a large customer base do not claim verifiability, which also seems to indicate that their customers, the election organizers, do not know about this. How can they be appropriately informed? Or is this the responsibility of the regulator, and therefore how can regulations (such as the CNIL recommendations) be formulated to establish an appropriate baseline, without overboarding compliant systems with bureaucracy, or arbitrarily restricting the solution space?

Distil sensible compositions of mechanisms When evaluating the mechanisms, most of the eight sufficiently documented systems do not implement state-of-the-art security mechanisms in all relevant categories. While asymmetric encryption is typically used to encrypt the vote, only seven systems combine it with distributed generation and distributed storage of the corresponding private key. Similarly, while homomorphic aggregation or verifiable shuffling are used in practice, only the same seven systems then perform the decryption in a verifiable and distributed fashion. All other systems omit at least the distributed generation and the distributed decryption, making the component which learns the full decryption key a potential single point of failure. While the state-of-the-art for casting and tallying seems mostly established, and most systems use the same mechanisms, many different approaches are in use for authentication, cast verification and recording, generally with no clearly emerging favourite. Notably for authentication, some systems support multiple mechanisms and let the election organizer pick the approach that suits their election context best.

The generation of a distributed election key, the encryption of the vote towards this key, and then the use of a privacy-preserving tallying method seems to be a universal composition that should be used by any state-of-the-art system. In contrast, versatility depending on the election context can be expected in the cast-as-intended techniques as well as the authentication, and as we observe this versatility in practice, it shows that being able to compose systems adapted to the specific election context is vital. However, the full guarantees of the system only establish if the overall combination of mechanisms is appropriate. Unfortunately, analysing the security of a specific composition is currently a high-effort task, unrealistic to be executed for each specific election context. Can we reduce the cost of this analysis, for example by providing pre-made proof structures (e.g., see Chapter 4)? Or can we define a set of properly verified combinations that are able to cover the large majority of elections?

Improve access to implementations of state-of-the-art mechanisms Besides the large gaps between practice and research reported above, we want to briefly discuss the systems which employ some of the relevant state-of-the-art security mechanisms. The systems Belenios and Helios, both largely developed directly within academia, provide very transparent and extensive documentation: While Helios shows no recent development anymore and is a considerably smaller system with not as much documentation, both systems provide source code and a range of academic publications about the systems and proposed variants, and Belenios even provides formal proofs [Adi08, Bel24b]. Further, some systems developed in industry publish extensive documentations and source code. Most notably, the Swiss Post Voting System provides detailed specification, source code, formal proofs, as well as extensive third-party expert reviews [Swi24e, Swi24a]. These extensive transparency measures are required by law for the Swiss national political elections [Swi13], for which the Swiss Post Voting System is built. Both, the Sequent (and hence also uniWAHL) and the IVXV system also publicly provide many details on their mechanisms and their source code [Seq24a, Val24, Val22]. Moreover, the Electa system provides an easily-accessible and high-quality specification describing their used cryptographic mechanisms [Ass23]. This holds for the Polyas system, too, where system specifications and an academic publication are available [Tru23].

While some systems develop their own cryptography from scratch, this is a very high-effort task and needs dedicated expertise, which is out of reach for most vendors. We have however seen many systems report using the open-source Verificatum [Wik24] and ElectionGuard [BNPW24] under the hood. Provided they are used correctly, they therefore allow a much larger part of the vendors to appropriately secure their systems. However, Verificatum [Wik24] focuses on verifiable shuffles, and ElectionGuard [BNPW24] does not really target internet elections. A future direction may therefore aim to provide vendors, in particular those without in-house cryptography expertise, more practical means to use such algorithms in their systems.

Chapter 3

Short Voting Codes

The Swiss setting is demanding in terms of the properties that need to be reached, and the trust assumptions that are deemed to be acceptable to reach these properties [Swi13]. However, the trust assumptions are somewhat inconsistent concerning the voter's device: While the device is not assumed honest for verifiability, it is assumed trustworthy for vote privacy. In this chapter, we aim to get rid of the trust in the voting device entirely.

To avoid trust in the voting device for privacy, *code voting* has been proposed, introduced originally by Chaum [Cha01]. A voter no longer enters their vote in clear, but a voting code instead. As the voting device cannot attribute the voting code to the candidate it represents, the voting device never learns the vote. The essential insight of this work is that these voting codes can be really short (one digit), making it feasible that voters enter them by hand.

We explore this building block in the multi-race Swiss setting, designing a full protocol complaint to the requirements of the Federal Chancellery. From a voter's point of view, besides the short voting codes, we remain close to the existing systems (i.e., CHVote [HKLD17] and Swiss Post [Swi24c]). Notably, we reuse the return codes to achieve cast-as-intended. From a cryptographic point of view, the proposal is however much simpler.

Our contribution Our main contribution is a voting protocol that satisfies the security requirements of the Swiss Chancellery (vote privacy and verifiability), without assuming a trusted voting device, even for vote privacy. We achieve this using *short* voting codes, fit to be entered by hand by the voter, and therefore applicable to the multi-race Swiss setting.

Interestingly, our protocol is very simple in terms of design and of cryptographic primitives. The core protocol relies only on very standard cryptographic primitives, such as signatures and hashes. Further, we reduce reliance on the fully trusted setup component by making its computations minimal and deterministic, to enable randomized audits checking for correct computation. In Chapter 4, we prove the proposed protocol secure.

Related publication This work has been published at ESORICS 2024 as joint work with Veronique Cortier and Alexandre Debant [CDM24].

3.1 Swiss setting

We have already given an overview the Swiss setting in the introduction. Here, we reintroduce the main actors and the trust assumptions, as described by the Swiss Chancellery [Swi13], in a more precise manner.

The **Setup Component** is active during the setup. It computes the voting material and distributes it to the voters through postal mail. It also prepares the material used to tally the votes.

The **Control Components** are entities that are active during all the election phases. They collectively generate the randomness used during the setup, and register the votes and compute return codes during the voting phase, and process (e.g., shuffle) the encrypted ballots during the tally phase.

The **Decryption Authorities** are another set of control components, in charge of generating the public key of the election and decrypting during the tally.

The **Voting Device** interacts with the **Voter** to help them cast a vote over the Internet.

The voting device does not communicate directly with the control components. Instead, all communications go through a **Voting Server** that is fully untrusted and simply sends the messages back and forth from the voting devices and the control components. Therefore, we will omit its description in what follows.

Trust assumptions The network is fully under the control of the attacker. In particular, the security of the protocol cannot rely on TLS. However, postal mail from the setup component to the voter is assumed to be trusted: the attacker can neither read nor tamper with the voting sheets sent to the voters.

The setup component is assumed to be fully trusted. It is only active during the setup phase and does not need to be online. The voting material (the voting sheets) is sent over secure channels to the voters (in practice, by Post). The control components are typically a set of 4 entities, among which only one is trusted for verifiability and vote privacy. The same holds for the decryption authorities, among which one is trusted for verifiability and privacy. Finally, the voting device is untrusted for verifiability but trusted for privacy in the trust model of the Chancellery. This is the point where our trust assumptions differ: we no longer assume the voting device to be honest for privacy.

Moreover, the Chancellery's model places a lot of trust in the setup component since it is trusted both for privacy and verifiability. This is something that we mitigate in our protocol by proposing a fully deterministic process that is easy to audit.

3.2 Related Work

In the Swiss context, two main protocols have been proposed, concretely CHVote [HKLD17] and the Swiss Post protocol [Swi22]. However, both assume that the voting device is trusted for vote privacy. In the literature, a few systems have been proposed to achieve vote privacy against a dishonest voting device, using code voting, following the seminal work of Chaum [Cha01]. Among them, only one, HKD [HKD23], has been designed specifically for the Swiss context, i.e., with the aim to satisfy the Swiss trust assumptions. Others have been designed in a more general context. Hence, their different trust assumptions led to different design compromises. We discuss here the systems that aim at protecting vote privacy against a dishonest voting device.

Comparison to HKD protocol [HKD23]. This recent proposal [HKD23] aims at the same goal than us, i.e. compliance to the Swiss setting while safeguarding privacy against a dishonest voting device. In the HKD protocol, voters receive a voting sheet in which each voting option is associated to a QR-code, that encodes the encrypted vote. Thanks to a verifiable mixnet during the setup, the relation between the ciphertext and its plain vote remains private for anyone without access to the voting sheet. A QR-code is proposed as the user-interface to enter the voting codes as they are long; notably each containing a BLS signature [BLS01].

One major drawback of this approach is its usability. Indeed, in Switzerland, a voter typically needs to answer about 6 questions, and for several questions, 3 options can be selected (yes, no, abstain). This would amount to selecting 6 QR-codes out of 18 for a typical election. But some elections have more questions (e.g. 13 questions in Zurich in autumn 2022 [Kan22, Sta22], therefore amounting to 39 QR-codes) and some elections even offer voters to select *several* options for each question [Kan]. Moreover, because the camera used to scan the QR-code(s) shall not see the other cryptographic data printed onto the voting sheet for security purposes, the voter is subject to additional constraints when manipulating the voting sheet. Recent studies [VKLF22, MSLM19] conclude that QR-codes are usable for electronic voting when voters select exactly *one* voting option in total, but it seems cumbersome to use such a technology in elections where voters have to select more than a few options.

Compared to HKD, our protocol is QR-code-free (exception, optionally, for the first step). Voters only need to enter a short code for each question: if a question has 15 possible answers, then the voter simply needs to enter a number between 1 and 15. The association between these small codes and the corresponding encrypted votes is committed by the setup component and audited by auditors before the start of the election. The return codes

guarantee that the voting device cannot modify the voting code without being detected. Studies have also explored the usability of entering voting codes [MSLM19, KNBV17], and even though the voting codes were longer than in our proposal (8 digits in [MSLM19], and 3 characters in [KNBV17]), reached acceptable usability scores.

Comparison to non-Swiss specific systems. Outside the Swiss context, BeleniosVS [CFL19] follows the same approach as HKD: the code is simply the encrypted vote. However, BeleniosVS assumes in addition that voters can access a public Bulletin Board. Unfortunately, existing techniques to implement a secure public Bulletin Board [HSB21] typically assume a signature mechanism. Since a voter cannot check the validity of a signature by themselves, they need to rely on their voting device to do so. Assuming the voting device dishonest as in our setting, it may simply lie to the voter. Therefore, an approach based on a public Bulletin Board does not seem to be a realistic option under our trust assumptions.

Pretty Good Democracy (PGD) [RT09] uses long, hard to guess, codes for each voting choice, that intuitively points to the right encrypted vote in the data prepared during setup. In PGD, if the attacker knows another code, it may swap votes. While long codes may be acceptable when only one vote is expressed, this does not fit well in the Swiss context. Furthermore, the control components (the trustees in the PGD setting) need to be trusted for verifiability, as they may lie on how they shifted the ciphertexts during the setup. Pretty Understandable Democracy (PUD) [BNOV13] also uses pointers to ciphertexts. The voter needs to enter several codes for each voting option, received from different voting sheets, one from each server (these servers intuitively correspond to the Swiss control components). This may again be very cumbersome in practice. The VeryVote system [JRF09] shifts the trust from the voting device to the voting server that (intuitively) encrypts the vote for the voter, but in a verifiable manner. The authors later extend their approach to remove the trust in the voting server [JFR13], but at the cost of a secure hardware token on the voter side, hence trusting again some part of the voting device.

All these systems use either voting codes which need to be resistant to random guessing or encode directly the ciphertext or signatures. This results into long voting codes in the order of 128 bits or even more. To our knowledge, the only scheme that uses short voting codes, hence in the order of number of candidates, is D-Demos [CZZ+16]. As in our proposal, the voting code simply points to the right ciphertext, and the number of needed voting codes is simply the number of voting options. D-DEMOS uses a double-ballot system, where the voter chooses which ballot to audit and which one to use for voting, which leads to a slightly more complex system. Moreover, D-Demos assumes a public Bulletin Board, which tends to

discard it for real-world usages under our trust assumptions as discussed above. Finally, their setup component has to generate a lot of cryptographic, randomized, material, which may be hard to audit.

Table 3.1 displays a comparison of the existing code voting systems that do not assume any trust in the voting device. We first compare their security (privacy and verifiability) for different trust models. We highlight the trust model of the Swiss Chancellery and our trust model. For the sake of comparison, we also display the two main systems that have been designed for the Swiss context, namely Swiss Post [Swi22] and CHVote [HKLD17]. None of them achieve privacy against an insecure voting device but in contrast, they achieve a better verifiability level than code voting when the setup component is dishonest but the voting device is trusted (a scenario outside the Swiss trust model).

We also compare the systems in terms of usability and auditability. We pinpoint which systems use short voting codes and which are "vote and go"; that is, a voter simply needs to interact with the system during the voting phase, and does not need to come back afterwards, e.g. to check that their vote has been counted. In terms of auditability and simplicity, we highlight systems that use a deterministic setup and do not use cryptography on the voting device. We further document whether the proposals are accompanied by a security proof of verifiability, vote privacy, or both.

	Swiss Setting				Voting Codes Systems				
	SPP [Swi22]	CHVote [HKLD17]	HKD [HKD23]	**Proposal**	PGD [RT09]	PUD [BNOV13]	D-DEMOS [CZZ+16]	BeleniosVS [CFL19]	VeryVote [JRF09]
Trust for Vote privacy									
1/n CC, SC, VD	●	●	●	●	●	●	●	●	
1/n CC, SC			●	●	●	●	●	●	
1/n CC, VD								●	
1/n CC									
Trust for Verifiability									
1/n CC, SC, VD	●	●	●	●		●	●	●	●
1/n CC, SC	●	●	●	●		●	◐	◐	◐
1/n CC, VD	●	●							●
1/n CC									◐
Security Proof									
Privacy	●			●			●	●	
Verifiability	●	●		●			●	●	
Construction									
Short Voting Codes				●			●		
Deterministic SC		●	●	●	●				
Vote and go	●	●	●	●	●	●			
No Crypto on VD			●	●	●	●	●		

Table 3.1: Blocks one and two describe scenarios with different trust assumptions for vote privacy and verifiability corresponding to the control components CC, the setup component SC and the voting device VD. In *italic* the Swiss trust model, and in gray our stronger trust model (we no longer assume a trusted voting device for privacy). Protocols marked with ◐ assume a public bulletin board that voters can securely access, which implicitly assumes some trusted device. In PGD, all control components need to be honest for verifiability, as they may lie on how they shifted the ciphertexts during the setup. In VeryVote, the server is trusted to encrypt the candidate for the voter, which is a stronger assumption than the ones we consider here.

3.3 Voter's view

To introduce the protocol, we first present the voter's view. The voter is given a ballot sheet with a QR code to login, and codes to cast and confirm their vote. See Figure 3.1 for an example using realistic parameters (e.g. length of codes). First, the voter scans the QR code to login ① with her identifier *Id*. The QR code contains the URL of the voting server, reducing the chance that the voter ends up on the wrong page. Then, the voter enters the short voting codes $c_1, \ldots, c_n$ of the voting choices they intend to vote for ②. The voting codes are sent to the voting system, which responds with return codes $rc_1, \ldots, rc_n$, one for each selected voting option (codes for blank votes shall be provided). If and only if these codes match what is printed on the voting sheet ②, the voter is instructed to confirm the vote ③ by sending the authentication code *ac*. This is the only long secret value that authenticates the voter. The voter finally receives a finalization code *fc*, that guarantees them that their vote has been counted.

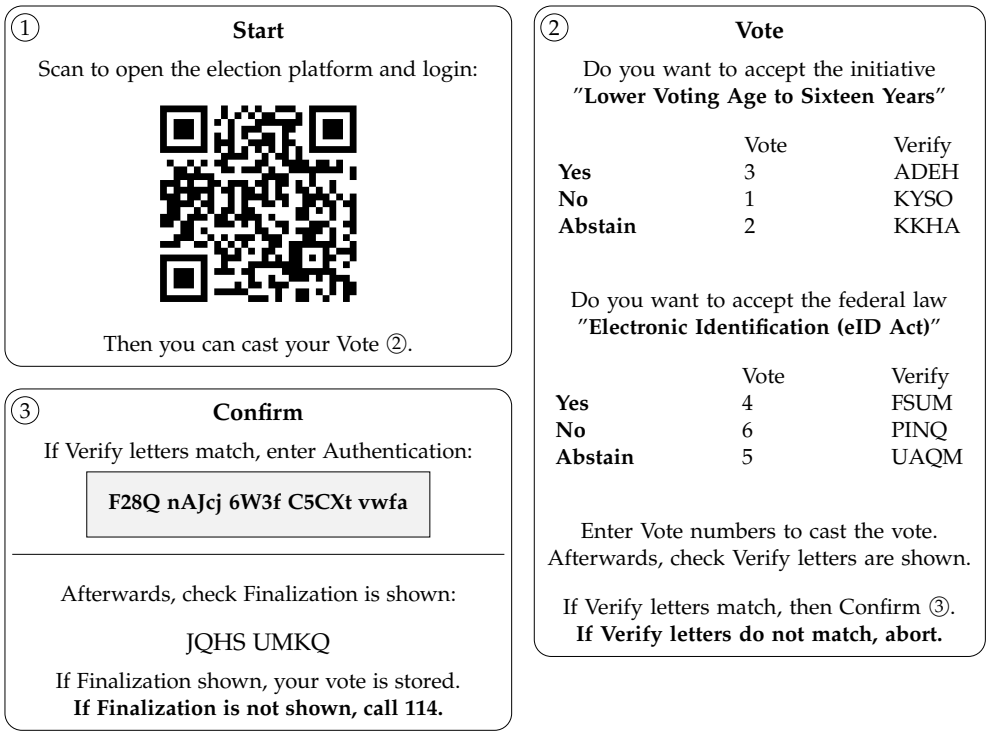

Figure 3.1: An example of a voting sheet using realistic parameters. ① contains a QR code with the voter id. ② contains the voting codes and their respective return code. ③ contains the authentication code and the respective finalization.

The voting client essentially needs to forward values to the voting system and back to the user. It does not need to encrypt or sign the vote. The

voting client may guide the user with basic validation (e.g. ensuring the voting code entered for the first question is between 1 and 3).

3.4 Protocol

The protocol is divided into the setup phase, the voting phase and the tally phase. In the setup phase, all parties generate and exchange the cryptographic material necessary to run the voting and tally phase. In the voting phase, the voter submits their vote and performs their individual verifiability checks (see Section 3.3). In the final tally phase, the votes are decrypted and counted. For simplicity, we present our protocol for an election with one question only, but it can easily be extended to several questions (e.g. the voting sheet presented in Figure 3.1 has two questions).

Notations We denote $\mathbb{Z}_n$ or $[1, n]$ the set of integers between 1 and n. The set of all possible permutations of $\mathbb{Z}_n$ is denoted S_n. The composition of two permutations π_1 and π_2 is denoted $\pi_1 \circ \pi_2$. Further, we denote $\oplus$ the addition modulo some m. Note that the m is always obvious from the context, inferred from the size of the set out of which the added values have been picked, hence we do not write it explicitly. We write $\sum_1^n$ (or $\oplus_1^n$) for addition (or modular addition) of values from 1 up to n. We sometimes abuse notation and perform it to tables, applying the operation to values under the same key (e.g., for $t_1 = \{a \to 1\}$ and $t_2 = \{a \to 2\}$, then $\sum_{i=1}^2 t_i = \{a \to 3\}$).

The set of voting options is denoted V and is of size n_v. The correspondence between voting options and voting codes will be denoted by an injective substitution (or a table) σ from V to $\mathbb{Z}_{n_v}$. The application of a substitution σ to an element $v \in V$ is denoted $\sigma[v]$. We assume an *initial correspondence* σ_0, chosen publicly by the authorities, that maps each voting option to some integer in $\mathbb{Z}_n$. This is fixed for all voters. For example, if $V = \{a, b, c\}$, the initial correspondence could be $\sigma_0 = \{a \to 1, b \to 2, c \to 3\}$.

Cryptographic primitives Our protocol relies on three standard primitives.

- a hash function denoted hash,
- a signature scheme. We denote $\sigma = \mathsf{sign}(sk, m, r)$ the signature with signing key sk, message m and randomness r. Similarly, we denote $\mathsf{verify}(pk, m, \sigma)$ the verification applied to public key pk, message m and signature σ. We write $\mathsf{sign}(sk, m)$ when the source of randomness r is clear from the context.
- an encryption scheme compatible with a homomorphic tally or a mixnet, such as the one used in Helios [Adi08]. We denote $E = \mathsf{enc}(pk, m, r)$ for the encryption of message m with public key pk and randomness r.

3.4.1 Setup

Before the setup, we assume that the authorities have agreed on a list of identifiers $\boldsymbol{Id}$, one for each eligible voter. The identifiers are used to keep track of the cryptographic material per voter, and they are not assumed to be secret.

The length of the different codes is given as a parameter of the election. For the return and finalization codes, the adversary has a single try to convince the voter, hence the values do not need to be hard to guess. Typically, we will consider 4 digits codes. For the sake of generality, we note n_{rc} and n_f the sizes of the sets in which the return codes, and the finalization codes live (in our example, with 4 digits, $n_{rc} = n_f = 10^4$). On the contrary, the adversary will have access to the hashes of the authentication codes. Hence, they must be hard-to-guess, and the probability of collisions must be negligible. Denoting n_a the size of their universe, we typically choose $n_a = 2^{256}$. Finally, we note $\mathbb{Z}_{n_r}$ the set of integers in which encryption randomnesses are drawn at random. In most cases, we will have $n_r = 2^{256}$.

Key generation The decryption authorities jointly generate the election public key pk_E, using a Distributed Key Generation, for example, the one used in Belenios [CGG19] or ElectionGuard [BNPW24]. They each hold a share of the decryption key.

We also assume that each control component CC_i has a signing key sk_i, with associated public verification key vk_i.

Preparation of the voting sheets. During the setup, the setup component is in charge of sending to each voter:

- their identifier $Id \in \boldsymbol{Id}$,
- a correspondence σ between voting options and voting codes,
- a table t that maps each voting option to a return code,
- an authentication code ac,
- a finalization code fc.

To obtain a deterministic setup component, each control component generates for each $Id \in \boldsymbol{Id}$:

- a randomly chosen correspondence π_i between voting options and voting codes,
- a table t_i that maps each voting option to a randomly chosen return code,

- a table r_i that maps each voting code to a randomly chosen encryption randomness,
- a random authentication code ac_i,
- a random finalization code fc_i.

Then, the setup component simply combines the contributions of all control components:

- $\sigma = \sigma_0 \circ \pi$ where $\pi = \circ_{i=1}^{m} \pi_i$: the initial correspondence σ_0 is permuted using the combination π of the permutations π_i chosen by the CC_i,
- $t = \Sigma_{i=1}^{m} t_i$,
- $r = \oplus_{i=1}^{m} r_i$,
- $ac = \Sigma_{i=1}^{m} ac_i$,
- $fc = \Sigma_{i=1}^{m} fc_i$,

where $\oplus$ denotes the bitwise exclusive-OR operator, and $\Sigma_{i=1}^{m} x_i = x_1 + \ldots + x_m \mod N$ with N defined from the context (e.g. $N = n_{rc}$ when computing t, $N = n_a$ for ac, and $N = n_f$ for fc). By abuse of notation, when operands are tables, the operator is applied for each index.

Additionally, the setup returns to the control components:

- $h = \mathsf{hash}(ac)$, the hashed authentication code,
- a table te from the voting codes to the corresponding (encrypted) voting options
$$\{c \to \mathsf{enc}(pk_E, \sigma^{-1}[c], r[c]) \mid c \in \mathbb{Z}_{n_v}\}$$
using the XORed randomness received from the CC_i.

The setup phase is depicted in Figure 3.2, as well as the remainder of the protocol.

Audit of the setup component The setup component needs to be fully trusted. Notably, it could switch the vote and verification codes of plain votes, such that a different voting option is voted for than the one intended. We introduce an audit procedure to improve the detection of a misbehaving setup component.

To make an audit of the setup component practical, we ensure it only runs deterministic algorithms. Note that if at least one control component performs honestly, the adversary has no advantage in guessing the ballots.

To audit the setup component, we first note that the identifiers in $\boldsymbol{Id}$ are not tied to voters. They are just numbers (possibly randomly chosen). Hence,

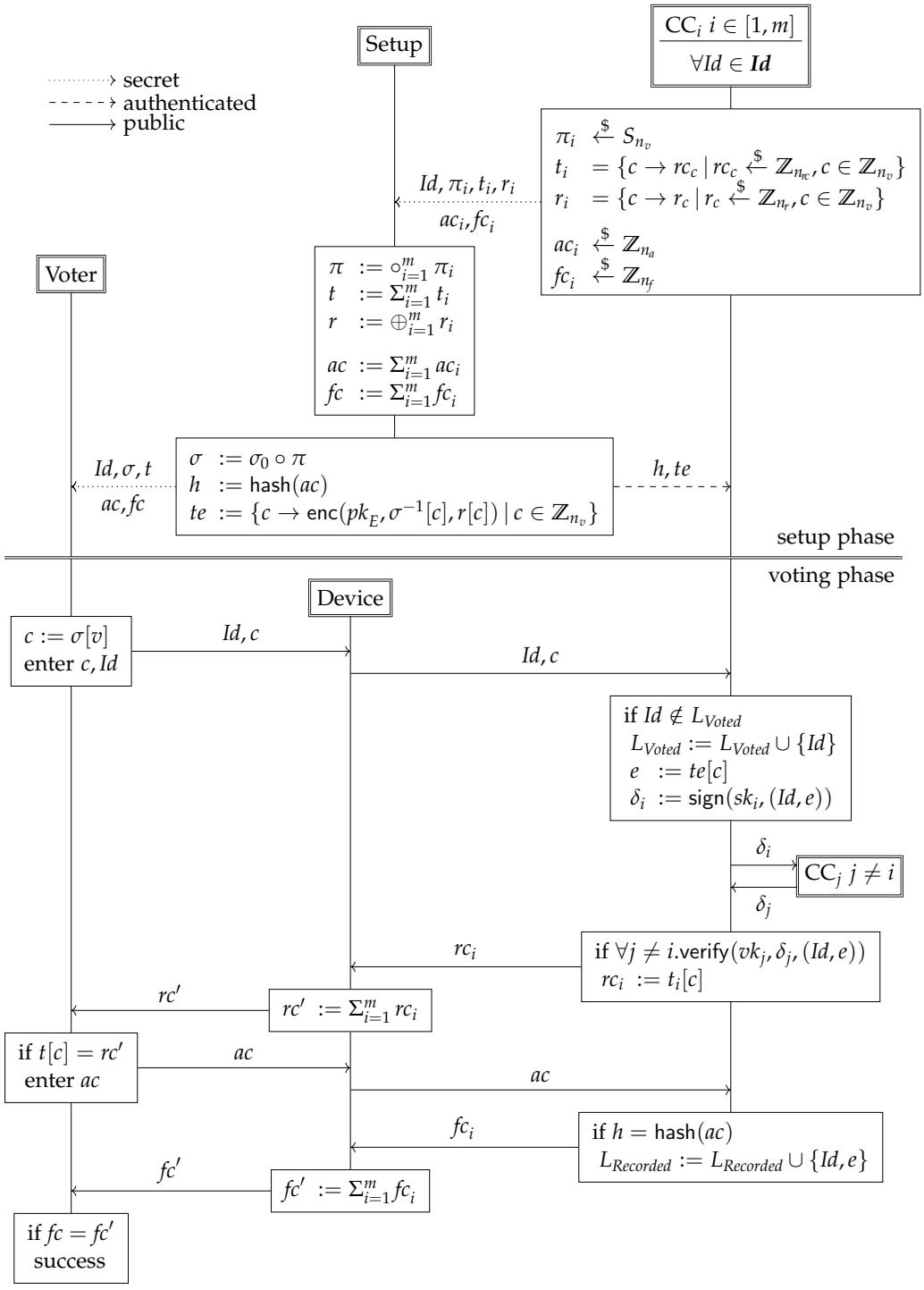

Figure 3.2: Protocol.

when preparing the election, the authorities simply add n additional entries to $\boldsymbol{Id}$, for n chosen in such a way that if n voters are audited, the risk is sufficiently minimized. After the setup component finishes execution, each control component i chooses some subset of voters to audit $\boldsymbol{Id}_A^{(i)} \subset \boldsymbol{Id}$ such that $|\cup_i \boldsymbol{Id}_A^{(i)}| = n$. Any corresponding Id can no longer be used to cast votes, but instead, the control components publish all values they generated for this Id. The control components then assert that the voting sheet, h, and te have been generated correctly. Finally, the remaining voting sheets are randomly assigned to voters, to prevent targeted attacks on specific Ids from the setup and control components.

3.4.2 Voting

The voter cast their vote by interacting with the control components.

- The voters first select the voting code c corresponding to their vote v and they simply enter c together with their identifier Id to their voting device.
- Each control component checks that they did not receive Id yet, signs Id together with the ciphertext e corresponding to c in their table te, and sends this signature to the other control components. Once the control component has received corresponding signatures back from all other control components, it sends their share rc_i of the return code.
- The voting device combines the return codes and displays $rc = \Sigma_{i=1}^{m} rc_i$ to the voter. If the voter agrees, they send their authentication code ac.
- Each control component checks that the code ac is valid by comparing it to its hashed value h. They now record the ciphertext e and Id in the list $L_{Recorded}$ of recorded votes and they each send their share fc_i of the finalization code.
- The voting device displays the combination of the fc_i to the voter, which ends the voting phase.

At any time, if the voter detects some inconsistency, they should contact the voting authorities, who then offer other means of voting.

Multiple voting options For simplicity, we considered the selection of one voting option only. In case the voter has to select k out of n options, then the voter simply sends the k corresponding voting codes, the control components return k return codes and the protocol runs accordingly. In case the voter may select *up to* k out of n options, we use the same approach as for other Swiss systems: we add k blank options so that the voter always selects k options, possibly with some blank ones. The voter hence has to

send k voting codes and check k return codes, even if they vote for less than k actual candidates.

Several questions can easily be supported using different code voting names. For example, the voting sheet provided in Figure 3.1 corresponds to an election with two questions, of 3 voting options each. How to adapt our protocol to cover more complex electoral rules, where a ballot may be further constrained (e.g. when at least one non-blank candidate needs to be selected) is left as future work.

3.4.3 Tally

At the end of the voting phase, each control component has a list of recorded votes $L_{Recorded}$ which need to be tallied. Ideally, all control components should have the same $L_{Recorded}$ in their local state. However, dishonest control components might have added, modified, or dropped entries in their local $L_{Recorded}$, hence the control components need to agree on their list $L_{Recorded}$ of encrypted votes.

Agreement procedure To establish L_{Agreed}, each control component CC_i sends $L_{Recorded}$ with the "proof" for each entry (that is, ac and signatures over (Id, e)) to all other control components CC_j . Each control component then defines L_{Agreed} to be $L_{Recorded}$, in which they add any e such that

- they received a valid signature $\mathsf{sign}(sk_j, (Id, e))$ from each CC_j,
- h is associated to Id and $h = \mathsf{hash}(ac)$.

After this step, all control components agree, or they can detect who is misbehaving.

Tally procedure After establishing L_{Agreed}, the decryption authorities execute the tally procedure with the ciphertext given by L_{Agreed} as input. We leave this step abstract as our protocol can support both a homomorphic tally [Adi08] or a mixnet-based tally [Dou]. In both cases, the decryption authorities provide verifiable proof of correct tally.

3.5 Security Claims

Our protocol satisfies all the security requirements of the Swiss Chancellery, even if the voting device is dishonest. More precisely, our protocol guarantees vote privacy, even when the attacker controls the voting device, all but one control component, and less than t decryption authorities. Our protocol also ensures verifiability, even when the attacker controls the voting device and all but one control component. In particular, it guarantees cast-as-intended and vote privacy against a dishonest voting device.

In this section, we give an intuitive explanation why our protocol guarantees the verifiability and privacy in our trust model. We provide formal proofs in Chapter 4.

Verifiability For some vote e to enter the tally, there are two conditions: there must exist a signature over (Id, e) from all control components, and ac must be known. The verifiable tally procedure then ensures the vote is not modified during the tally.

The fact that ac is known implies that the vote c of some given voter has been registered by all control components. First, we note that ac is hard to guess for the adversary, and therefore must have been entered by the voter. The voter only enters ac if they observe rc to be correct in relation to their vote c. As the adversary has only a single try to present a correct rc to the voter, it is unlikely that the adversary has guessed it. Instead, rc was reconstructed out of rc_i published by the control components. The honest control component will only publish the corresponding rc_i of the vote c, if said c is indeed the first vote of that voter.

When the voter has seen a correct finalization code fc, it implies their vote will indeed enter the tally. As the adversary has only a single try to present a correct fc to the voter, it is unlikely the adversary guessed it. Instead, fc was reconstructed out of the fc_i published by all the control components. The honest control will only publish their part if it has seen ac, and has a vote c registered for that voter. Stronger, it has indeed received signatures over (Id, e) for $e = te[c]$ of all other control components. Therefore, the honest control component is in possession of all the values necessary to guarantee the inclusion of e in the tally.

Privacy We first collect which values learned by the adversary depend on the selected voting option. When the voter has chosen their plain vote v, they then map it to the voting code $c := \sigma[v]$, hence c clearly depends on v. The other values that depend on v are the corresponding partial return codes $rc_i := t_i[c]$, and the encrypted vote $e := te[c]$. All the other values, including those from other voters, are independent (i.e. remain the same if the voter would chose a different v').

The adversary cannot relate c to its corresponding v, as it does not know σ, and cannot reconstruct π (as π_i of the honest control component is unknown to the adversary). The adversary cannot learn additional information from rc_i, as they were randomly chosen. Further, the adversary is unable to decrypt e as at least one decryption authority is assumed honest. The privacy-preserving tally procedure then ensures that the adversary does not learn anything about v, apart from the result of the election.

3.6 Discussion

We have provided a simple voting protocol that satisfies both verifiability and privacy, even against a dishonest voting device. We achieve this using short voting codes, suitable to be entered by the voter by hand. These voting codes are as short as they could possibly be: only as many different voting codes are needed as the number of voting options available.

The protocol relies on only standard cryptography, such as signatures and hashes. While we rely on less common internet-voting specific building blocks, such as distributed key generation and a verifiable shuffle, they are not deeply integrated into our core protocol, and therefore existing implementations can be reused as a black-box (e.g., ElectionGuard [BNPW24] and Verificatum [Dou]). Further, the time-critical actions to be performed by both the voting device and the control components during the voting phase are trivial and fast, and therefore cause no (cryptographic) bottleneck that would need any complex performance optimizations. We believe therefore that our protocol can easily be implemented and maintained, even by programmers with a light background in cryptography.

Simplicity as a goal Both the Swiss Post protocol and CHVote have complex specifications, implementation and proofs. While it is hard to quantify complexity, both specifications span more than 300 pages to detail the cryptography used and the messages exchanged between the various components [HKLD17, Swi24c, Swi24d, Swi25a]. Some of these precise descriptions concern the setting (e.g., defining voting rules) or generic building blocks such as authenticated channels or login methods, and any fully specified voting system will need to describe them in detail. However, while it is hard to quantify, considerable effort is spent to describe the system-specific cryptography (e.g., the return codes table in the Swiss Post Voting System [Swi24c]). Besides the enormous amount of work necessary to create high-quality documents of this size, the extent of the work also impacts anything else that is based upon it, for example laborious computational proofs [BCG+18b, Swi25c], or even harder to quantify efforts in review and implementation.

Hence, future research may also prioritize how to design simpler schemes. Can we design such a measure of complexity (e.g., as proposed with the "practicality" criterion from [MMC+24]) to quantify this, and potentially even trade it against slightly stronger security properties of more complex schemes? Further, can we devise a method to derive simple baseline schemes in a given setting, to get a conceptual answer on the minimality of acceptable schemes? And finally, can we quantify how those simpler schemes indeed lead to more secure systems (in the sense of that fewer critical bugs are uncovered)?

New tradeoffs in corruption scenarios The proposal strengthens two possible corruption variants of the Swiss setting. It protects privacy against a dishonest voting device, which is notably already untrusted for verifiability. Further, it provides stronger protection against a store-now-decrypt-later attack on the channel between the voter and the voting system, as no ciphertext is transferred over the wire. However, it also crucially weakens other corruption scenarios. A dishonest control component can now, on its own, break verifiability by misattributing the codes printed on the voting sheet. The randomized audit of the setup component might detect this; but this attack may also be executed on the postal channel. For example, a person living in the same household might open the letter, scan and edit the voting sheet, and trick the real voter with a fresh print to cast their vote against their intention. Further, a dishonest setup component with access to the ballot box is now also able to break privacy.

While all of these tradeoffs are of no concern according to the current trust assumptions, they are exactly that: *assumptions*. But what happens if one of these assumptions ceases to hold? Further research may focus on devising these corruption scenarios, and systematically extend the robustness of the schemes, even in the presence of more powerful adversaries than foreseen. In the Swiss setting, legislation may then want to further constrain compliance to systems that are more robust in these corruption scenarios.

Interestingly, we may also explore to let the voter decide which corruption scenario they want to defend against. To apply this idea to this work, we could let the voter decide whether they want to enter their vote in plain, reducing reliance on the setup component, or keep using the voting codes, to avoid any trust in the voting device.

Usability There are already promising results from literature documenting that entering short character strings reaches acceptable usability [MSLM19, KNBV17], while the voting codes proposed in this work are even shorter. However, existing usability studies do not yet explore the multi-race setting (e.g., [HBT+24, TVK22, CDGG23, MZR+21, NKV+24]). This multi-race setting is precisely where we claim a clear advantage over the QR-codes necessary by previous work [HKD23]. In this work, we inspire mostly from Switzerland, where voters decide on many questions in the same election: a reasonable average of questions is 6, but up to 13 have been observed in major cities [Kan22, Sta22]. However, multi-race elections are indeed common, for example when multiple layers of government are voted on at the same election day, as it happens in Germany, Estonia, France and many more.

Hence to really confirm or refute our usability claim, we need to explore the multi-race setting. In this setting, the *efficiency* of how voters enter their choices becomes more important, as these steps need to be repeated for ev-

ery choice. Further, also for verification, the multi-race setting introduces additional complications: If the verification needs to be performed per voting choice, the multi-race setting allows new attacker strategies, e.g., modifying all but the first or the last run. If the usability of detecting these attacks is low, new trade-offs may emerge. For example in [HKD23], the voting codes cannot be forged, hence it would be sufficient that the voter only performs a single check to confirm the server has received the intended number of ballots. Hence the (expected) more cumbersome way to cast the ballots using QR-codes may still be worth it, versus comparing return codes for each voting choice, if the usability of the whole system is considered.

Chapter 4

ProVerif Proof Framework

Proving the security of voting protocols pushes available tools to their limits. Two main properties need to be proven, E2E-Verifiability and vote privacy. E2E-Verifiability attests that the final result is composed out all votes of honest voters who verify, some votes of honest voters who do not verify, plus at most one vote per dishonest voters [CGGI14]. This is modelled as a correspondence property, and notably requires to *count* the votes, in order to ensure that the tallied votes exactly correspond to the votes cast by voters. Vote privacy ensures that the adversary cannot learn how the voters cast their votes, besides what is revealed by the election result anyway [BT94]. This is formalized by the adversary being unable to distinguish when Alice is voting 1 and Bob is voting 0 from the converse scenario where Bob is voting 1 and Alice 0. This needs to be expressed as an equivalence property [DKR09], which are notoriously harder to verify than correspondence properties [BS16, BDS15]. As a third complication, neither the setting nor the protocols under proof are simple. In our generic election setting, we aim to model an arbitrary number of elections, voters, and votes.

To deal with the complexity of analysing a voting system, instead of a single full model, often several simplified models are developed. This is not only more work — devising meaningful simplifications that still allow to prove some relevant property (and reviewing them to be sound) — but it also increases the risk of unsound models. To prove verifiability, it is tempting to simplify the tally phase that is mainly there to protect privacy. Some models even omit the encryption entirely, resorting to casting the votes in clear (e.g., [BCG$^+$18a]). Worse, for privacy, models often do not include the verification steps (e.g., [HC19]) made by the voters, while the verification could potentially break privacy, too.

To make the verifiability proofs in ProVerif more practical, in 2023 a framework [CCD23] has been introduced to ease these kinds of proofs. It offers pre-optimized building blocks common to voting systems such as a bulletin board or a tally process. It implements revote, and an unbounded number of elections and voters. It includes 40+ generic lemmas and axioms to aid with termination, building upon this rather recent capability of ProVerif [BCC22]. The framework has been successfully applied to several existing protocol

models such as the Swiss Post Protocol and has been used in subsequent works to prove verifiability of novel protocols [CDGL24, CDM24].

However, no such support exists for privacy proofs. Therefore, we extend the verifiability framework towards proving privacy. This requires reworking the model, as the existing verifiability framework is incompatible with vote privacy: Indeed, to ease the verifiability proof, the pre-built tally process opens the ballots in the order in which they were received. So if Alice votes first, her vote will be published first, which completely breaks vote privacy. Further, we devise a new set of lemmas and axioms to aid termination and precision when proving privacy.

Our contributions We provide a unified framework to prove both vote privacy and verifiability based on a single model of the protocol. Our framework is backward compatible with the initial framework [CCD23] in order to be able to reuse existing models.

We apply our novel framework to all previous protocols considered in the initial version of the framework, as well as to protocols subsequently using it [CDGL24, CDM24]. We could prove privacy and verifiability in all cases but two: CHVote [HKLD17] and Vote&Check [CDGL24]. We will elaborate on the case studies and the limitations in Section 4.7.

Related publication This work is published at CSF 2026 as joint work with Vincent Cheval, Veronique Cortier and Alexandre Debant [CCDM26].

4.1 ProVerif

ProVerif [Bla16] is an automatic tool designed to conduct security analyses of cryptographic protocols.

It is set in the symbolic model, i.e., cryptographic primitives are abstracted by functions that only capture the functional behaviour of the primitives. For example, public-key encryption and decryption can be represented as follows. First, we define the functions symbols `pk` and `sk` to derive the public and the private key, respectively, out of some value of type `bitstring`. The function symbols return values of type `public_key` and `secret_key`, with these custom types declared using `type` directly above. Further, we mark the function symbol `sk` private, hence the adversary is not able to use it. In contrast, `pk` remains public to model that public keys are known by everyone.

```
1 type public_key, secret_key.
2 fun pk(bitstring): public_key.
3 fun sk(bitstring): secret_key [private].
```

Given this rudimentary public key system, we define the function symbols `encrypt` and `decrypt`. They model respectively the encryption algorithm applied to a term of type `vote` and the decryption algorithm applied to any `bitstring`. Further, to provide semantics to these abstract symbols, we define a *rewriting rule* using `reduc` that specifies that the decryption of an encryption yields the plaintext as expected. Here, we define the decryption key corresponding to some encryption key `pk(elec)` as the application of `sk` over the same `elec`, i.e., `sk(elec)`. As it is specified with rewriting rules, we call the function symbol `decrypt` a *destructor*. By convention, a function symbol occurring at the top of a rewriting rule is called a *destructor*, while the others are called *constructors*.

```
4  type vote, random.
5  fun encrypt(public_key, vote, random): bitstring.
6  fun decrypt(secret_key, bitstring): vote
7  reduc forall elec, m, r;
8    decrypt(sk(elec),encrypt(pk(elec),m,r)) = m
9  .
```

This system of function symbols and rewriting rules is expressive, and besides support for the usual cryptographic primitives (e.g, hashes, signatures, zero-knowledge proofs), also complex structures can be expressed, such as shared decryption. Given a set of atomic values such as names or variables, and a set of function symbols, we call *terms* the infinite set of elements inductively defined by applying function symbols over atomic values or terms. These terms are equipped with rewriting rules (as presented before) or equations (see [Bla23] for details). For sake of simplicity, we call them both the *equational theory* accompagnying the terms.

Then a protocol is modelled through *processes*, written in an abstract programming language. For example, we model a voter who casts an encrypted ballot as follows. We define the process `Vote`, which takes as argument the election id `elec` and the vote `v`. We type these arguments as `elec_id` and `vote`, respectively. We then generate a fresh name unknown to the adversary using the `new` keyword: the encryption randomness `r`. Now, we can define the ciphertext `c` using the `let` keyword. Concretely, it is derived by applying the function symbol `encrypt` to the election public key `pk(elec)`, the vote `v` and the randomness `r`. Finally, using the `out` keyword, the ciphertext `c` is sent over the channel `ch_pub`. This channel is public, and therefore the ciphertext becomes attacker knowledge.

```
10 channel ch_pub.
11 let Vote(elec: elec_id, v: vote) =
12   (* form ballot *)
13   new r: random;
14   let c = encrypt(pk(elec), v, r) in
15
16   (* cast ballot *)
17   out(ch_pub, c)
18 .
```

ProVerif models a Dolev-Yao adversary [DY03] who fully controls the network. They can read and intercept any message sent over a public channel and they can send any message that it can build from their knowledge by using public function symbols. For example, if the attacker would know `sk(elec)` and the ciphertext `c` as defined above, then they could apply the `decrypt` function symbol to `c` to derive `v` using the rewriting rule introduced before. However, the attacker cannot build `sk(elec)` by themselves, even if they had knowledge of `elec`, as `sk` is declared [`private`]. While deriving the proof, ProVerif ignores the declared types, hence these do not impact soundness of the proof. The types are however useful to write correct models, as they aid to spot simple modelling mistakes.

We can now propose a small `Server` process. The server receives messages using the `in` keyword, over the public channel `ch_pub`. While we imply that it receives the ciphertext `c` send by the voter above, as the channel is public the attacker may supply any value it is able to derive. Finally, the server stores the value in the table `ballot_box` using `insert`. In ProVerif, tables are persistent storage filled using the keyword `insert` and accessible inside process with the keyword `get`. When accessing an element, one can look for an element which satisfies some property `phi` using the command `get tab(x) suchthat phi in P else Q`. If such an element exits, the process `P` is executed, otherwise the process `Q` is. Both the condition `phi` as well as the else branch are optional, and can be omitted when empty. The adversary cannot access the tables, i.e., can neither read or write to the tables.

Lastly, we propose a simple `Tally` process. First, it reads out all the values from `ballot_box` using `get`, while we filter the values to those where the first column is equal to `elec`. To filter, we write `get ballot_box(=elec, c) in`, which is a convenient shorthand to write `get ballot_box(x_elec, c) suchthat x_elec=elec in`. Then, the tally decrypts, and raises an event `Tallied` with the vote `v` it recovered. Events are outside of the protocol (notably do not contribute to attacker knowledge), they are however useful to track specific steps and states of the protocol, to then formulate security properties over it.

```
19  table ballot_box(elec_id, bitstring).
20  let Server(elec: elec_id) =
21    in(ch_pub, c: bitstring);
22    insert ballot_box(elec, c).
23
24  event Tallied(elec_id, vote).
25  let Tally(elec: elec_id) =
26    get ballot_box(=elec, c);
27    let v = decrypt(sk(elec), c);
28    event Tallied(elec, v);
29    out(ch_pub, v).
```

Finally, we can boot our processes. In any ProVerif model, there is a default process under the keyword `process`. This process can be used to boot up all

other processes. Here, we let the attacker pick the election `elec` and the vote `v` to cast. We again raise an event `Voting`, to track the vote that is intended to be cast. Lastly, we boot up all other processes in parallel using the pipe operator.

```
30  event Voting(elec_id, vote).
31  process
32    in(ch_pub, (elec: elec_id, v: vote));
33    event Voting(elec, v);
34    Vote(elec, v) | Server(elec) | Tally(elec)
```

Then, ProVerif exhaustively explores all the different *execution traces* of the protocol, i.e., the different ways the protocol might be executed. Over these traces (security) properties can be formulated.

Properties ProVerif supports correspondence and equivalence properties.

Correspondence properties allow to say that whenever some event $A(n, \ldots)$ has occurred in the execution of the protocol, then another event $B(m, \ldots)$ must have occurred before. Further, we can express relations inside those events, for example that m must equal n. To formulate a simple property over our introduction example, we may state a `query` which states that the tallied vote was the one the voter intended to vote for:

```
1  query elec, v;
2    event(Tallied(elec, v)) ==> event(Voting(elec, v))
3  .
```

ProVerif will not be able to prove this query. Indeed, there is a simple attack where the adversary is able to switch out the vote. First, the adversary starts the protocol by sending (`elec`, `v`) to `ch_pub`. This leads to the default process to raise `Voting`(`elec`, `v`), and then boot up the other processes. Now, as the adversary knows `elec`, it can build `pk`(`elec`). Therefore, it can build a ciphertext `c'` encrypting some different vote `v'`. It can then send `c'` over `ch_pub` to the `Server`, which stores it in the table `ballot_box`. Finally, the `Tally` will decrypt `c'`, and raise the event `Tallied`(`elec`, `v'`). This contradicts our query, as no corresponding event `Voting`(`elec`, `v`) has been raised. We will see in Section 4.2 how to build a more secure internet voting protocol.

The other type of properties that ProVerif is able to prove are equivalence properties, which express that the adversary cannot distinguish between two scenarios. This means that for all execution traces in the first scenario, there exists another trace in the second scenario with the same visible actions (i.e., communications on a public channel), such that if the adversary performs a test, it succeeds in scenario 1 if and only if it succeeds in scenario 2. We omit the formal definition of equivalence ProVerif is considering and let the reader refer to [BCC22] for more details. An equivalence property is used in particular to express vote privacy [Ben87, KR05] of the form

$$Alice(0) \mid Bob(1) \approx Alice(1) \mid Bob(0)$$

This means that the adversary should not be able to distinguish the scenario on the left where Alice votes 1 and Bob votes 2 from the converse scenario on the right. To translate this equivalence property to ProVerif, instead of formulating a query as for correspondence properties, we use the special `diff` operator. We map 0 and 1 to constants `voteA` and `voteB`, and then express that Alice votes `voteA` left and `voteB` right with `diff[voteA, voteB]`. We do the converse for Bob, and let the two voters act in parallel. This implements the definition, and altogether, gives us the following new default process. Note how we use the pipe operator to replicate the processes `Server` and `Tally` (i.e. boot them multiple times) that they can register and tally multiple ballots.

```
30  const voteA, voteB: vote.
31  process
32    in(ch_pub, elec: elec_id);
33    Vote(elec, diff[voteA, voteB]) | Vote(elec, diff[voteB, voteA]) |
34    !Server(elec) | !Tally(elec)
```

Again, in this model the property is trivially broken in the traces where the `Tally` process publishes the vote of either Alice or Bob. For example, in the trace where the ballot of Alice is decrypted, the tally process will therefore publish `diff[voteA, voteB]`. The attacker can however trivially distinguish this, for example by checking equality to `diff[voteA, voteA]`, a term known by the attacker since `voteA` is a public constant. The equality check will succeed on the left, but not on the right, and therefore equivalence is broken. Intuitively, this is due to the fact that votes are not shuffled in `Tally`. Modelling this in ProVerif is difficult, and is the focus of this chapter.

Termination and precision ProVerif is proven to be sound [Bla16], hence it will never declare a property to be proven when there is an attack. Further, ProVerif is automated, and therefore will try to prove the properties (given the model) on its own. However, the fully automatic proof may not terminate for a variety of reasons. Further, ProVerif overapproximates, and therefore may find false attacks. In these cases, ProVerif needs to be guided to find a suitable trade-off between termination and precision.

Under the hood, ProVerif works as follows. It first translates the processes and equational theory into Horn clauses, which are of the form $H_1 \wedge H_2 \wedge \cdots \implies C$. Together with general clauses capturing the Dolev-Yao adversary capabilities (e.g., forging and sending messages), these clauses are then *saturated*. We give an intuition of how this saturation works. First, all initial clauses are added to a queue. The first clause R' of this queue is taken and added to the base (intuitively, the database of all generated clauses). Then, using a heuristic, one of the literals of R' is selected. If the literal is in the hypothesis, new clauses are generated by replacing the selected literal F with

the hypothesis of other clauses $R = H \to C$ for which it holds that $C = F$[1]. Simplified, this unification of R and R' into R'' works as follows:

$$\frac{R = \mathrm{H} \to C \qquad R' = F \wedge \mathrm{H}' \to \mathrm{C}' \qquad C = F}{R'' = \mathrm{H} \wedge \mathrm{H}' \to \mathrm{C}'}$$

All newly generated clauses are added to the queue, and the saturation continues. Further, queries in the base and in the queue are only added if they represent new information (e.g., do not exist yet). The saturation continues until a fixpoint is reached, hence the queue is empty. Then, using the generated clauses in the base, the security properties are verified.

Unfortunately, this saturation may not always terminate with the default behaviour of ProVerif. Then, the generated clauses need to be inspected. Often, at least some of these clauses are unnecessary, and ProVerif can be guided to not generate them. For example, clauses may contain a fact F in the hypothesis that the adversary will never be able to generate. We can then use the so-called *secrecy assumption* `not` F to advise ProVerif to discard all clauses that contain F in their hypothesis. This helps with termination, while preserving soundness: ProVerif checks that the secrecy assumption is indeed fulfilled, i.e., there is no clause that derives F.

A reason for large clauses, possibly leading to non-termination, are the fresh names created by `new`. These names are abstracted, because ProVerif cannot reason over an infinite set of names. Instead, the fresh names are represented as terms with context, where the context is given by the value of the previous inputs located above the particular `new`. This context may however be the reason of non-termination, as terms where the same name has different context are treated as different terms (and therefore for example prevent de-duplication of clauses). Per default, ProVerif adds all inputs to the context. The user can then fine-tune which inputs are indeed considered in the context, in case this high precision is not necessary for the proof.

During saturation, ProVerif needs to decide which literal of a given clause it will select, to resolve against literals of the other clauses. All selection functions are proven sound, while their default heuristic may however lead to non-termination. In this case, the user can use `noselect` declarations. This declaration tunes the literals' selection bias in such a way that it is likely never going to be selected. Further, these declaration may additionally also be annotated with [`ignoreAFewTimes`], which enables to ignore the declaration for a predefined number of times, to e.g., allow selecting the literal once using the default heuristic. We let interested readers refer to [Bla23, Section 6.7.1] for more details.

[1] ProVerif is a bit more clever, and allows to consider C different from F as soon as both facts can be unified applying some substitution. We refer to [BCC22] for details.

A more recent update to ProVerif has added additional capabilities [BCC22]. This includes the addition of lemmas, which are user-provided queries that ProVerif is then applying during the saturation procedure. This enables the user to communicate to ProVerif early some properties that are expected to hold, to improve both precision and termination. This allows to declare some invariants of a model that ProVerif would otherwise only discover later (or not at all, and then not terminate or find false attacks). Lemmas preserve the soundness of the proof, as ProVerif first proves them, and only then leverages the sound lemmas to attempt to prove the main queries.

Together with the lemmas, axioms and restrictions have been introduced [BCC22]. Restrictions form a new convenient way of modelling. Instead of a process which describes precisely the protocol under verification, the defined process may over-approximate the set of execution traces of the protocol. Then, using a restriction, these traces are filtered to only keep the traces that can satisfy it and thus actually occur in the protocol. In contrast, axioms are lemmas that are assumed to be satisfied by the model under verification. Like lemmas, they can greatly improve ProVerif accuracy and termination by stating invariants about the model. However, they are not proven by ProVerif, and their soundness has to be established outside of ProVerif.

Generic techniques to improve precision of ProVerif have been proven sound, and we highlight here GSVerif-like [CCT18] axioms. These include the concept of cells and counters. Cells are private channels where each output is preceded by exactly one input, except for one which has no preceding input but is considered as the initialization of the cell. Counters are a type of cells that count up, i.e., store and read i, then $i + 1$, and so on. GSVerif-like axioms then reflect this behaviour precisely in ProVerif. For the model to be sound, usage of cells and counters need to correspond to the semantics of the actual protocol that is being modelled. We make heavy use of these techniques in the proof framework.

Some of the contributions of GSVerif are now integrated directly into ProVerif. For example, ProVerif by default overapproximates the inputs when transforming the processes to Horn clauses. Concretely, after the transformation, the semantics allow to execute an input of the same process arbitrarily often *with different values*. A GSVerif-like axiom can then be used to attest that the same input of the same process always has to the same value. This particular axiom has been proven sound [CCT18], and is now built in into ProVerif by attaching the [`precise`] option to the relevant inputs [BCC22]. Of course, this increase in precision may lead to models that no longer terminate.

Writing larger models Larger models are harder to write and review, and therefore to ensure they are sound. This needs to be counter-balanced with cleaner modelling approaches, as well as extended sanity checks.

ProVerif provides a simple type system. This includes some built-in types (such as `bitstring` and `nat`), but also user-defined types. It is good practice to use the type-system to spot simple modelling mistakes, e.g. passing a signature where a ciphertext would have been expected. To convert between types, users may define unary function symbols and annotate them with [`typeConverter`]. All types and type-converters are removed during the conversion to Horn clauses, hence have no impact on soundness of the proof. More complex data constructs (e.g., a `ballot` that is composed out of multiple values) can be represented using non-unary function symbols which are annotated with [`data`]. These data converters are then included in the equational theory as public and invertible function symbols.

Besides clean modelling, sanity checks are indispensable. To test basic properties of the model, one can use the secrecy assumptions to quickly verify whether the adversary has access to some value or not. Further, executability lemmas can be formulated. These lemmas can state some event to be reachable, or even a whole chain of events corresponding to the expected flow of the protocol. If these lemmas cannot be proven as expected, then the model is not executable. For equivalence proofs, another useful technique is to publish a message trivially breaking equivalence at some specific place in the model. If equivalence is then still proven to hold, then clearly that part of the model is not executable.

To scale to larger models, instead of proving directly a complex property, one can also start to prove simple lemmas first. Once they are proven, in the next step of the proof, these lemmas are then assumed as axioms. Then, more complex lemmas can be formulated and proven, and this process is repeated until the complex property is within reach. Important is also to keep the model small. For example, it is advisable to not model actions of the adversary, e.g. participants that are dishonest. This will generate additional clauses that are unnecessary, as ProVerif can use the equational theory to derive any term that dishonest participants could possibly derive.

4.2 Running example

We sketch a simple voting system, inspired from Helios [Adi08], which will serve as the running example in this work. We summarize the voting phase in Figure 4.1.

Setup phase The talliers jointly compute the public key of the election, formed by the product of the public keys of the talliers: each tallier has a share of the decryption key and is needed for decryption.

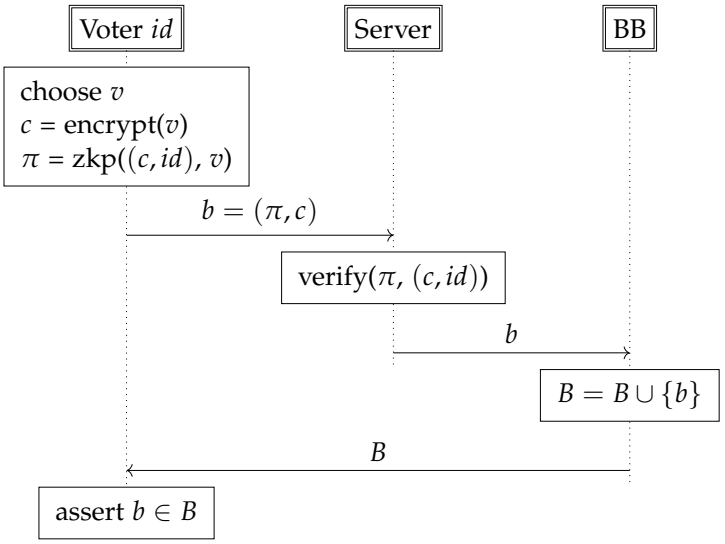

Figure 4.1: The voting phase of our running example. The voter forms their ballot b, and sends it to the server. The server verifies the zero-knowledge proof, and then publishes the ballot on the bulletin board. Lastly, the voter verifies that their ballot has indeed been stored on the bulletin board.

Voting phase The voter forms the ballot by encrypting their vote using the public key of the election, and attaching a zero-knowledge proof that binds the encryption to the voter identity. The ballot is then sent to the voting server over an authenticated channel (e.g., the voter signs the ballot using a national eID scheme). Afterwards, the voter verifies that the voting server has indeed published that exact ballot on the public bulletin board.

The voting server checks for each received ballot whether it is authenticated by an eligible voter, and that the zero-knowledge proof is valid. If both checks succeed, then the ballot is published on the public bulletin board, else it is discarded.

Tally phase A chain of mix servers — each operated by a tallier — shuffle and partially decrypt the ballots. The first mix server starts with the list of ballots as published on the public bulletin board and shuffles it. It then partially decrypts each ballot using its share of the election private key, and generates zero-knowledge proofs that this has been done correctly. Each mix server one after the other performs this process, taking as input (and verifying) the output of the previous mix servers in the chain. The last mix server publishes its output, which, after verification by the other mix servers, is attested to be the election result.

Such a chain of mix and decrypt servers is used for example in the Swiss Post Voting System [Swi22]. Another common option is to first shuffle the ballots using several mixers, and then decrypt the shuffled ballots. Alter-

natively, the ballots may be combined homomorphically and the resulting ciphertext is decrypted.

Security claims For this simple protocol, the voting server needs to be trusted for verifiability since it is in charge of authenticating the voters. It is possible to remove this assumption, for example by assuming an external Registrar that is in charge of generating voting credentials such as in Belenios [CGG19]. For vote privacy, voters need to trust the voting client and one of the several mix-servers.

4.3 Verifiability framework

The framework proposed in [CCD23] provides a flexible skeleton to integrate internet voting protocols, including a pre-built public bulletin board. Further, it defines generic lemmas and axioms which aid towards a precise and terminating proof.

4.3.1 Verifiability notion

The verifiability notion proven by the framework is the definition by Cortier et al [CGK+16]. Intuitively, this notion captures E2E-verifiability as fulfilled if the result of the election corresponds to:

- all votes of the honest voters that verified their vote,
- a subset of votes of the honest voters that did not verify,
- additional votes cast by corrupted (adversary-controlled) voters, bound by the number of corrupted votes.

Intuitively, the definition therefore needs to count, which would be difficult to prove for ProVerif (and other proof tools). To actually prove the definition, [CCD23] presents a characterization based on two injective queries. These two queries are shown to be equivalent to the E2E-verifiability notion.

```
1  query event(Finish(elec)) && inj-event(Counted(elec, v)) ==>
2    inj-event(HV(elec, id)) && event(Verified(elec, id, v))
3    || inj-event(HNV(elec, id)) && event(Voted(elec, id, v))
4    || inj-event(Corrupt(elec, id))
5
6  query event(Finish(elec)) && inj-event(Verified(elec, id, v)) ==>
7    inj-event(Counted(elec, v))
```

The first query states that whenever a vote `v` in an election `elec` is counted `Counted(elec, v)` , and the tally of that election is finished, then this vote must come from:

- either an honest voter (`HV`) that verified their vote `v`
- or an honest but non verifying voter (`HNV`) that voted `v`

- or a corrupted voter

This property is *injective* (with `inj-event`) to specify that any `Counted` event can be injectively mapped to an honest or a corrupted voter. This for example ensures that a corrupted voter cannot cast multiple votes.

The second query states that any verified vote is part of the tally, exactly as verified. Again, this is formulated using an injective event, so for honest verifying voters `HV`, this results in a bijection (i.e., all votes are counted as expected).

Both queries only need to hold once the election is fully tallied, i.e., `Finish` has been raised.

4.3.2 Model

To model a concrete protocol, the framework supports a total of 11 user-defined functions and processes for custom functionality. Further, it provides framework-defined channels and tables for storage. By these means, the provided skeleton internet voting system can be heavily customized, including the registration of voters, the casting and verification of ballots, the storage of these ballots on the bulletin board, as well as finally their decryption during the tally.

To demonstrate how to use the framework, we model the protocol introduced in Section 4.2. For visual clarity, this section differentiates framework-defined channels, tables and processes with a grey background.

Setup For simplicity, our running example models an authenticated channel to the server that can be derived using a function. Concretely, the private function `ch_auth` takes a `voter_id` as an argument, and then returns a `channel`. As `ch_auth` is private, the derived channels are, too.

```
fun ch_auth(voter_id): channel [private].
```

Therefore, we do not need to generate state per voter. Hence, when interfacing with the setup of the framework, we use the default implementations, which semantically do nothing.

```
(* nothing known by voter beforehand *)
letfun gen_initial_vd(elec: elec_id, id: voter_id) = ()

(* take over (empty) value from gen_initial_vd *)
let Gen_VD(elec: elec_id, id: voter_id, vd: voter_data) =
  insert voting_data(elec, id, vd)
```

Under the hood, the framework creates for every voter an entry in the framework-defined table identifiers.

Voting To model how the (honest) voter casts their ballot, we define the process `Vote`. It uses the election public key `pk(elec)` to encrypt the vote `v` into the ciphertext `c`. Then, it creates a zero-knowledge proof `p` of knowledge of the vote `v` and the encryption randomness `r` which binds `c` to the casting voter. The ballot `b = (c, p)` is sent over the private channel `ch_auth` to the voting server. We additionally publish `b` over the public channel `ch_pub` to model an authenticated channel.[2]

```
let Vote(elec: elec_id, id: voter_id, v: vote) =
  (* form ballot *)
  new r: random;
  let c = encrypt(pk(elec), v, r) in
  let p = zkp_encrypt(pk(elec), (c, id), (v, r)) in
  let b = (c, p) in

  (* cast ballot *)
  event Voted(elec, id, v);
  out(ch_pub, b); out(ch_auth(id), b);
  out(res_vote(elec, id), b)
```

The event `Voted(elec, id, v)` is used to specify security properties. For example, it will allow to state that any tallied vote `v` should come from a voter that indeed cast `v` (or from a dishonest voter). Further, we need to inform the framework about the result of the `Vote` process, by sending the ballot `b` on the framework-defined channel `res_vote`.

To model how the election server stores the ballot, we define a process `Server`. It queries the known voters from the framework-defined table `identifiers`, filtering by the active election `elec`. Then, it receives the ballot `b` sent over the authenticated channel `ch_auth` belonging to one of the known voters. Finally, it verifies the zero-knowledge proof, and if it is valid, adds `b` to the framework-defined table `ballot_box`.

```
let Server(elec: elec_id) =
  get identifiers(=elec, id: voter_id) in
  in(ch_auth(id), (c, p));
  if verify_zkp_encrypt(p, (c, id)) then (
    insert ballot_box(elec, (id, (c, p)))
  )
```

The framework then adds all ballots in the `ballot_box` to the bulletin board, modelled by private channels of the form `cell_BB(elec, i)`, with one per eligible voter `i` and election `elec`. Due to the simplicity of the protocol, the bulletin board functions are mostly using trivial implementations.

[2] ProVerif only distinguishes between private channels, which the attacker can neither read (confidentiality) nor write (authenticity), and public channels, where the adversary can both read and write. To model the authenticated channel for our running example, publish the ballot on the public channel (no confidentiality), and then send it over the private channel (authentication).

```
(* extract id from the tuple in ballot_box. *)
letfun get_id(elec: elec_id, b: ballot) =
  let (id,_) = b in id

(* check that ballots on the bulletin board have a valid ZKP *)
letfun is_valid(elec: elec_id, b: ballot) =
  let (id, (c, p)) = b in verify_zkp(p, (c, id))

(* store the full ballot on the bulletin board. *)
letfun update_BB(elec: elec_id, b: ballot) = b

(* leave the public part of the bulletin board empty. *)
letfun update_gBB(elec: elec_id, b: ballot) = empty
```

Once the ballot is on the bulletin board, the voter can verify whether their ballot has been processed correctly. Concretely, the verification checks whether the exact ballot as cast is contained on the bulletin board. Note that in our running example, the voting device is assumed trusted, and therefore the ballot is formed as intended.

```
let Verify(elec, id: voter_id, v: vote, b: ballot) =
  in(cell_BB(elec, id), =b);
  event Verified(elec, id, v);
  out(cell_BB(elec, id), b)
```

Tally To model decryption by the trustees, we define a `Decrypt` process. Given some ballot `b = (c, p)`, the election private key `sk(elec)` is used to decrypt the ciphertext `c` into the plaintext vote `v`. This `v` is then sent over the framework-defined channel `res_decrypt` back to the framework. The framework then collects all the plain votes after decryption, and finally publishes them to a public channel.

```
let Decrypt(elec: elec_id, id: voter_id, b: ballot) =
  let (c, _) = b in
  let v = decrypt(sk(elec), c) in
  out(res_decrypt(elec, id), v)
```

System Finally, we can construct the full voting system. For this, we use the macro system provided by ProVerif. Concretely, the framework defines a macro called `Framework`. We `expand` the macro by passing our protocol-specific functions and processes, and in turn get back the `System` process composed by the framework.

In the default process, we generate a fresh election `elec`, and then launch this `System` process. In parallel, we launch other processes modelled outside the framework (e.g., in our example the `Server` process). We write at the start of the default process a bang, which ensures ProVerif replicates the process, i.e., executes it an unbounded amount of time.

```
expand Framework(
  gen_initial_vd, Gen_VD, (* setup *)
```

```
47    Vote, Verify, Decrypt, (* Voter & Trustees *)
48    get_id, is_valid, update_BB, update_gBB, (* BB *)
49    System
50  )
51
52  process
53    ! new elec: elec_id; (* bang to repeat (unbounded) *)
54    System(elec) | !Server(elec)
```

Under the hood, via the `System` process, the framework boots multiple internal processes, e.g., for the registration, for the honest voters who vote and then verify, or for the tally. These processes then call the user-defined functions and processes at appropriate places with appropriate arguments, effectively combining the generic framework functionality with the protocol-specific user-defined functionality.

As an example, we give an intuition for the `Tally` process.

```
1  let Tally(elec: elec_id) =
2    ! in(cell_tally(elec), i: nat);
3    if i > 0 then (
4      in(cell_BB(elec, i), b: ballot);
5
6      Decrypt(elec, i, b) |
7      in(res_decrypt(elec, i), v: vote);
8
9      event Counted(elec, v);
10     out(ch_pub, v);
11     out(cell_tally(elec), i-1)
12   ) else (
13     event Finish(elec)
14   )
15 .
```

First, the process receives `i` on the `cell_tally` channel. At the very end of `Tally`, over this same channel $i - 1$ is sent out, modelling a decreasing counter. The initial message of the counter is the number of registered voters in the corresponding election, sent out by a preceeding process of the framework. Together with the replication bang, the process therefore counts down in `i` until it reaches 0.

For each voter `i`, the process then retrieves the corresponding ballot from the bulletin board channel `cell_BB`. This ballot is decrypted by invoking the user-defined `Decrypt` process, which responds with the decrypted vote `v` on the channel `res_decrypt`. Then, the `Counted` event is raised to mark `v` as counted. Finally, `v` is published on the public channel `ch_pub`. Then, the next iteration with $i - 1$ is started.

4.3.3 Limitations

The verifiability framework cannot be reused as such to prove privacy. Indeed, the tally counts votes one-after-the-other. This is a sound simplification in the context of verifiability since the order in which the ballots are

published does not matter. However, this simplification trivially breaks privacy: the adversary can easily attribute the plain votes to their respective voters by considering the order in which they are published. This is one important issue that we needed to overcome to adapt the framework to cover *both* verifiability and privacy.

4.4 Add support for vote privacy

To extend the framework towards vote privacy, we need a tally process that allows to model the privacy measures (e.g., mixing). We also need to model the vote privacy definition. Then, the second main task is to provide techniques to *prove* privacy. We first start with a simple tally and then show how to model complex tally procedures with several mixers, where only some of them are honest.

4.4.1 The privacy definition

Vote privacy guarantees that the vote of a voter is kept secret. This is typically expressed as an indistinguishability property [Ben87, KR05]: an adversary should not be able to distinguish the case where Alice votes 0 and Bob votes 1, from the converse scenario where Alice votes 1 and Bob votes 0.

$$Alice(0) \mid Bob(1) \approx Alice(1) \mid Bob(0)$$

We cast this definition in our framework as follows. First, our voters are numbered, and we choose Alice and Bob to be voter 1 and 2, respectively. Second, we define the two constants `voteA` and `voteB` to represent the two vote choices. The voter 1 and 2 therefore vote `diff`[`voteA`, `voteB`] and `diff`[`voteB`, `voteA`], respectively. This means that we ask the adversary to distinguish between the case where voter 1 chooses `voteA` and voter 2 chooses `voteB` from the converse case where voter 1 chooses `voteB` and voter 2 chooses `voteA`. Other voters and vote-choices are left as in the verifiability framework, that is, the adversary can freely decide which voters are honest and what they vote for. In particular, our voting framework supports an arbitrary number of honest and dishonest voters and votes. As in the verifiability framework, we support revotes. Revotes allow a voter to override a previously cast vote, e.g. to correct a wrongly cast vote, or a vote cast under coersion. The honest voters 1 and 2 will always vote `diff`[`voteA`, `voteB`] and `diff`[`voteB`, `voteA`], while all other voters may revote arbitrarily.

One of our primary targets is to be able to use the same model for both verifiability and privacy and to make minimal modifications to the verifiability framework in terms of protocol model. Therefore, we limit our modifications to the introduction of two functions and one process that are

defined differently depending on whether privacy or verifiability is considered. Namely, we introduce two functions `map_status` and `map_vote`. The first function is called by the framework when the adversary decides on whether a given voter is honest or not, and the second when the adversary decides on the vote to be cast for a given voter. By defining these functions differently for privacy and verifiability, we can implement the respective definition without changing any other part of the framework processes.

For privacy, `map_status` enforces that the voters 1 and 2 always have their status set to `honest`. Further, `map_vote` enforces that the voters 1 and 2 choose their vote according to the privacy definition. For all other voters, the functions return the attacker chosen status or vote, respectively.

```
letfun map_status(id: voter_id, attacker_s: nat) =
  if id = 1 || id = 2 then honest
  else attacker_s
.

letfun map_vote(id: voter_id, attacker_v: vote) =
  if id = 1 then diff[voteA, voteB]
  else if id = 2 then diff[voteB, voteA]
  else attacker_v
.
```

For verifiability, the attacker chooses the values freely, as before encoded in the verifiability framework.

```
letfun map_status(id: voter_id, attacker_s: nat) = attacker_s.

letfun map_vote(id: voter_id, attacker_v: vote) = attacker_v.
```

Besides these two functions, the only other difference between the privacy and verifiability model is a new process `HBallots`. For privacy, this process encodes that the ballots of the honest voters 1 and 2 indeed reach the bulletin board, which is necessary for vote privacy. For verifiability, the trivial implementation that semantically does nothing is again sufficient. We describe this process in detail in the next section.

4.4.2 Problem statement

We need to enrich the tally phase in order to model a variety of privacy-preserving tally processes. In particular, we want to be able to model both corrupted and honest parties during the tally and several types of mixnets, for example onion decryption mixnets or pure shuffling followed by threshold decryption. We note that this is already richer than many existing models where the tally is often abstracted away, even for privacy.

The main issue is to guide ProVerif in order to prove vote privacy. This needs to be expressed as an equivalence property, but due to its internal proof strategy, ProVerif has issues in proving even simple equivalences

where outputs are made in parallel (concurrently). The reason is that while ProVerif aims to prove observational equivalence, this is difficult, and instead ProVerif proves the stronger notion diff-equivalence [Bla23]. Intuitively, diff-equivalence is stronger in the sense that it cannot reason over multiple (parallel) processes at the same time, but instead requires each action on one side of the equivalence to always be matched with the exact same action on the other side. While this has been shown to imply observational equivalence [BS16], it is too strong in our case. As it defeats the modelled parallelization, for ProVerif the actions happen sequentially, and equivalence is trivially broken. We elaborate on this issue and how to circumvent it in the rest of this section.

Secondly, the core reason why a voting protocol preserves vote privacy is that the ballots of Alice and Bob are encrypted as long as the adversary can track them, and they are decrypted only after destroying this link (e.g., by cryptographically shuffling the ballots). We therefore need to follow the journey of a ballot, from the voter to the final decryption, in order to let ProVerif reason that vote privacy is indeed preserved. This necessary precision is difficult to balance with termination, and we will elaborate on this in Section 4.5.

4.4.3 Model a simple tally

The original framework publishes tallied votes one-after-the-other, which trivially breaks privacy. We need to model that votes are shuffled before being published, which is usually done by publishing votes in parallel. This also maps directly to real-world systems, which typically announce the voting result as a set of (aggregated) votes. Syntactically, this is easy to implement into the framework. The `Tally` process, instead of decrypting and publishing each vote `v` sequentially for each voter `i`, can do this in parallel.

```
10    out(ch_pub, v) | (* semicolon replaced by pipe *)
11    out(cell_tally(elec), i-1)
```

Besides the parallel tally, we need to handle an additional aspect: vote privacy is trivially broken if the adversary can block votes. Concretely, the adversary may block all but the vote of Alice, and then learn the vote of Alice from the result. Cast to our vote privacy definition, it implicitly assumes that the ballots of voters 1 and 2 are always counted. To make this explicit in our model, we introduce a private channel `res_vote_ballot` that

- records the ballots formed by voters by sending it out over the channel `out(res_vote_ballot(elec, id), b)` in the user-defined `Vote` process,
- magically passes this information to a new process `HBallots` that ensures that the ballots of voters 1 and 2 are transmitted to the tally.

The process HBallots is defined as follows for privacy. It receives the ballots from the honest voters 1 and 2 over res_vote_ballot. Then, it checks that the bulletin board, modelled by the cell_BB channels, indeed stores these ballots. Finally, it forwards all ballots on the bulletin board (cell_BB) over to the new channel cell_BB_checked. For verifiability, HBallots is the trivial implementation, i.e., simply forwards all ballots from cell_BB to cell_BB_checked.

```
let HBallots(elec: elec_id) =
  (* get the honest ballots *)
  in(res_vote_ballot(elec, 1), b1: ballot);
  in(res_vote_ballot(elec, 2), b2: ballot);

  (* ensure ballots are on the bulletin board *)
  in(cell_BB(elec, i1), =b1));
  in(cell_BB(elec, i2), =b2));

  (* record the two indices *)
  insert honest_voters_index(elec, i1, i2);

  (* forward all ballots to the tally *)
  ! get identifiers(=elec, i) in
  in(cell_BB(elec, i), b: ballot);
  out(cell_BB_checked(elec, i), b)
.
```

Then, we adapt the Tally process correspondingly. Concretely, we read from the new cell_BB_checked instead of cell_BB. The rest of the process does not change, hence the ballots are then decrypted, and finally published.

```
let Tally(elec: elec_id) =
  ! in(cell_tally(elec), i: nat);
  if i > 0 then (
    in(cell_BB_checked(elec, i), b: ballot);

    Decrypt(elec, i, b) |
    in(res_decrypt(elec, i), v: vote);

    event Counted(v);
    out(ch_pub, v) |
    out(cell_tally(elec), i - 1)
  )
.
```

From a pure modelling perspective, this is all we need to define a privacy-preserving tally. However, as mentioned before, ProVerif attempts to prove diff-equivalence [Bla23]. Therefore, the modelled parallelization is ineffective, and the processes are checked to behave the same *step-by-step*. To illustrate the problem, we can focus on what the Tally process output on the public channel ch_pub corresponds to. For the sake of this example, we assume there are only the two honest voters 1 and 2, and replace the variable v with their respective vote. This leads to the following process:

```
out(ch_pub, diff[voteA, voteB]) |
out(ch_pub, diff[voteB, voteA])
```

When it eventually executes, the adversary can observe that voteA and voteB are different values. For example, the adversary can compare the message send out on ch_pub to diff[voteA, voteA], a public constant known to the adversary. This will succeed on the left side but not on the right, and therefore break (diff-)equivalence.

To circumvent this well known issue, a standard technique consists in swapping terms in between the processes, for parts that are executed in parallel. This has been proven sound by Blanchet and Smyth [BS16]. Hence, this technique allows us to sidestep the limitation of diff-equivalence in this case. To apply this to our framework, all what we need to do is to replace lines 9-10 of process HBallots by

```
 9    out(cell_BB_checked(elec, i1), diff[b1, b2]) |
10    out(cell_BB_checked(elec, i2), diff[b2, b1]) |
```

This concludes the extension to privacy for simple tally procedures and the necessary developments to avoid the diff-equivalence tightness. Further, as it is fully backward compatible, the models written for the verifiability framework directly translate. More complex internet voting protocols however employ a so-called *verifiable shuffle* to anonymise the votes, which unfortunately interferes with our current tally modeling. We show in the next section how to further extend our framework to complex tallies.

4.4.4 Modelling a verifiable decryption shuffle

Verifiable shuffles typically take as input a list of ciphertext, shuffle it, and then re-randomize or (partially) decrypt each individual element. This allows to break the link between input and output elements. The shuffles are made verifiable by producing zero-knowledge proofs that the plaintext of the elements have not been altered. This is not only important for correctness of the shuffle, but also for its privacy: For example, in an unverified shuffle, an adversary might remove all but one ciphertext, trivially exposing its plaintext after decryption.

A verifiable shuffle is typically executed by multiple components one-after-the-other, as depicted in Figure 4.2. This allows to distribute the trust between the components: as long as one component is honest, the shuffle effectively anonymizes the elements. Intuitively, privacy and verifiability are ensured despite the existence of dishonest shufflers through three steps:

1. the dishonest shufflers that play first may only return a list of ciphertexts whose plaintext correspond the initial list of ciphertexts, thanks to zero-knowledge proofs. This preserves verifiability and also privacy since the votes of all voters (including 1 and 2) are kept.
2. at least one honest shuffler honestly permutes the ciphertexts, which prevents the adversary from tracking the votes and preserves privacy.

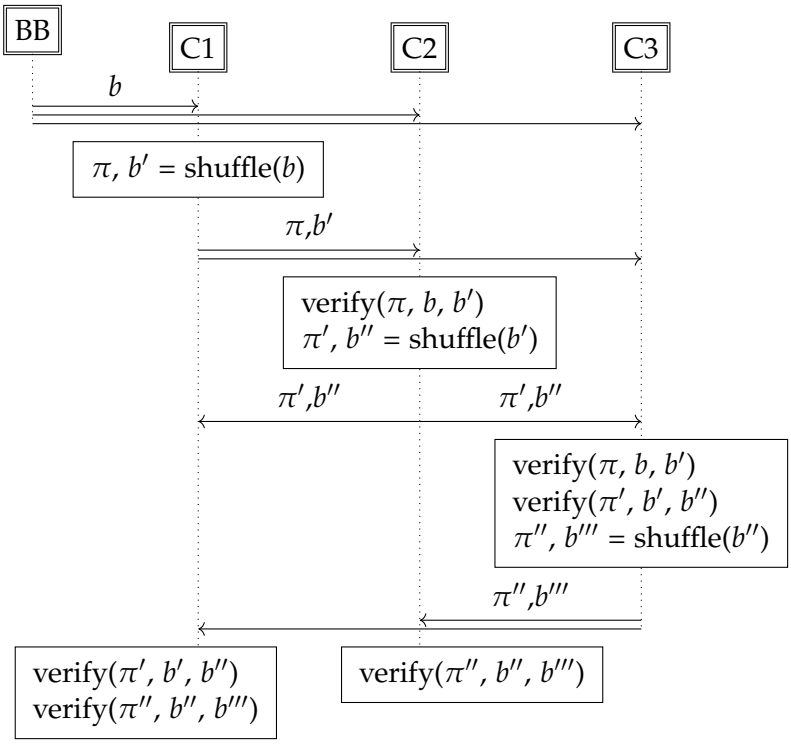

Figure 4.2: Verifiable shuffle with three components C. They receive the list of ballots b from the bulletin board BB. Each then performs a verifiable shuffle which rerandomizes or partial decrypts each ballot, and changes their order in the list, while producing a proof π that this has been done correctly. Each component verifies the execution of the other components.

3. finally, the dishonest shufflers that play last may not change the set of plaintexts, again thanks to the zero-knowledge proofs, which preserves verifiability.

From a modelling perspective, the shuffles performed by the adversary during Steps 1 and 3 strongly interferes with our current tally. Indeed, the honest ballots are (legitimately) modified by the dishonest shufflers. However, the framework needs to be able to continue to track the individual ballots that are part of the tally. This is essential not only to prove the correspondence queries required for verifiability, but also to rule out false privacy attacks that could arise from losing track of honest voters' ballot.

To tackle the modelling of the verifiable shuffles, we split the mixing and tally procedure as illustrated in Figure 4.3.

First, we introduce a new framework process `Pre_Tally`. It reads all ballots from the bulletin board, and then launches for each ballot the new user-defined `Pre_HBallots` process.

```
1  let Pre_Tally(elec:elec_id) =
2  ! get identifiers(=elec, i) in
```

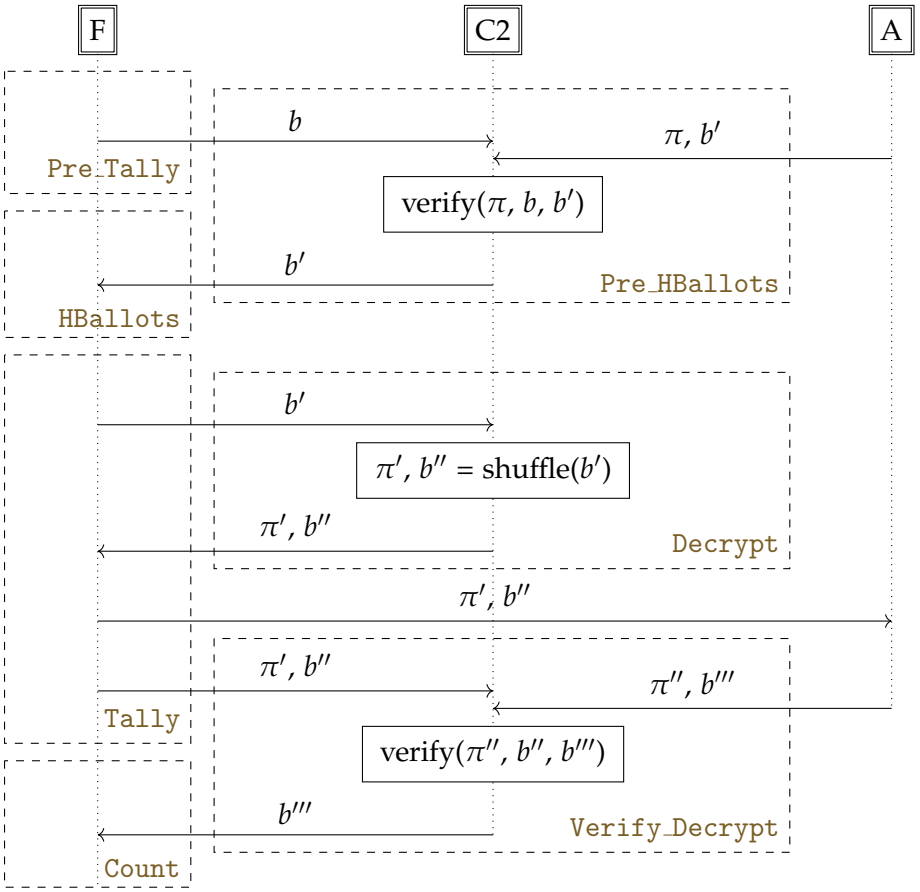

Figure 4.3: Modelling of a verifiable shuffle with honest component C2, visualizing the interactions with the framework F and the adversary A (which notably runs the untrusted components C1 and C3). C2 performs the same actions as in fig. 4.2, while we visualize how these actions are split between the different processes. `Pre_HBallots`, `Decrypt` and `Verify_Decrypt` are user-defined processes, while the others are provided by the framework. Newly introduced for privacy are the upper three processes, which together allow to implement the privacy definition and handle the swap-related challenges.

```
3  in(cell_BB(elec, i), b: ballot);
4
5  Pre_HBallots(elec, i, b)
```

The `Pre_HBallots` process is in charge of reading the ballots produced by the dishonest shufflers. Using the zero-knowledge proofs, this processes attributes the permuted ballots back to their respective index on the new channel `res_pre_hballots`. Then, the `HBallots` only forwards these ballots if the honest ballots have indeed been part of the tally. Concretely, we can replace the last lines of the `HBallots` process with:

```
13 (* forward all ballots to the tally *)
14 ! get identifiers(=elec, i) in
15 in(res_pre_hballots(elec, i), b: ballot);
```

```
16  out(cell_BB_checked(elec, i), b)
```

Then, as before, the Tally continues on the ballots it receives over the channel cell_BB_checked. The decryption or re-randomizing step performed by the honest shuffler, as well as the shuffling step, is implemented in the user-defined Decrypt process. The Decrypt process no longer computes directly the result in clear but computes instead some partial decryption, output on channel res_pre_decrypt.

We now differentiate between the Tally process, in charge of the honest contributions of the tally process and a new Count process in charge of announcing the result, possibly getting some data from an untrusted source. This is for example the case when the final shufflers are dishonest.

The Tally process now reads the ballot box, launches the Decrypt process, retrieves its result and stores it into a table log_i_c. When all the ballots are processed, i.e., when the decreasing loop on channel cell_tally(elec) reached 0, all the stored information in the table log_i_c is published and passed to the final Verify_Decrypt process which will determine the result from the adversarial contributions and write it on res_decrypt. These actions are performed in parallel to avoid assuming a specific order of execution between them.

```
6   let Tally(elec: elec_id) =
7     ! in(cell_tally(elec), i: nat);
8     if i = 0 then (
9       ! get log_i_c(elec,i,c) in
10      (
11        out(ch_pub,c) |
12        Verify_Decrypt(elec, i, c)
13      )
14    ) else (
15      in(cell_BB_checked(elec, i), b: ballot);
16
17      Decrypt(elec, i, b) |
18
19      in(res_pre_decrypt(elec, i), c:bitstring);
20      insert log_i_c(elec,i,c);
21      out(cell_tally(elec), i-1)
22    )
23  .
```

Finally, the Count framework process announces the result by reading it on res_decrypt and raising Counted(elec,v) events.

```
24  let Count(elec: elec_id) =
25    ! in(cell_result(elec), i:nat);
26    if i = 0 then event Finish(elec)
27    else (
28      in(res_decrypt(elec, i), v:vote);
29      event Counted(elec, v);
30      out(cell_result(elec), i - 1)
31    )
32  .
```

The channel cell_result is a private channel that allows to emulate a loop on all voters, similarly to the cell_tally channel.

We illustrate this on an example.

4.4.5 Example

We continue our running example Helios model, assuming a verifiable shuffle with three components, the middle one being honest. The first (dishonest) shuffler may provide an arbitrary list of ciphertexts as output, provided that it can also provide a valid zero-knowledge proof of correct mixing. We assume here that the list formed of the ballots that appear on the ballot box has been signed by an authority with verification key vk_auth. This guarantees that all the shufflers agree on the same initial list.

```
let Pre_HBallots(elec: elec_id, i: nat, b: ballot)
  let (c, _) = b in
  out(ch_pub, c);
  in(ch_pub, l0, l1, zkp, s);
  if verify_sign(l0, s, vk_auth) &&
    valid_mix(l0, l1, zkp) &&
    c = get_list(l0, i)
    then (
      let c' = get_list(l1, i) in
      out(res_pre_hballots(elec, i), c')
  )
.
```

where the function get_list(l, i) returns the i-th element of the list l, and the functions verify_sign and valid_mix check respectively the validity of the signature s and of the zero-knowledge proof of correct mixing zkp. Since we consider an arbitrary number of voters, defining such a zero-knowledge proof of correct mixing would require to check equality of the plaintexts of lists of ballots of arbitrary size. This seems difficult to encode without AC (associative and commutative) symbols, that are currently out of scope of ProVerif.

Instead, we hardcode that there is a bijection between the ballots of the bulletin board and the ones returned by the first dishonest shuffler. Indeed, the zero-knowledge proof of correct mixing [Wik04, Gro10, BG12] ensures that the content of the ciphertexts has not been modified: the multiset of votes contained in the initial list l_0 is equal to the multiset of votes contained in the list l_1 returned by the dishonest shuffler. Hence, we know that there exists such a bijection between the ciphertexts of l_0 and of l_1 such that the corresponding plaintexts are equal. So we consider instead the following Pre_HBallots process.

```
let Pre_HBallots(elec: elec_id, i: nat, b: ballot)
  let (c, ) = b in
  out(ch_pub, c);
```

```
70
71    (* associate output of first shuffler *)
72    in(ch_pub, (c': ciphertext_t));
73    if recognize_partial_decrypt(c, c') then (
74      out(res_pre_hballots(elec, i), wrap_m_ballot(c'))
75    )
76  .
```

The function `recognize_partial_decrypt` magically associates each input ciphertext `c` with the output ciphertext `c'` by decrypting `c` and attributing `c'` to the cell initially corresponding to `c`. This enforces that ballot content are not modified and that the framework can track ballots.

Then the second (honest) shuffler mixes the ballots (in `HBallots` by sending all ballots in parallel), decrypts and returns the return to the `Tally` that will forward the result to the attacker.

```
77  let Decrypt(elec: elec_id, i: nat, b: m_ballot)
78    let wrap_m_ballot(c') = b in
79
80    (* perform decryption as second shuffler *)
81    let c'' = decrypt(sk(elec, 2), c') in
82    out(res_decrypt(elec,i), c'')
83  .
```

The third shuffler is the last one and it returns the votes in clear. The second shuffler waits for its answer and checks the zero-knowledge proof. We again do not model this proof and assume instead that there is a bijection between the input ciphertexts and output cleartext, yielding the following `Verify_Decrypt` process.

```
84  let Verify_Decrypt(elec: elec_id, i: nat, c'':bitstring)
85    (* associate output of third shuffler *)
86    in(ch_pub, v: vote);
87    if recognize_partial_decrypt(c'', v) then (
88      out(res_count(elec, i), v)
89    )
90  .
```

We can similarly consider the case where the honest shuffler plays first or last. For these cases, the only difference is that the invocation of the function `recognize_partial_decrypt` moves as expected. Concretely, if the honest shuffle plays first, both function invocations move after its decryption; and the inverse if the honest shuffle plays last. We provide implementations thereof as part of this work.

4.4.6 Limitations

Our framework can cover a variety of mixnet-based tally operations but does not model homomorphic tally. Indeed, to express an homomorphic tally, it would be necessary to specify an equational theory that expresses that the

multiplication of two encryptions yields the encryption of the addition of the respective plaintexts:

```
equation forall pke, m1, m2, r1, r2;
  encrypt(pke, m1, r1) * encrypt(pke, m1, r2)
    = encrypt(pke, m1+m2, r1+r2)
```

where $*$ and $+$ are two AC symbols. Such an equation is however out of the scope of ProVerif at the moment. More generally, our framework cannot cover complex tally systems such as tally-hiding computations [KLM+20, CGY22].

Moreover, as mentioned in Section 4.4.5, we cannot fully capture the zero-knowledge proof of correct mixing, since this would also require AC symbols.

4.5 How to prove privacy?

Attempting a proof over the model as proposed is not yet possible (even with a simple tally), due to both a lack of precision and non-termination. We therefore make sound transformations of our framework, in order to prove privacy, without changing the semantics of our model. This was already the case for proving verifiability. Indeed, the verifiability framework comes with a library of 40+ lemmas and axioms to aid termination. A large part of these lemmas are inspired from GSVerif [CCT18] that generates axioms that avoid some of the overapproximations done by ProVerif. For the privacy proof, we identify which lemmas and axioms are still useful and we develop a few new ones. Finally, we need to write a protocol-specific lemma that helps ProVerif to track the honest ballots from their emission to the bulletin board, in order to prove privacy.

4.5.1 Transformations of the verifiability framework

The verifiability model proves injective properties, hence needs to be very precise. Intuitively, it needs to track that all votes in the tally have not been modified, and that the verified votes indeed all reached the tally. As a basic building block to reach this precision, the framework extensively uses channels to count (e.g., see `cell_tally`) and to store state (e.g., see `cell_BB`). To increase their precision, they are often annotated with GSVerif-like events or the [`precise`] option, additionally to some custom overall lemmas [CCD23].

For example, the `Tally` process is annotated as follows.

```
let Tally(elec: elec_id) =
  !
  in(cell_tally(elec), i: nat);
  let () = PH_counter_change(count_tally(elec), i) in
```

```
5
6     if i > 0 then (
7       in(cell_BB(elec, i), b: ballot);
8       let () = PH_synch_value(count_BB(elec, i), b) in
9       event FinalBallotTally(elec, i, b);
10
11      Decrypt(elec, i, b) |
12      in(res_decrypt(elec, i), v: vote) [precise];
13
14      event Counted(v);
15      out(ch_pub, v);
16      out(cell_tally(elec), i-1)
17    )
18  .
```

Functions such as PH_counter_change and PH_synch_value raise events that are then used in GSVerif-like axioms. For example, it is modelled that cell_BB is a cell. For this, we generate an identifier for the cell using count_BB(elec, i). Then, by PH_synch_value, the event SyncValue is raised, which takes this cell identifier and the content of the cell as an argument. Finally, a GSVerif-like axiom enforces that if two SyncValue events referring to the same cell are found in the same trace, then then their content must be equal. Concretely, the axiom looks as follows.

```
1  (* axiom for reachability *)
2  axiom
3    event(SyncValue(cell, b)) &&
4    event(SyncValue(cell, b'))
5      ==> b = b'
6  .
```

Besides the GSVerif-like structures, some custom lemmas are declared. For example, the event FinalBallotTally is used to write a custom lemma that guides ProVerif to only consider a single ballot processed per voter. Finally, the [precise] option at res_decrypt prevents overapproximations. Concretely, the [precise] enforces that the channel may only receive a single value per invocation.

Importantly, these transformations are correct hence adding these events and axioms does not change the semantics of the reference model [CCD23]. A large part of the work of [CCD23] consists in identifying useful (and sound) axioms, as well as developing a library of lemmas, that are then used as proof helpers.

4.5.2 Architecture of the voting framework

We start from a reference framework, that models a generic voting system as sketched in Sections 4.3 and 4.4. The user-defined processes and functions are left unspecified since they need to be populated for each considered voting protocol.

For privacy, we transform the reference model as follows:

- swap ballots in the `Tally` process (see Section 4.5.3) to avoid false attacks due to the fact that ProVerif proves a too strong notion of equivalence.
- we remove the `Count` process as it only contains non observable actions (see [CCDM26, Proposition 3, Appendix D]).
- use some GSVerif-like axioms, whose soundness has been already been proved in [CCD23],
- design some lemmas, in particular the CellBB lemma (see Section 4.5.5), that states an important invariant for proving privacy,

We summarize our approach in Figure 4.4.

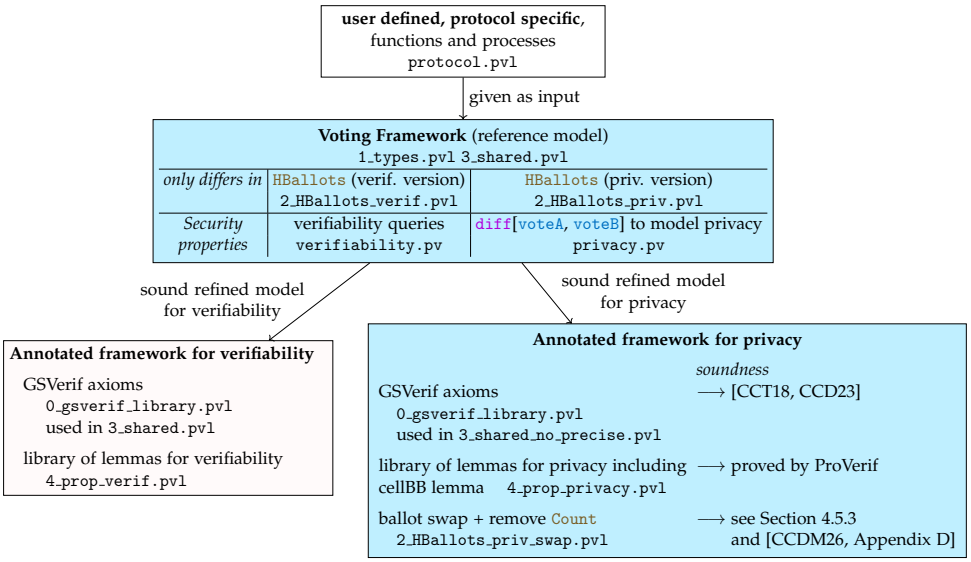

Figure 4.4: Architecture of the voting framework. In grey : previous verifiability framework [CCD23]. In blue : our work. `files.pv(l)`: corresponding ProVerif file.

4.5.3 Ballot swap

ProVerif does not prove observational equivalence, but the stronger notion diff-equivalence [Bla23], which checks whether the processes behave the same *step-by-step*. This defeats the fact that we model mixnet-based tally through parallel output of decrypted votes. In particular, ProVerif cannot prove:

```
out(ch_pub, choice[voteA, voteB]) |
out(ch_pub, choice[voteB, voteA])
```

because it will try to match the first output on the left with the first output on the right.

To circumvent this well known issue, a standard technique consists in swapping terms in between the processes, for parts that are executed in parallel. This has been proven sound by Blanchet and Smyth [BS16]. Hence, we replace the lines

```
 9  ! get log_i_c(elec,i,c) in
10  ( out(ch_pub,c) | Verify_Decrypt(elec, i, c) )
```

of process `Tally` by

```
 9  get honest_voters_index(=elec,i1,i2) in
10  (
11    ! get log_i_c(=elec,=i1,c1) in
12      get log_i_c(=elec,=i2,c2) in
13        out(ch_pub,diff[c1,c2]) |
14        Verify_Decrypt(elec, diff[i1,i2], diff[c1,c2]) |
15        out(ch_pub,diff[c2,c1]) |
16        Verify_Decrypt(elec, diff[i2,i1], diff[c2,c1]) |
17  ) | (
18    ! get log_i_c(elec,i,c) suchthat i <> i1 && i <> i2 in
19    ( out(ch_pub,c) | Verify_Decrypt(elec, i, c) )
20  )
```

If the index does not correspond to indices `i1` or `i2` of the ballots of the two honest voters, the behavior is unchanged. Otherwise, we retrieve the two ballots of the honest voters, and swap them.

Intuitively, this swap is sound because the processes `Verify_Decrypt` and the outputs are all launched in parallel, hence

```
 9  out(ch_pub, c1) |
10  Verify_Decrypt(elec, i1, c1) |
11  out(ch_pub, c2) |
12  Verify_Decrypt(elec, i2, c2)
```

is equivalent to

```
 9  out(ch_pub, c2) |
10  Verify_Decrypt(elec, i2, c2) |
11  out(ch_pub, c1) |
12  Verify_Decrypt(elec, i1, c1)
```

A formal proof of soundness of this swap is provided in [CCDM26].

4.5.4 Balancing precision and non-termination

Axioms for privacy For privacy, the precision developed for proving verifiability is not only largely unnecessary, it even leads to a non-terminating proof. The diff-equivalence attacker needs to explore all execution traces, and check that they perform the same actions in both worlds. If branches of the proof are not cut off early (e.g., because precision measures prevent it), the proof therefore quickly explodes in size, and termination becomes unlikely.

We therefore ignore most of the GSVerif axioms and events for the privacy proof. We however need to keep a GSVerif-like event at a crucial place: over the cell_BB channel modelling the bulletin board. This prevents that two different ballots are contained in the same cell, which is impossible by construction of our model. Otherwise, the adversary finds a false attack with the honest ballots: While the swap is applied to one ballot, the other ballot(s) in the same cell remain untouched and are published. The diff-adversary can use this to again trivially distinguish the worlds. While this is exactly the case for which GSVerif was built, reintroducing the GSVerif-like events for all cells on the bulletin board is however too coarse-grained, and again leads to non-termination. Instead, we may raise GSVerif-like events only for the relevant ballots, i.e., the ballots of honest voter 1 and 2.

```
in(cell_BB(elec, i), b: ballot);
if i <= 2 then (
  let () = PH_synch_value_bi(count_BB(elec, i), b) in
)
```

We also adapt the GSVerif-like axioms to diff-equivalence. In our traces, the content on the left and on the right side of the equivalence may not always be equal (e.g., for the honest ballots). The naive GSVerif-like axioms are therefore not always applied, as their precondition implicitly requires this equality. Hence, we need to adapt the GSVerif-like axioms to consider that the left and the right value may be different. We can do this with the diff operator, and using different variables left and right.

```
(* GSVerif-like axiom for equivalence *)
axiom
  event(SyncValueBi(cell, diff[bL, bR])) &&
  event(SyncValueBi(cell, diff[bL', bR']))
  ==> bL = bL' && bR = bR'
.
```

The soundness of this axiom, simply stated for bi-processes, follows directly from its soundness for standard processes [CCD23].

Lemmas to guide natural numbers and equalities Another source of non-termination are overapproximations around natural numbers and equalities. They lead to generation of traces where the cell itself is different in the left and right side of the trace. Further, some traces include cells that are not indexed by natural numbers, but arbitrary terms. This leads to an explosion of possible traces, where notably the GSVerif-like axioms are no longer applied (as the cells are not equal in both worlds). We guide ProVerif to avoid these overapproximations with corresponding lemmas over the cells. For example, we enforce for the cell cell_BB(elec, i) that elec and i are equal left and right, and that i is a natural number.

```
lemma
  mess(cell_BB(diff[eidL, eidR],  i), diff[bL, bR])
```

```
3       ==> eidR = eidL;
4     mess(cell_BB(elec, diff[iL, iR]), diff[bL, bR])
5       ==> is_nat(iL) && is_nat(iR) && iL = iR
6   .
```

Guide resolution algorithm After all these changes, ProVerif will still loop on counters and cells. We therefore need to prevent the resolution algorithm from picking these terms. This can be done by adding appropriate `noselect` statements. This changes how the resolution algorithm applies the heuristic of picking terms, without breaking its soundness. Concretely, we can tell ProVerif to avoid selecting `mess` facts (representing sent messages) that were produced by the counter and cell channels.

```
1  noselect mess(cell_tally(elec), diff[nL, nR]).
2  noselect mess(cell_BB(elec, i), diff[bL, bR]).
```

However, by preventing the resolution on counters and cells, the model becomes very imprecise. Concretely, ProVerif can no longer relate the ballots cast by the voters to the ballots on the bulletin board. We therefore need a way to encode the relevant invariants for ProVerif. More precisely, we need to enable ProVerif to track the ballots cast by the honest voters, as those are the ballots that might allow the diff-equivalence adversary to separate the two worlds. If the corresponding traces of these ballots can be explored exhaustively by the adversary, our proof will terminate and potentially succeed. We approach this by writing a custom lemma, that we describe in detail in the next section.

4.5.5 The cellBB lemma

To help ProVerif establish a link between the ballots cast by the honest voters 1 and 2 and those on the bulletin board, we write a lemma over `cell_BB`.

The lemma unpacks the plain vote from the ballot. The plain vote is then asserted to be either an attacker generated value, that has been generated *before* the `cell_BB` message, or to belong to the honest voters 1 and 2.[3] To formulate the former, we use an `attacker` fact (representing attacker knowledge), and require this attacker knowledge to have existed before the `mess`(`cell_BB`) fact. For the lemma to be effective, the ballot must include some cryptographic structure that binds the ballot to the concrete bulletin board cell. This can for example be a zero-knowledge proof which includes some voter identifier `id`. Then, with a mapping of `id` to the index `i` of the `cell_BB`(`elec`, `i`) cell, this strong link can be established by ProVerif.

For our running example, the lemma looks like this:

[3] Note how this lemma assumes that the bulletin board stores the ballots of the honest voters 1 and 2 in a predictable cell. While this assumption is true for all our case studies, it might not be true for all secure protocols.

```
mess(cell_BB(elec, i), diff[bL, bR])@i ==>
  (bL = empty && bR = empty) (* either empty... *) ||
  (
    id = get_vid(i) &&

    (* ...or correctly formed ballot in that cell... *)
    (i, (cL, pL)) = bL &&
    (i, (cR, pR)) = bR &&

    (* ...encrypting some vote... *)
    cL = encrypt(pk(elec), vL, rL) &&
    cR = encrypt(pk(elec), vR, rR) &&

    (* ...cryptographically bound to the id...*)
    pL = zkp_encrypt(pk(elec), (cL, id), (vL, rL)) &&
    pR = zkp_encrypt(pk(elec), (cR, id), (vR, rR)) &&

    (
      (* ...with the vote known to the attacker, *)
      (* notably known before the cell_BB message was formed,... *)
      (attacker(diff[vL, vR])@j && i > j) ||

      (* or from the honest voter 1 and 2. *)
      (i = 1) || (i = 2)
    )
  )
.
```

This lemma is however complex, even for the comparatively simple ballots used by our mock example. Notably, the ballot structure has to be defined twice, once for the left and once for the right world. As this ballot structure is further protocol dependent, the lemma must be written by the user of the framework. However, to write the lemma, the user would need to understand the precise internal workings of the framework, which is something we have been able to avoid until now. We improve on this by adding a new feature to the lemmas of ProVerif, and also slightly change the framework, as described in 4.6. These changes enable that the lemma can be written protocol-independent, hence we can again hide its complexity from the user.

4.6 Extending ProVerif

4.6.1 A generic cellBB lemma

The cellBB lemma presented in the last section is complex. However, the actual protocol-specific part only concerns the extraction of the vote. This extraction could also be formulated as a destructor `extract_vote` which rewrites a given ballot `b` into the vote `v` it represents. For our running example, such a destructor would look as follows.

```
fun extract_vote(ballot, elec_id, voter_id): vote
  reduc
    let c = encrypt(pk(elec), v, r) in
    let p = zkp_encrypt(pk(elec), (c, id), _(v, r)) in
    let b = (c, p) in
```

```
6       extract_vote(b, elec, id) = v
7     [private]
8   .
```

Further, the function `get_vid`, which attributes the cell index `i` to the voter identifier `id`, is mostly a technicality of the framework. In the verifiability framework, this attribution can be customized to the protocol with the custom process `Gen_Voter_Ident`. However, most models do not use this capability, and opt for the trivial implementation where `id = i`). If there is a need, this mapping can also be moved to the custom function `get_ident_from_ballot`. We therefore decide to get rid of `Gen_Voter_Ident` entirely, and thereby alleviate the need for `get_vid`.

So, without the need for `get_vid`, and given an implementation of the destructor `extract_ballot`, the cellBB lemma can be formulated protocol-independent. As such, it can now be provided directly by the framework, as no user customization is required anymore.

```
1  mess(cell_BB(elec, i), diff[bL, bR])@i ==>
2    (bL = empty && bR = empty) ||
3    (
4      vL = extract_vote(bL, elec, i) &&
5      vR = extract_vote(bR, elec, i) &&
6      (
7        (attacker(diff[vL, vR])@j && i > j) ||
8        (i = 1) || (i = 2)
9      )
10   )
11 .
```

However, destructors such as `extract_vote` in lemmas are not supported in ProVerif. Further, for more complex ballots, instead of a destructor, a `letfun` would be even more convenient, as they allow conditionals and variable assignments. As an example of an artificial `letfun`, we may think of a protocol where the ballots of some voter 3 always contain `voteA`. This can then be formulated using `if` / `else` as follows (reusing the destructor `extract_vote` from above).

```
1  letfun extract_vote'(b: ballot, elec: elec_id, id: voter_id) =
2    if id = 3 then voteA
3    else extract_vote(b, elec, id)
4  .
```

We therefore add support for both destructors and `letfun` to the conclusion of queries of ProVerif, which then allows their use also in the conclusion of lemmas, axioms and restrictions. This is of general interest, and contributes to writing concise and comprehensible lemmas for all uses of ProVerif. In the following section, we provide an informal description of our transformation. We formally prove the soundness of it in [CCDM26, Appendix C].

4.6.2 Add destructors and letfun to queries

In ProVerif, only constructor function symbols can appear in queries, such as `encrypt`. One of the reasons is that the semantics of queries and the algorithm for verifying them rely exclusively on unification and equality between messages, modulo the equational theory (excluding the rewrite rules). Destructor function symbols, defined solely through rewriting rules, therefore cannot appear in clauses, and, by consequence, queries, lemmas, axioms, or restrictions.

Adding destructors When a destructor occurs in a fact of the conclusion of the query, such as in events, we extend the semantics of queries by only considering the possible instantiations of the destructor's arguments that produce a message, that is, cases in which a rewriting rule can be applied to *reduce* the destructor symbol, i.e., make it disappear. When no such rule exist, we say that the term evaluation fails. For instance, observe how the following query ψ is rewritten, using the `reduc` rewriting rule for `decrypt`.

```
reduc forall e, m, r;
  decrypt(sk(e),encrypt(pk(e),m,r)) = m.

query x,y; event(A(x,y)) ==> event(B(decrypt(x,y))).
(* is equivalent and rewritten to *)
query x,y,e; event(A(x,y)) ==>
  event(B(m)) && x = sk(e) && y = encrypt(pk(e),m,r)
.
```

Now, a destructor's semantics can be defined via multiple reduction rules, intuitively representing the different ways in which the destructor may be rewritten. For example, instead of failing, the `decrypt` destructor could be encoded to return the constant `error` when decryption is not possible:

```
fun decrypt(bitstring, bitstring): bitstring
  reduc forall e, m, r;
    decrypt(sk(e), encrypt(pk(e), m, r)) = m
  otherwise forall t, u;
    decrypt(t, u) = error
.
```

Hence, a term `decrypt(x, y)` may have multiple ways of instantiating its arguments to produce a message. When such a destructor occurs in conclusion of the query, this is translated as a disjunction over all these possible instantiations. Internally, these possibilities are generated by computing the variants [CLD05] of the terms containing the destructors, modulo the equational theory. This set is always finite as proverif considers only theories satisfying the *finite variant property*. Our previous query ψ would then be rewritten to

```
query x,y,e; event(A(x,y)) ==>
  (* the first case *)
  (event(B(m)) && x = sk(e) && y = encrypt(pk(e),m,r))
```

```
4
5     (* the second case *)
6     || (
7       (* the result of the second case *)
8       event(B(error)) &&
9       (* a negation of the first match criteria *)
10      (forall e',m',r'; (x,y) <> (sk(e'),encrypt(pk(e'),m',r')))
11    )
12  .
```

We note that this syntax extends the syntax exposed to the ProVerif user. Concretely, it contains a universal quantification `forall` in the conclusion. This was already supported by the internal representation of ProVerif clauses and queries, while it remains unexposed to the user.

Adding letfun The construct `letfun` can be seen as an extension to term generation that incorporates some components of process calculi, such as conditional tests, variable assignment, name generation, and events. In general, the evaluation of a `letfun` construct depends on where it is called in the executed process. For the conclusion of queries, we restrict the use of `letfun` to functions that are protocol independent, meaning those that are built solely from terms, conditionals, and variable assignments. Internally, this class of `letfun` can be expressed as a destructor term. For example, we can convert the `extract_vote'` from above into the following destructor:

```
1  fun extract_vote'(ballot, elec_id, voter_id) =
2    reduc forall b, e;
3      extract_vote'(b, elec, 3) = voteA
4    otherwise forall b, e, id;
5      let v = extract_vote(b, elec, id) in
6      extract_vote'(b, elec, id) = v
7  .
```

Hence, we convert the letfun into destructors on-the-fly, and then reuse the destructor logic described above.

4.7 Case Studies

We apply our unified framework for verifiability and privacy to several voting protocols. We consider all protocols previously proven using the verifiability framework, namely the ones listed in [CCD23] as well as two new ones: Vote&Check [CDGL24] and Short Voting Codes [CDM24]. We were able to prove both verifiability and privacy for all of them with two exceptions: Vote&Check and CHVote [HKLD17]. For CHVote, the verifiability model omits many important details for privacy (such as encrypting the votes). There is no existing privacy model, and there are actually unresolved privacy issues [CDG25], hence we did not attempt to model privacy. Vote&Check highlights a limitation of the verifiability framework that as-

sumes that verification is performed before or during the tally. We further comment on this case in Section 4.7.5.

An overview of the considered models and their respectively proven properties is given in Table 4.1. Interestingly, for some models, proving equivalence is faster than proving verifiability. This can be explained by the fact that the cellBB lemma allows ProVerif to easily reason where the ballots of Alice and Bob are, but also because it remains demanding for ProVerif to track the counters necessary for verifiability. Compared to the verifiability framework, our novel framework has mostly the same performance, but may be slower in some scenarios (e.g., when proving that the tally counter is decreasing), likely because of the additional counter introduced for `Count`.

Table 4.1: Proven properties per system.

	Verifiability		Privacy	
Running example Helios-like [Adi08]				
with honest decryption 1	✔	6s	✔	11s
with honest decryption 2	✔	6s	✔	17s
with honest decryption 3	✔	6s	✔	5s
Belenios [CGG19, CCD23]				
with honest server	✔	13s	✔	20s
with honest registrar	✔	13s	✔	13s
with honest server & registrar	✔	13s	✔	16s
Swiss Post Protocol [Swi22, CCD23]				
with honest CCR1 & CCM1	✔	1092s	✔	18s
with honest CCR1 & CCM2	✔	1089s	✔	11s
with honest CCR2 & CCM1	✔	1093s	✔	18s
with honest CCR2 & CCM2	✔	1091s	✔	11s
Short Voting Codes [CDM24]	✔	851s	✔	1228s
CHVote [HKLD17, CCD23]	✔		open issues in [CDG25]	
Vote & Check [CDGL24]	✔		not supported	

The models are provided as supplementary material for [CCDM26]. The test cases can be rerun using the util `driver.py` in the `tests` directory. The measurements have been done using ProVerif commit `de4036ba`, the newest commit on the branch necessary for the verifiability framework, on a Laptop with a 11th Gen Intel i7 at 4.70 GHz. The reported time includes everything necessary to fully reason about the respective property, hence for verifia-

bility notably includes the time to prove the lemmas, which are sometimes proven separately to the main verifiability queries.

Method Thanks to our backwards-compatibility with the verifiability framework, existing models need only superficial extensions:

- We need to add the channel `res_vote_ballot` for the ballots of the honest voters.
- For models that implement a verifiable shuffle, we need to implement the `Pre_HBallots` and the `Verify_Decrypt` process.
- To let the cellBB lemma help with termination, we further need to implement `extract_vote(ballot)`.

This is best demonstrated on the Short Voting Codes model. This model already fully included privacy-related aspects, and therefore adding these small extensions was sufficient to directly reach a proof in our new framework. The models that were more difficult to extend delivered either comparatively weak guarantees (e.g., Belenios allows to misattribute ballots, which made detection of the ballot of Alice and Bob by the framework difficult), or did not fully model privacy-related aspects in their verifiability model (e.g., the Swiss Post Voting System model did not model the voting device in their verifiability model, as it is untrusted for verifiability).

4.7.1 Running example

Our running example is based on [Adi08], but adds a zero-knowledge proof to prevent ballot copying [CS11]. Further, to showcase the framework's capabilities, we implement a tally phase that supports a decryption shuffle. We note that the same structure can also be reused to support a reencryption shuffle.

Due to the complexity of the tally in this example, compared to the other models, we had to give some additional guidance to ProVerif. This includes a lemma that asserts that the randomness in the ballot remain consistent in the left and right parts of the bitrace. Concretely, we place an event `GeneratedRandom` to record the randomness generation `new r` by the honest voter. The lemma then asserts that if some recorded `r` is used in the ballot in the left world, then the right world corresponds (and vice-versa).

```
1  mess(cell_BB(elec, i), (encrypt(pk(elec), v, diff[rL, rR]), p)) &&
2    event(GeneratedRandom(diff[rL, rR'])) ==> rR = rR'
3  .
4
5  mess(cell_BB(elec, i), (encrypt(pk(elec), v, diff[rL, rR]), p)) &&
6    event(GeneratedRandom(diff[rL', rR])) ==> rL = rL'
7  .
```

4.7.2 Belenios

Belenios [CGG19] is very close to our running example, but it adds a registrar as an additional authority. The registrar is in charge of delivering private credentials to the voters and publishing the corresponding public parts (and whom they belong to). To cast their ballot, the voter signs it using the registrar's private credential, and then sends it over an authenticated channel to the server. The protocol is secure if either the server or the registrar is honest, by notably preventing ballot stuffing by either authority.

However, the protocol allows a malicious registration authority to cast a ballot in another voter's name. Concretely, the attacker can intercept an honest voter's ballot, and recast it over the authenticated channel of a malicious voter, while claiming that the public part of the credential of the honest voter actually belongs to the malicious voter. Thereby, the ballot is validated successfully by the honest server, who attributes it wrongly to the malicious voter. While this does not break privacy (as, e.g., copying ballots is prevented), it is hard to prove in our framework, as we relay on detecting the honest voter's ballots to perform the swap.

To ensure in our model that the ballots of honest voters 1 and 2 are attributed correctly, we let our framework read the private credential used by honest voters 1 and 2 to attribute the ballots, and for all other ballots, allow any other attribution.

```
letfun get_ident_from_ballot(elec: elec_id, b: ballot) =
  let c = get_credential(b) in
  get credentials(=elec, id, =c)
    suchthat (id = 1 || id = 2) in ( id )
  else (
    get public_credentials(=elec, id, _) in ( id )
  )
```

We highlight that this function never blocks, which is important for soundness: `public_credentials` will always contain at least one entry, as the server uses this same table to check whether to add a given ballot to the ballot box. Concretely, the function delivers a superset of all real traces, by allowing to attribute a given ballot to any `id` instead of only the `id` where the corresponding public key matches. We choose this approach to avoid placing any assumption on the consistency of the attacker-controlled `credentials` and `public_credentials` tables. For verifiability, we keep the trivial implementation that uses the public credential attached to the ballot to query `public_credentials` for the corresponding `id`.

Unfortunately, ProVerif's precision is insufficient to prove the privacy model secure without additional help. Concretely, ProVerif overapproximates the else branch, by not including in the else branch the negated if condition. To resolve this, we use the same technique as presented in [CDGL24], which we will extend upon directly below in Section 4.7.2.1.

4.7.2.1 Proving soundness of the axiom

In order to prevent over-approximations when verifying protocols with `else` branches, we need to refine our ProVerif model. In the spirit of [CDGL24], we follow a hybrid approach based on a pen-and-paper proof and additional lemmas proved by ProVerif. In the following we sketch this approach, while we refer interested readers to [CDGL24] for the precise details.

Main idea To prevent ProVerif from over-approximating `else` branches, we define a generic axiom enforcing the `else` branch semantics in `get` commands. For sake of simplicity, we state it for `get` commands of the form: `get tab(x1,...,xn) suchthat xi = y in P else Q` where `y` is a bound variable. We introduce two events: `Inserted(x)`, which is placed each time an entry `x` is added in the table `tab`, and `NotFound(y)`, which is placed in the `else` branch with `y` being the bound variable occurring in the get condition. Then, we want to add the following axiom. Note that this axiom is too strong in general, and the axiom defined in the ProVerif models is actually weaker according to the changes explained just after.

```
1  axiom
2    event(Inserted(x)) && event(NotFound(y)) && x = y
3      ==> false
```

Because ProVerif does not check soundness of axioms, we do a pen-and-paper proof to prove it. Concretely, we manually prove that if some simple properties hold, then our main axiom holds as well. These simple properties are proven as lemmas with ProVerif.

Proof We need four events to define our axiom and our supporting lemmas. As already stated, the event `Inserted(x)` is placed each time an entry `x` is added in the table `tab`. Further, we define the event `CreatedFresh(x, r)`, which follows directly in the process after some action `new r` in a context `x`. Then, we extend the process with the `get` command as follows. The event `Received(b)` is added after some `in(c, b)` of the process. Lastly, the event `NotFound(y, b)` is extended to record an extra value `b` used when executing the `get` command leading to the `else` branch. The process therefore looks as follows.[4]

```
1  let R() =
2    in(c, b);
3    event Received(b);
4    ...
5    get tab(x1, ..., xn) suchthat xi = y in P
6    else (
7      event NotFound(y, b);
8      Q
```

[4]Note that the `get_ident_from_ballot` function follows the structure of this process `R`.

9)

Given the placement of these events, we now first give a preliminary proposition. Then, we define the precise axiom, and two lemmas to help us prove the axiom. Finally, we prove two main theorems.

The intuition behind the proposition is that when some `CreatedFresh(x, r)` appears in the trace, it does so directly after the corresponding `new r` action. Any other event containing `r` must have been raised later on. We state the proposition using timestamps `@t` attached to events. Further, we use the operator < to denotes ordering between the timestamps. Hence, for example, for some `Event1()@t1` executed before some `Event2(b)@t2`, it holds that `t1 < t2`. The proposition is stated as follows.

Proposition 1 *For any trace t with `CreatedFresh(x, r)@t1` and any other event `Event(b)@t2` for `subterm(r, b)`, it holds that `t1 < t2`.* □

Proof We proceed with a proof by contradiction. Assuming there is a trace t with `CreatedFresh(x, r)@t1` and some other event `Event(b)@t2` such that `subterm(r, b)`, for which it holds that `t1 > t2`. This implies that the prefix t' of this trace until `t2` includes the `Event(b)@t2`, but t' does not include `CreatedFresh(x, r)@t1`. As `Event(b)@t2` includes `r` because of `subterm(r, b)`, it must have been created after `new r`. However, then t' contradicts how we defined `CreatedFresh(x, r)`, which is specified to directly follow `new r` in the process (without `Event(b)@t2` "in-between"). Proposition 1 follows. ■

To begin our main proof, let us propose two lemmas, and state a weaker axiom sufficient for our proof:

- lemma 1: `Received(b)` $\wedge$ `CreatedFresh(x, r)` $\wedge$ `subterm(r, b)` $\implies$ `Inserted(x)`
- lemma 2: `NotFound(y, b)` $\implies$ `Received(b)`
- axiom: `Inserted(x)` $\wedge$ `CreatedFresh(x, r)` $\wedge$ `NotFound(y, b)` $\wedge$ `subterm(r, b)` $\wedge$ `x = y` $\implies$ `false`

Lemma 1 intuitively encodes that before the ballot `b` is received, the credential `x` used in that ballot has been created. The ballot `b` and the credential `x` are linked via the `CreatedFresh` event and the subterm function. Lemma 2 is purely syntactic, and follows from how we defined the events.

Theorem 1 *If lemma 1 and lemma 2 hold, then the main axiom holds.* □

Proof The proof that the two lemmas imply that the axiom holds is done by contradiction. Assuming that the axiom does not hold, but the two lemmas do, we prove that we can derive a contradiction. Concretely, given a

execution trace which falsifies the axiom, we can derive that it must contain the following events:

`Inserted(x)@t1` ∧	(by ¬ axiom)
`CreatedFresh(x, r)@t2` ∧	(by ¬ axiom)
`NotFound(x, b)@t3` ∧	(by ¬ axiom)
`subterm(r, b)` ∧	(by ¬ axiom)
`Received(b)@t4` ∧ `t4 < t3`	(by lemma 2)
`Inserted(x)@t5` ∧ `t5 < max(t2, t4)`	(by lemma 1)

We now show that such an execution trace cannot exist:

- We first conclude that `max(t2, t4) = t4`. Concretely, we observe that `b` contains the fresh term `r` by `subterm(r, b)`, and that there exists `CreatedFresh(x, r)@t2`. According to Proposition 1, it must be the case that `t2 < t4` for `CreatedFresh(x, r)@t2` and `Received(b)@t4` such that `subterm(r, b)`. Therefore, `max(t2, t4) = t4`.
- Combining this with lemma 2, which asserts that `t4 < t3`, we can conclude that `t5 < t4 < t3`. However, semantically, `NotFound(x, b)@t3` is not accessible if `Inserted(x)@t5` has been raised before. This leads to the contradiction.

Hence, either one of the two lemma does not hold, or the axiom holds. Or in other words, if the lemmas hold, then the axiom holds. ■

Proving lemmas Following the reasoning presented in [CDGL24], the two lemmas previously defined need to be proven in a model without the axiom. Unfortunately, when proving equivalence queries, this leads to non-termination issues.

In order to overcome this limitation, we decided to slightly restrict our model: we only consider traces in which the setup and the voting phases are disjoint, i.e., all the events `Inserted(x)` are executed before all the events `CreatedFresh(x, r)`. In our model, `Inserted(x)` corresponds to the registration of the voter public signing key, while `CreatedFresh(x, r)` is the generation of the randomness used to form the ballot. When considering an implementation of the Belenios protocol in practice, this restriction seems very realistic.

```
1  restriction
2    event(Inserted(x))@t1 && event(CreatedFresh(x, r))@t2
3      ==> t1 < t2
4  .
```

Theorem 2 *Assuming the above restriction and lemma 2 hold, then all traces which might falsify lemma 1 must satisfy the axiom.* □

Proof Now, given a execution trace which falsifies lemma 1, we can derive the following. First, if the `Inserted(x)` event is never raised, then the axiom is trivially true (as its precondition is never fulfilled). Second, when the `Inserted(x)` is raised, we get the following trace:

```
Received(b)@t1 ∧                          (by ¬ lemma 1)
CreatedFresh(x, r)@t2 ∧                   (by ¬ lemma 1)
subterm(r, b) ∧                           (by ¬ lemma 1)
Inserted(x)@t3 ∧ t3 > max(t1, t2)         (by ¬ lemma 1)
t3 < t2                                   (by restriction)
```

This execution trace cannot exist. We first observe that `t3 > max(t1, t2)` implies that `t3 > t1` and `t3 > t2`. Then, by the the restriction, which asserts that `t3 < t2`, we immediately get a contradiction. ■

So if we prove now, with ProVerif, that any trace that satisfies the axiom satisfies lemma 1, we can conclude that all traces indeed satisfy lemma 1, whether or not they satisfy the axiom. This allows to conclude that lemma 1 holds. Indeed, this proof succeeds with ProVerif.

With the lemma 1 fully proven, and lemma 2 syntactically true, we can then also conclude that the axiom holds.

4.7.3 Swiss Post Protocol

The Swiss Post Protocol [Swi22] is set in the demanding setting enforced by the Swiss chancellery [Swi13], and bases its security on return codes and a verifiable decryption shuffle in the tally phase. In a nutshell, an honest setup component prepares the election key material with four online components, of which one is honest. The setup component sends for each voting option a return code to the voter over the postal channel. The voter selects the candidates on their voting device, which sends the encryption thereof to the online components. These jointly derive the corresponding return code, notably without decrypting the ballot, and send it back to the voting device. Finally, if the return codes shown on the device match with the return codes received over the postal channel, the voter confirms their vote. After the voting phase, the confirmed ballots are verifiably shuffled and decrypted.

In this setting, there is an additional trust assumption for privacy: The voting device is assumed honest, as the voter directly selects their candidates on its screen. As it is not assumed honest for verifiability, it is not modelled in the existing verifiability model of [CCD23]. Therefore, we need to first implement the voting device for the privacy proof. Further, the verifiability model omits the lexicographic ordering of the list with the return codes, hence deriving the return code would trivially leak the voter's choice, by

their corresponding position in the list. ProVerif indeed immediately finds an attack. We therefore model the lexicographic ordering by making the ordering indistinguishable relative to `voteA` and `voteB` of the honest voters.

```
letfun sort(id: voter_id, return_codes: list3_t) =
 let (voteA, voteB, voteC) = return_codes in
 if (id = 1) then (
   (diff[voteA, voteB], diff[voteB, voteA], voteC)
 ) else if (id = 2) then (
   (diff[voteB, voteA], diff[voteA, voteB], voteC)
 ) else (voteA, voteB, voteC)
.
```

Further, we need to adapt the modelling of the tally slightly to be able to prove privacy. In the verifiability model, the adversary is given all decryption keys, and only the correctness of the decryption is checked. For privacy, we need to fully model the honest decrypting component, and cannot simply give its decryption key to the adversary, as then the adversary could trivially decrypt all ballots. It now however matters for the model which decryption authority is honest. When the first decryption authority is honest, we perform the decryption as before in `Decrypt`. When the second decryption authority is honest, we need to verify the proofs of the preceding (dishonest) decryption authority *before* the swap, hence in `Pre_HBallots`. This looks as follows.

```
let Pre_HBallots(elec: elec_id, id: voter_id, b: ballot) =
  let c = get_ciphertext(b) in
  in(ch_pub, (p, c'));
  if verify_zkp(pk(elec), p, (c, c')) then (
    out(res_pre_hballots(elec, id), wrap_m_ballot(c'))
  )
.

let Decrypt(elec: elec_id, id: voter_id, b: m_ballot) =
  let wrap_m_ballot(c') = b in
  let v = decrypt(sk(elec), c') in
  out(res_decrypt(elec, id), v)
.
```

If we would perform the zero-knowledge proof verification after the swap, they would fail for the honest ballots, as these have been swapped. For the same reason we highlight that the honest decryption authority does not create or publish a corresponding zero-knowledge proof in the model. Again, this zero-knowledge proof would make the swap (trivially) visible to the adversary, and therefore break the privacy proof. For our purposes, this is sound, as the zero-knowledge proof of the honest control component will always be correct and not reveal any other information.

4.7.4 Short Voting Codes

The Short Voting Codes [CDM24] protocol is also set in the Swiss setting. However, the voter enters voting codes, which the adversary cannot relate to the corresponding plain candidates. Consequentially, the voting device remains untrusted for privacy. When we take over the verifiability model [CDM24], this simplifies our work, as we do not need to implement the voting device. Further, as the existing privacy model of [CDM24] is close to the verifiability model, it is straightforward to integrate its existing capabilities (which prevent trivial attacks on the return codes table, as also necessary for the Swiss Post model). We are also able to directly take over the implementation of `Decrypt`, as the tally phase is not modelled (instead, the ballots on the bulletin board are simply decrypted). Overall, the model allowed to test our backwards-compatibility: besides not needing any non-superficial adaption, it also demonstrates that the execution time of our unified framework has not changed compared to the verifiability framework.

4.7.5 Vote&Check

Vote&Check [CDGL24] is a verifiable postal voting protocol. Its individual verification relies on trackers: At the end of the election, each vote is published in plain together with a unique tracking code. Each voter may then check that their personal tracker appears next to their intended choice.

In order to prove the verifiability of this protocol, the proposed model over-approximates some critical steps of the protocol such as the tally phase. As previously explained, the verifiability framework over-approximates the tally phase and in particular the mixnet used to protect vote privacy. Thanks to this simplification, modelling a voter verifying the presence of their ballot in the input of the mixnet is equivalent to modelling a voter verifying in the output of the mixnet. If this simplification is sound when verifying end-to-end verifiability, it is no longer applicable when verifying vote privacy; as explained above our privacy framework provides a more faithful model of the tally phase which inevitably breaks how voters verifications were modelled in Vote&Check. Concretely, because of the swap, the input of the mixnet is no longer equivalent to the output of the mixnet. Therefore, modelling a voter verifying the presence of their ballot changes depending on whether is done before or after the tally.

This limitation is generic and applies to any protocol whose voters' verifications apply to the output of a tally relying on a mixnet. Such a situation arises in any protocol with transparent verification such as SElect [KMST16], Selene [RRI16] and Hyperion [DRRR24]. In order to preserve backward compatibility of our new framework to the verifiability framework, we did not address this limitation and kept these protocols out-of-the-scope.

4.8 Discussion

We have provided a generic framework to prove both verifiability and privacy for voting protocols. We have successfully applied it to several protocols of the literature, including Belenios [CGG19], Swiss Post [Swi24c] and the short voting codes proposal from Chapter 3. To ease termination and precision, we added 20+ lemmas and axioms, notably a custom lemma that establishes a link between the ballots cast by the voters and those stored on the bulletin board. Further, to ease reuse of the framework, we have extended the expressiveness of the syntax of queries in ProVerif, which is of general interest. Concretely, ProVerif now supports the inclusion of destructor symbols (such as `decrypt`) in the conclusion of queries.

Simplify and clarify user-interface For backwards-compatibility towards the verifiability framework, changes to the user-interface were out of scope in this work. Nonetheless, with the continuation of the development of the framework, re-evaluating the user interface towards simplicity and improving clarity may be a good idea. While undoubtedly, the framework needs to provide the `Vote`, `Verify` and `Decrypt` functions, the other functions seem more dependent on the precise modelling approach chosen, of which there are plenty of valid options.

For example, as superficial changes, the user interface could aim for more consistency: this concerns the naming of types and function signatures, but also how the user sends messages to the framework (i.e., either by channel or by table, which is both done currently). The user interface would however also benefit from more structural changes, to aid towards that the user can use the framework as a full black-box. For example, one could discuss the removal of the `get_ident_from_ballot` function, and instead directly insert the `ballot` together with the `ident` in the ballot box. This makes it more clear to the user that for any `ballot` in the ballot box, the `ident` is needed (which is why `get_ident_from_ballot` was introduced in the first place, and which is its only use by the framework). Further, the functions should guide the user to not hide assumptions in the model: For example, the Belenios verifiability model placed the functionality of the registrar in `Gen_Voter_Ident`, and the verification by auditors is placed in `is_valid_ballot`. Arguably cleaner would be the placement in processes that explicitly reflect the role executing it, such as how the `Gen_Voter_Ident` functionality is now placed in its own `Registrar` processus.

Vote privacy for voters who verify To avoid trivial false attacks, we model privacy by assuming that the ballots of (honest and verifying) voter 1 and 2 are included in the tally. This therefore assumes that individual verifiability holds, i.e., that voters who verify have their ballot correctly included in the

tally. This is a necessary condition (as highlighted in [CLW20]), and we always proof it in our context: for all protocols where we proof privacy, we also have a corresponding proof of E2E-verifiability.

However, the privacy model would be cleaner if it would be independent from the verifiability proof. This would allow to prove protocols for which E2E-verifiability does not necessarily hold, and it avoids any potential misinterpretation of the privacy proof results. Instead, we could model that the honest voter 1 and 2 unlock the tally phase only after successful verification (e.g., with a private channel from the end of `Verify` to the start of `Tally`). Intuitively, this would however require ProVerif to prove individual verifiability inside of the privacy proof. In will likely be hard to find an appropriate trade-off between precision and termination for a model that needs to reason about correspondence and equivalence at the same time. As an alternative modelling approach, we could define a restriction to only consider traces in the privacy proof for which the two honest voters successfully verified. This would probably additionally require to characterize under which conditions equivalence properties with restrictions indeed model the desired property.

Individual verification after the tally The framework currently allows to model verification before the tally. The voter may query the ballots stored on the bulletin board `cell_BB`, or (for revote scenarios) the ballot confirmed to enter the tally on `cell_tally_last_vote`. In contrast, after the tally finished decrypting the ballots, the plain votes are simply published on a public channel. There is no bulletin board or equivalent structure modelled that stores all these published votes. Therefore, verification after the tally cannot be modelled in general. However, many protocols rely on this, notably those relying on tracking code verification such as Select [KMST16], Selene [RRI16], or Hyperion [DRRR24]).

To allow these protocols to be modelled, the framework can introduce another bulletin board at the end of the tally phase. Its cells would then be filled with the published votes, indexed again by the election id `elec` and the voter index `id`. While there might be some precision and termination surprises, overall this seems doable, notably in a backwards-compatible way. Where it gets more complicated is how to treat the verification of the honest voters 1 and 2 in the privacy model. Per the privacy definition, these voters vote `diff`[`voteA`, `voteB`] and `diff`[`voteB`, `voteA`], respectively. However, after the tally, the framework has performed the swap to prevent false attacks, leading to the votes being now `diff`[`voteA`, `voteA`] and `diff`[`voteB`, `voteB`], respectively. As it is the case now, we again would want to avoid the user of the framework to worry about these implementation details. We could therefore introduce a new `Verify` function for verification after the tally phase, and adapt its vote parameter `v` correspondingly.

More faithful modelling of verifiable shuffling Verifiable shuffling as modelled in this work is not yet very faithful. Concretely, none of our case studies model a shuffle proof. The shuffle proof allows to verify that the plaintexts of the list of input ciphertexts and the list of permuted output ciphertext are equal. Instead, our running example uses the magic function `recognize_partial_decrypt` instead. The function captures the properties of the shuffle proof by forcing a bijection between the input and output lists of encrypted ciphertext. Worse, the function also essentially enforces the identity function to be used as the permutation, which helps ProVerif reason about the correspondence of the shuffled ballots.

In order to more faithfully model the shuffle proof, we would need an equational theory that reflects the corresponding zero-knowledge proof. In a bounded setting, we could encode all possible permutations of the lists into the equational theory. We would then likely encounter new precision and termination challenges, as a real permutation (not just the identity function) of the list of ballots will make reasoning harder for ProVerif. However, in the unbounded setting the framework currently resides in, the lists and therefore the permutations would need to be of arbitrary size. This seems difficult to encode without using AC (associative and commutative) symbols. AC is unfortunately currently out of scope of ProVerif.

Chapter 5

Conclusion

In this thesis, we first present a systematization of knowledge of mechanisms used in practice. Most of the systems we found through our market analysis were effectively black-box systems, and of the few more transparent ones, again many did not use state-of-the-art mechanisms in all categories. However, we also found 8 systems we consider well-documented, a necessary precondition for any verifiable system, and which indeed implemented the mechanisms we would expect from a system that is used in practice.

Then, we propose short voting codes. These codes are short enough so that they can easily be entered by hand by the voter (typically only one or two digits), and therefore make code voting more practical in multi-election settings. We embed the mechanism in a protocol compliant to the demanding Swiss legislation [Swi13], which we surpass by supporting a fully untrusted voting device, thanks to the voting codes. The protocol remains nonetheless simple, notably relying only on standard primitives, especially when compared to other protocols in the same setting [HKD23, Swi24c, HKLD17].

Finally, we detail our work on a proof framework for internet voting protocols in ProVerif [Bla16]. Based on previous work that models verifiability only [CCD23], we extend towards privacy while keeping backwards compatibility. The framework models an extensible election setting, unbounded in the number of (concurrent) elections, voters and votes. Further, it provides a library of lemmas to balance termination and precision. Importantly, the framework allows to reuse the same protocol model for both types of proofs. To further increase usability of our framework, we implement an improvement to the expressiveness of the syntax of lemmas in ProVerif, with is of more general interest. Finally, we demonstrate the versatility and effectiveness of the framework by applying it to various case-studies. We include systems from both literature and industry, and also proof the short voting codes proposal secure using this approach.

At the end of each chapter, we have already sketched some future work directly motivated out of the presented work. Here, we will extend on some more fundamental potential future research directions.

5.1 Future research

5.1.1 Modular systems and proofs

The voting systems differ heavily in how they implement the voting phase: in terms of how the ballot of the voter is formed an authenticated, and how the voter verifies. These mechanisms are strongly guided by the election setting, and therefore a wide variety of mechanisms are used in practice (compare Chapter 2). For example, there is an established E-ID in Estonia, which is therefore also used for authentication in their internet voting system [Val24]. In Switzerland, no such E-ID exists, instead the authentication data is sent to the voter over postal mail [Swi24b]. The resulting systems are fundamentally different, and therefore need their own specification, implementation and proofs.

However, state-of-the-art voting systems have also strong similarities, in how they start and end the protocol. In the setup phase, trustees generate a distributed election key, while each trustee holds part of the private key. In the tally phase, the trustees shuffle (or homomorphically aggregate) the ballots, and then perform the distributed decryption using their part of the private key. This basic setup holds true for many systems (e.g. [CGG19, Swi24c, HKLD17, Tru23]), even if they aim for very different election settings.

It would be interesting to clearly characterize these commonly used mechanisms concerning their contribution to vote secrecy and verifiability. This would then allow to use them as a black box — during specification, proof and implementation — to focus efforts more on the specificities of the given election setting. This characterization would be already very useful on the primitive-level. For example it could capture the precise properties of verifiable shuffles, allowing to abstract away the concrete implementation chosen (e.g. [BG12, TW10]), such as their provided notion of zero-knowledge, correctness, and possibly other properties. However, the characterization would be especially useful on a system-level. A single blackbox could include the distributed key generation, verifiable shuffling and verifiable decryption, and therefore fully capture the election-independent part of an internet voting protocol.

5.1.2 Verifiability notions

In this work, we use the E2E-Verifiability notion as characterized by Cortier et al. [CGGI14]. However, not all protocols are provable using this definition, as it demands that ballots cast by honest voters are always received correctly by the bulletin board (i.e., the adversary can never change honest ballots). The protocols we have seen in the Swiss setting achieve this [Swi24c, HKLD17], including our own proposition presented in Chapter 3, as they al-

ways include a confirmation round that the voter only enters if the return codes imply that the ballot is correct. However, for some systems this definition is too strong, and the adversary can indeed change ballots of non-verifying voters (e.g. [CDGL24, Val22, Tru23]). Further, the definition may also be too abstract in other aspects, missing to capture all required security properties. For example in the Swiss setting [Swi13], abstaining voters need to be able to verify that indeed no ballot of theirs has been tallied, which is a verification not captured at all by the definition. Similar arguments can be made for other notions of E2E-verifiability [BBMP20, KTV10], which all need to choose a somewhat arbitrary level of abstraction, and then specific guarantees given the abstraction.

To more precisely state system behaviour, verifiability can alternatively be specified as multiple fine-grained properties. For example, one common decomposition divides into cast-as-intended, recorded-as-cast and tallied-as-recorded [BRR+15, BBH+17], and additional properties such as eligibility verification or ballot validity (e.g., [JMP13, BCG+18a]). Such decompositions can be useful as the fine-grained properties relate directly to concrete mechanisms of the protocol (easing the proof) or to required security properties (e.g. to define trust assumptions per property). However, it often remains unclear whether these fine-grained properties then indeed imply some form of E2E-Verifiability, or whether they leave open gaps (e.g. clash attacks [KTV12]). This notably includes work on (computational) proofs (e.g., [BCG+18a, Swi25c]) that omit to show which notion of E2E-verifiability are implied by the fine-grained properties. Further, some combination of fine-grained properties can also be observed to be unnecessarily strong. For example, both Baloglu [BBMP20] (strong adversary variant) and BeleniosVS [CFL19] specify that all ballots of honest voters in the bulletin board must be correct. They then combine this with cast-as-intended verification, where the voter again verifies that the ballot is correct. This is unnecessarily strong, and it would be sufficient for the voter to verify that the ballot *exists*. Using these compositions we would for example be unable to prove a code voting scheme where the voter only verifies participation (e.g. [RT09]).

A refreshed understanding of the E2E-verifiability notions, and the necessary and sufficient fine-grained properties to imply them, seems promising to capture more settings appropriately. New E2E-verifiability notions could precisely capture the Swiss setting (including the abstention verification) or the Estonian setting (for which [CGK+16, BBMP20] are too strong or too weak[1]). A better understanding of the fine-grained properties would

[1] [BBMP20] defines a weak and a strong variant. The strong variant is too strong for the same reason as [CGK+16], namely that it requires even the ballots of non-verifying honest voters to be correct. The weak variant permits an adversary to insert ballots for all non-verifying voters, which is too weak: Ballots in the Estonian system are signed, which prevents ballot insertion for abstaining (non-verifying) voters.

make proofs such as [BCG+18a, Swi25c] less effort and more convincing, as they could choose a set of *minimal* fine-grained properties that *provably* imply some notion of E2E-verifiability. Further, the fine-grained properties could increase our capabilities to reason about corruption scenarios, where only some but not all fine-grained properties are broken. The approach also seems promising to ease performing back-of-the-hand analysis of protocols, in particular useful during protocol design (see appendix Section B.1 for an example that finds the mechanisms of [Val22] to be too weak, and of [HKD23] to be unnecessarily strong).

5.1.3 E-Collecting

While this thesis investigates internet voting, signature collection is another form of direct democratic participation. It is known for example in the European Union [Eur23] or Switzerland [Fed24]. In Switzerland, the introduction of electronic signature collection suddenly gained strong traction in 2024 [Ann24b], with members of almost all parties of parliament notably submitting a motion to introduce a electronic collection immediately [Ann24a]. This followed a report by a newspaper which highlighted that in the current paper-based process, many invalid signatures are observed, with probably a large part of the abuse uncovered [Mar24].

There are strong parallels to the internet voting setting in terms of security properties such a system needs to reach. Similar to vote privacy, the signers of an initiative or a referendum need to remain private, as signing is a strong indication of voting preference. Further, similar to E2E-Verifiability, it needs to be verifiable that the signatures come from real voters (no signature stuffing), and that all signatures have been considered (no signature dropping). However, there are important conceptual differences:

- The participation in a collection already exposes the choice. Therefore, we need a stronger privacy notation than in the internet voting setting: instead of vote privacy, we need *participation* privacy. In turn, as there is only participation, there is no equivalence to altering a ballot, hence a somewhat weaker notion of E2E-Verifiability suffices. The last consequence of participation is that the usual solution in internet voting to overcome the authentication-privacy dilemma cannot be applied: Verifiable shuffles or homomorphic aggregations are not useful, as there is no encrypted vote that can be mixed with other votes.
- During the participation period, the number of received signatures is public, i.e., fairness is not a target.[2] This is a threat to privacy, as an attacker can coerce a voter to sign at a specific time, and observe whether the number of signatures changes.

[2]Fairness refers to the aim to publish the result only after the collection period.

- Compared to election periods, the collection periods are much longer (months instead of days). This necessitates that the list of eligible voters is continuously updated (naturalization, coming of age, relocation, death), but past participations need to remain valid (i.e., if signed while eligible, a signature needs to remains valid). This needs a reworked understanding of what eligibility concretely means, and explicit inclusion of eligibility mutations into the protocol.
- Multiple collections generally run at the same time, while each collection has different start- and end-dates. This notably renders a setup for the voter *per collection* impractical, i.e., a secret channel to the voter with collection-specific secrets is unrealistic.

Interestingly, there is no academic work we are aware of that covers the E-Collecting setting. Hence, we first need to define precisely the security properties we want to reach. As sketched above, this includes a privacy notion such as participation privacy, and a verifiability notion such as a weak form of E2E-Verifiability. Further, we need to derive a realistic setting for the context we aim for, and then propose mechanisms to implement the security properties. In Switzerland, the cantons administer the electoral roll, while the federal administration attests whether a signature collection has been successful [Fed24]. This could serve as a blueprint of the setting for E-Collecting to continue distributing trust between the cantons and the federal administration. In terms of mechanisms, we get new possibilities using cryptography. For example, validation of a signature can be detached from the authentication of the corresponding public key, providing a basis for anonymous validation of signatures (e.g., the cantons whitelist eligible public keys, and the federal government validates the received signatures against these whitelists). Further, by encrypting the signature when sending it to the server, and through the use of a verifiable shuffle that includes signatures of different active collections, we may even be able to achieve participation privacy without assuming the internet an anonymous channel. Overall, E-Collecting opens up a rich new domain, with its own set of mechanisms to reach new notions of privacy and verifiability.

Appendix A

Contributions

A.1 PHP Bugfixes

We have implemented a tool for individual verification in the GI elections (see Section 1.5.2). This implementation was done in PHP, and as part of these efforts, we have discovered and resolved issues in PHP itself.

Bug 11054 filed in 2023 diagnoses a problem with the parsing of public keys. Essentially, parsing is tried with each supported format one-after-the-other, until some parsing attempt is successfully, or an overall failure is reported to the user. However, when the overall parsing succeeded, PHP failed to cleanup the errors of the unsuccessful parsing attempts, leading to false errors remaining in an error buffer. We have proposed and implemented a fix in PR 11055, which was merged and is generally available starting from php 8.3 onwards (November 2023).

Bug 16870 filed in 2024 diagnoses a regression where reasonably big exponentiations were no longer executable due to a botched fix to prevent memory overflows in PR 16384. Concretely, this prevented execution of elliptic curve cryptography, and therefore could have rendered the individual verification inoperable.[1] The botched fix was reverted quickly, leading to no real-world impact on our side, but it made it obvious that there are no suitable unit tests in the PHP code-base to check for vital cryptography use-cases. Therefore, we implemented unit tests executing finite field crypto in PR 16896, and later on elliptic field crypto in PR 18363, with the latter based on an extensively tested elliptic curve library developed initially for this exact purpose at famoser/elliptic. All changes to PHP are now executed against our new tests, which prevents another similar regression to reoccur.

[1] Indeed, its corresponding unit tests were failing, which is why we became aware of the issue early. Even worse, the PHP minor release with the bug was released during an active election period, hence the tool was in active use. Luckily, it was never put in production by our server operators, after we have made them aware of the issue.

Appendix B

Future work

B.1 Example: Fine-grained verifiability notions

We illustrate here how clearly defined fine-grained definition, composed into E2E-verifiability notions, can increase our understanding of a setting and protocols. Concretely, we propose the following decomposition, categorized into Individual Verification (IV) performed by the voters, and Universal Verification (UV):

- *IV: Cast-as-Intended*: Mirroring the classical notion of cast-as-intended [BRR+15], we understand this as a voter Id that verifies that their i-th ballot with plaintext P has been received by the system. If this verification succeeds, the event $\mathsf{VerifiedVote}(\mathsf{Id}, i, \mathsf{P})$ is raised, and the event implies that any future state of the system $\mathbf{B}$ has some ballot that opens correspondingly, i.e., $\exists b \in \mathbf{B}$, $\mathsf{Open}(b) = (\mathsf{Id}, i, \mathsf{P})$.
- *IV: Participation-as-Intended*. This is a weaker notion of cast-as-intended. Concretely, for Participation-as-Intended, the voter Id raises the event $\mathsf{VerifiedParticipation}(\mathsf{Id}, i)$ if they are convinced that their i-th ballot has been received by the system, i.e., any future state of the system $\mathbf{B}$ has some ballot that opens correspondingly $\exists b \in \mathbf{B}$, $\mathsf{Open}(b) = (\mathsf{Id}, i, .)$.
- *UV: Authentication*. Corresponding to the eligibility verifiability notion [KRS10, SRKK10], this property attests that all ballots have either been cast by honest voters (raising $\mathsf{CastVote}(\mathsf{Id}, i, \mathsf{P})$ in the process) or by adversary-controlled voters $\mathbf{Id}_A$. The weak notion Authenticated-Voter attests that for any state of the system $\mathbf{B}$ the honest voter has indeed cast the ballot attributed to them, i.e., $\forall b \in \mathbf{B}$, $(\mathsf{Id}, i, \mathsf{P}) = \mathsf{Open}(b)$, $\exists\, \mathsf{CastVote}(\mathsf{Id}, i, .) \vee \mathsf{Id} \in \mathbf{Id}_A$. The strong notion Authenticated-Ballot additionally attests that the ballot has been cast correctly, i.e., $\forall b \in \mathbf{B}$, $(\mathsf{Id}, i, \mathsf{P}) = \mathsf{Open}(b)$, $\exists\, \mathsf{CastVote}(\mathsf{Id}, i, \mathsf{P}) \vee \mathsf{Id} \in \mathbf{Id}_A$.
- *UV: Tallied-as-Recorded*. Mirroring again the classical notion of tallied-as-recorded [BRR+15], it is understood that if the event $\mathsf{Tallied}(r)$ is raised, the r corresponds to the voting result according to some function Result. Concretely, the event implies that for any future state of the system $\mathbf{B}$ the ballots open as such, i.e., $r = \mathsf{Result}(\{\mathsf{Open}(b) \mid b \in \mathbf{B}\})$.

Now, given these fine-grained definitions, we draft the following notions of E2E-verifiability:

- *Strong E2E-Verifiability*. We name the composition of Participation-as-Intended, Authenticated-Ballot and Tallied-as-Recorded the Strong E2E-Verifiability notion. We claim equivalence to the definition by Cortier et al. [CGGI14, CGK+16], and Baluglu et al. [BBMP20] (strong adversary variant), implied notably in the Swiss setting [Swi13]. Observe that Participation-as-Intended is sufficient as IV, as if a ballot corresponding to some honest voter exists in the ballot box, it must already be correct (by Authenticated-Ballot).
- *Weak E2E-Verifiability*. Some proposals are unable to achieve Ballot-Authentication (e.g. [CDGL24]), but in turn, the voter verifies their ballot indeed contains the intended content. We name this Weak E2E-Verifiability, composed by Cast-as-Intended, Authenticated-Voter and Tallied-as-Recorded. We claim equivalence to the verifiability notion aimed for in the Estonian setting [HMVW16] and the proven E2E-Verifiability notion in Vote&Check [CDGL24].

To indeed proof these claims, we would need to appropriately formalize the composition, and then show equivalence to referenced definitions. Only then are the definitions useful for use in proofs or similar. However, assuming the claims are correct, the definitions are already useful for some back-of-the-hand analysis of protocols:

- *Estonia [Val22]*: In the Estonian setting, the voting device forms the ballot, but is not assumed trustworthy. As the device can therefore modify the ballot content, the system fulfils at most Authenticated-Voter. To fulfil Weak E2E-Verifiability, it therefore requires Cast-as-Intended. However, Cast-as-Intended is not fulfilled, as the voter only checks the ballot content and that the ballot was cast by themselves, but does not check the iteration of the vote. Therefore, E2E-Verifiability cannot be proven, and indeed, a corresponding reordering attack has already been described [Per22].
- *HKD Protocol [HKD23]*: In this proposal, the voter casts (long) voting codes, which contain a signature by a trusted component. As the device cannot modify ballots, this fulfils Authenticated-Ballot. To have Strong E2E-Verifiability, the combination with Participation-as-Intended is sufficient. However, the proposal foresees that the voter checks a unique return code per voting code, corresponding to the stronger Cast-as-Intended notion. This is unnecessarily strong, and if the protocol would instead implement Participation-as-Intended, the voter task could likely be simplified (e.g. by the voter only checking a single participation code, equal for all voting codes).

Appendix C

Introduction (French)

Box 1: Machine translation of the introduction to French

This is a machine translation of Chapter 1 to French. The translation has been done using ChatGPT in January 2026 in a single conversation with the initial prompt *Translate the following english text to french. It is an academic publication, and therefore make sure the translation is high-quality. Keep the latex syntax.* We have proof-read and improved the automatic translation, but in case of discrepancies, the English version counts.

Le vote par Internet vise à organiser des élections via Internet. Pour certaines parties de l'électorat, le vote par Internet présente un avantage évident et accroît la participation, par exemple pour les Australiens vivant dans l'Outback, où le bureau de vote le plus proche est éloigné [NSW23], pour les expatriés pour lesquels système postal ne permet pas un acheminement dans les temps du matériel de vote et des bulletins [OEAMiEOfDIR08, DMSTW12, Ger21], ou pour les personnes en situation de handicap qui ne peuvent autrement pas voter de manière indépendante [Bun21]. Bien que, pour les électeurs disposant déjà d'un canal de vote pratique, aucune amélioration de la participation ne soit observée [GS17, SW24], le vote par Internet peut également revêtir une importance stratégique à long terme (e. g.,, comme c'est le cas pour l'administration suisse [Bun13, fK20]), en lien avec une volonté générale de fournir des services numériques aux citoyens.

Au regard des avantages intrinsèques et perçus du vote par Internet, l'état actuel de son utilisation en pratique semble limité. Hormis l'Estonie [Est25], aucun pays ne propose à ce jour le vote par Internet à une part significative de sa population. En Estonie, le système est continuellement mis à jour depuis son introduction [HW14, HMVW16, HKW20], mais il a été et continue d'être perçu de manière critique dans le milieu académique [SFD+14, Mül22, SHR23]. Le système de vote par Internet de la Poste suisse est actuellement le plus avancé et le plus transparent, cette dernière permet à des chercheurs d'étudier sa sécurité et ainsi d'améliorer le système [Swi24a]. Toutefois, ce système a également connu son lot de controverses [LHK19, HLPT20, PT19, Bun19b], lesquelles ont conduit à d'importantes refontes du système et de sa base législative [Bun19a, Pos19]. Il continuera à faire

l'objet de transformations majeures [Ste23, CDE+25], tandis que son utilisation en Suisse est actuellement limitée principalement aux expatriés [Swi13, Fed25a]. Quelques autres pays ont introduit le vote par Internet, mais en limitent également l'usage aux expatriés (e.g., en France [DH23]), ou ont abandonné le système après seulement quelques élections (e.g., la Norvège [BGS+16] et l'Australie [Com23]). Tous ces exemples ont été accompagnés de controverses concernant la sécurité des systèmes employés [DH23, ØK12, VT19]. Il apparaît clairement que la mise en œuvre d'un système de vote par Internet n'est pas une tâche triviale.

Si les objectifs du vote par Internet — protéger le secret du vote et l'exactitude du résultat électoral — restent identiques à ceux des élections traditionnelles, les hypothèses de confiance changent fondamentalement. Comparé au vote par correspondance, dont le traitement est complexe mais fortement distribué [KS19], les élections par Internet sont mises en œuvre sous la forme de systèmes largement centralisés (e.g., [Swi24b, Val22]). La compromission de ces serveurs centraux peut avoir un impact bien plus important que la compromission de parties d'élections classiques très distribuées, car les attaques peuvent passer à l'échelle beaucoup plus efficacement et sont plus difficiles à détecter (e.g., en l'absence de trace papier pouvant être falsifiée et servant de preuve de manipulation [SRF21]).

Pour le vote par Internet, cela signifie que le simple durcissement conventionnel des systèmes (e.g., le respect des principes de programmation sécurisée, des contrôles d'accès stricts, ...) est, dans de nombreux contextes, considéré comme insuffisant [oE17, Swi13, Est14], y compris pour des élections non politiques à plus faibles enjeux [Fre25, Fed25b]. Les systèmes doivent au contraire rendre leurs calculs *vérifiables* : l'électeur, d'autres composants du système, voire des auditeurs tiers, doivent être en mesure de vérifier que les bulletins sont traités conformément aux intentions. Cela doit bien entendu être réalisé sans compromettre le secret du vote. Nous approfondissons ces propriétés — la vérifiabilité et le secret du vote — dans la section suivante.

Box 4: Le vote par Internet n'est pas la banque en ligne

La numérisation des élections est fondamentalement différente de celle d'autres services, tels que la banque en ligne. Par exemple, un traitement incorrect en banque en ligne — qu'il résulte d'une fraude ou d'une simple erreur de programmation — est facilement détectable : le titulaire du compte dispose d'une liste détaillée de chaque entrée et sortie figurant sur son relevé bancaire, et peut identifier les transactions qu'il n'a pas initiées. Il peut alors se plaindre auprès de sa banque, qui est en mesure de lancer une enquête pour déterminer qui a initié ces transactions, quel montant est concerné et où il a été transféré.

Il en va tout autrement pour le vote électronique. Premièrement, les utilisateurs ne remarqueront jamais un dysfonctionnement du système, car cela nécessiterait de déduire, uniquement à partir du résultat électoral agrégé officiellement publié, si leur vote individuel a été modifié ou supprimé. En outre, même si un électeur dépose une plainte, celle-ci ne peut pas (et ne doit pas) recevoir de réponse : par construction, le système doit préserver le secret du vote, et il doit donc être impossible de retracer la manière dont le vote d'un individu a été traité. Comparées à d'autres systèmes sécurisés, les élections nécessitent donc des techniques de sécurisation différentes.

C.1 Propriétés

Afin d'attester qu'un système de vote fonctionne correctement, ses fonctionnalités sont conçues pour être vérifiables. Certaines vérifications doivent être effectuées directement par les électeurs ; on parle alors de vérification individuelle [SK95]. L'électeur doit pouvoir s'assurer que son bulletin a été créé correctement (*cast-as-intended*) et qu'il fait bien partie de l'urne (*recorded-as-cast*) [BRR+15, BBH+17, BCG+18a]. Dans certains contextes, un électeur qui s'abstient peut également vérifier qu'aucun bulletin en son nom n'a été inclus [Swi13, BCG+18a]. D'autres vérifications peuvent être réalisées par des composants tiers du système ou par des auditeurs, et sont désignées sous le terme de vérification universelle [SK95]. Cela inclut la propriété dite *tallied-as-recorded* [BRR+15, BCG+18a], qui garantit que tous les bulletins enregistrés sont correctement dépouillés. En outre, le système doit également garantir que seuls les électeurs éligibles participent, ce qui est appelé la vérification de l'éligibilité [KRS10, SRKK10].

La composition de ces différentes propriétés peut toutefois encore laisser subsister des failles. Par exemple, des attaques par collision (*clash attacks*) peuvent se produire [KTV12], lorsqu'un serveur de vote malhonnête fournit un reçu correspondant au même bulletin à deux électeurs distincts. C'est dans ce contexte que la notion de vérifiabilité de bout en bout (appelée E2E-verifiability) a été introduite, afin de définir la vérifiabilité à l'échelle de l'ensemble du système [KTV10, CGGI14, CGK+16, BBMP20]. La formalisation que nous utilisons dans ce travail définit que le résultat final est composé de l'ensemble des votes des électeurs honnêtes qui vérifient, de certains votes d'électeurs honnêtes qui ne vérifient pas, ainsi que d'au plus un vote par électeur malhonnête [CGGI14]. Cette notion relie ainsi directement les actions des électeurs au résultat de l'élection. Sa formulation considère un adversaire potentiellement actif, c'est à dire pouvant controller les canaux de communication ou même corrompre certains composants du système (e.g., votants, ou serveur de vote). Les électeurs honnêtes qui vérifient ont la garantie que leur vote est dépouillé. Les électeurs honnêtes qui omettent la

vérification peuvent voir leur vote supprimé par l'adversaire (par exemple, si celui-ci empêche la transmission via Internet), mais leur vote ne peut pas être modifié. En outre, l'adversaire ne peut pas insérer davantage de votes que le nombre d'électeurs malhonnêtes qu'il contrôle.f

Certains systèmes visent des propriétés de correction allant au-delà de la simple vérifiabilité du résultat de l'élection. La responsabilité (*accountability*) [KTV10] garantit que, lorsqu'une déviation par rapport à la spécification est détectée, celle-ci peut être attribuée à un composant défaillant. De plus, la résolution des litiges [KWC+13] vise à garantir que les litiges puissent être correctement résolus (i.e., en blâmant toujours quelqu'un mais jamais à tort), tandis que l'absence de litige (*dispute freeness*) cherche à empêcher l'apparition même de litiges [KY02].

Malgré ces fortes garanties de correction et vérifiabilité, le système doit continuer de préserver le secret du vote. L'adversaire doit être dans l'incapacité d'apprendre comment les électeurs ont voté, au-delà de ce que révèle le résultat de l'élection [Ben87, DKR09]. Cela peut être formalisé comme une propriété d'indistinguabilité, selon laquelle l'attaquant est incapable de distinguer un monde où Alice vote 1 et Bob vote 0 d'un autre monde où Bob vote 1 et Alice vote 0 [DKR09].

En plus du secret du vote, certains systèmes visent des garanties de confidentialité plus fortes. L'absence de reçu (*receipt freeness*) formalise l'idée que le système de vote ne doit pas fournir de preuve de la manière dont le vote a été exprimé [BT94]. Cela vise à empêcher la vente de votes par l'électeur à un adversaire. Renforçant encore cette notion, certains systèmes de vote sont également conçus pour résister à la coercition [MN06] : même si un électeur est contraint pendant certaines phases du vote (par exemple, si un attaquant l'espionne ou lui impose un comportement spécifique), il doit être en mesure d'exprimer son vote conformément à sa préférence réelle. Enfin, la confidentialité perpétuelle (*everlasting privacy*) vise à garantir le secret des votes indéfiniment [MN06, ACKR13], y compris face à des attaquants futurs ayant d'avantages de capacité de calculs. Par exemple, un attaquant pourrait stocker aujourd'hui des bulletins chiffrés et les déchiffrer ultérieurement lorsque des ordinateurs quantiques seront capables de casser la sécurité des mécanismes de chiffrement traditionnels [Sho94].

Box 5: La tension entre vérifiabilité et confidentialité

La confidentialité et la vérifiabilité semblent s'opposer l'une à l'autre. Du point de vue de la confidentialité, il serait idéal de ne publier aucune information autre que le résultat de l'élection. Cependant, afin de vérifier que ce résultat est effectivement correct, la publication de données supplémentaires est nécessaire. Ces données additionnelles

doivent être d'une manière ou d'une autre liées au contenu de l'élection, ce qui met intuitivement en péril la confidentialité.

Cette intuition a été formalisée, conduisant à des résultats d'incompatibilité. Par exemple, il a été montré que la vérifiabilité et le secret du vote inconditionnel (c'est-à-dire la confidentialité face à un adversaire non borné) sont incompatibles [CMFP+10]. Inversement, il a été démontré que le secret du vote nécessite la vérifiabilité individuelle, à savoir que les votes des électeurs honnêtes soient effectivement dépouillés [CL18]. Dans le cas contraire, un adversaire pourrait simplement supprimer tous les bulletins sauf un, et apprendre trivialement à partir du résultat de l'élection le vote qu'il contient.

C.2 Mécanismes

Atteindre simultanément les propriétés quelque peu contradictoires que sont la vérifiabilité et la confidentialité est un défi. Dans les élections à l'urne, cela est rendu possible grâce à des contraintes physiques et organisationnelles. Seuls les électeurs authentifiés sont autorisés à entrer dans l'espace de vote et à déposer un unique bulletin papier dans l'urne. À l'issue de la période de vote, l'urne est mélangée, puis les bulletins sont comptés. L'ensemble de ce processus peut être vérifié et, en cas de doute, le dépouillement peut même être répété. La confidentialité des électeurs est néanmoins préservée : ils remplissent leur bulletin en privé dans un isoloir et le déposent dans l'urne, où il sera mélangé avec tous les autres bulletins avant le dépouillement.

Dans le vote à distance (par correspondance), ces garanties sont affaiblies par rapport au vote à l'urne : l'électeur peut être observé, voire contraint, lors de la préparation de son bulletin (e.g., une contrainte familiale, un représentant de parti ou d'autres personnes), et il n'a aucun moyen de savoir si son bulletin parvient effectivement dans l'urne du bureau de vote. Au niveau du bureau de vote, certaines garanties élémentaires de confidentialité et de vérifiabilité sont mises en place — en vérifiant les informations d'authentification par rapport au registre électoral, puis en séparant les bulletins de leurs informations d'authentification avant leur ouverture — mais le processus est complexe et repose sur le comportement honnête de nombreuses personnes [KS19]. De fait, des cas de fraude lors du vote par correspondance ont déjà été documentés. Par exemple, un employé municipal malhonnête a ajouté des bulletins papier supplémentaires, qui n'ont pu être identifiés que parce qu'ils ne présentaient pas les plis habituels [SRF21]. Dans les élections par Internet, tous les mécanismes de sécurité fondés sur le papier — et plus généralement sur des processus physiques — sont toutefois inapplicables ; d'autres moyens de sécurisation des élections doivent

donc être déployés.

Il existe une multitude de mécanismes cryptographiques proposés pour réduire (grace au numérique) la tension entre les garanties de confidentialité et de vérifiabilité [BBH+17, FDL07, JMP13]. Ces mécanismes sont ensuite composés en systèmes sécurisés de bout en bout (e.g., [Ben06, CGG19, Swi24c, Val22]), dans le but de choisir un équilibre entre vérifiabilité et confidentialité, tout en reposant sur des hypothèses de confiance réalistes. Un système de vote par Internet simple (e.g., [Ben06]), offrant des garanties élémentaires de confidentialité et de vérifiabilité, chiffre le bulletin à l'aide d'une clé électorale gérée par des assesseurs ou membres du bureau de vote, puis envoie le bulletin au serveur via un canal authentifié. Ensuite, une fois la phase de vote terminée, le serveur mélange de manière vérifiable[1] tous les bulletins reçus, afin de les dissocier de leurs électeurs. Enfin, les membres du bureau de vote déchiffrent les bulletins mélangés.

Les systèmes à l'état de l'art utilisent des mécanismes visant à éviter de faire confiance à un composant unique, tel que le serveur de vote ou un unique membre du bureau de vote. Grâce à la génération distribuée de clés [CGGI13] (e.g., telle qu'implémentée dans Belenios [CGG19]), la clé d l'élection peut être générée de sorte que le déchiffrement des bulletins chiffrés sous cette clé nécessite un quorum de détenteurs de parts de la clé de déchiffrement. L'authentification des bulletins peut être rendue vérifiable par des tiers au moyen de signatures numériques (e.g., comme en Estonie [Val22]), éventuellement combinées à une authentification multifactorielle impliquant plusieurs autorités (e.g., comme dans Electa [Ass23] ou Belenios [CGG19]). Afin de protéger les bulletins jusqu'à la phase finale de dépouillement et de garantir qu'ils ne sont ni supprimés ni modifiés, ceux-ci sont stockés simultanément par plusieurs composants (e.g., dans le système de vote de la Poste suisse [Swi24c] et en Estonie [Val22]) ou publiés sur des tableaux d'affichage numériques (publics) (e.g., [HL08, KKL+18, HSB21]).

Lors de la phase de dépouillement, la confiance peut à nouveau être distribuée. De nombreuses méthodes de vérification des mélanges ont été proposées [SK95, JJR02, TW10, BG12], le choix le plus établi reposant sur des preuves à divulgation nulle de connaissance de mélange (e.g., la construction de Bayer–Groth [BG12] utilisée dans le protocole de la Poste suisse [Swi24c], ou la proposition de Terelius–Wikström [TW10] implémentée dans Verificatum [Dou] et CHVote [HKLD17]). Au lieu d'utiliser des mélanges

[1] L'idée du mélange (*shuffling*) consiste, étant donnée une liste de bulletins, à modifier la représentation de chaque bulletin individuel et à randomiser leur ordre dans la liste. Ainsi, chaque bulletin de la liste de sortie ne peut plus être relié au bulletin correspondant de la liste d'entrée. Ce mélange est rendu vérifiable en fournissant aux auditeurs une preuve convaincante que la liste de sortie représente les mêmes votes que la liste d'entrée (c'est-à-dire qu'aucun bulletin n'a été supprimé, ajouté ou modifié). Bien entendu, cette preuve ne doit pas non plus révéler le lien entre les bulletins d'entrée et de sortie.

vérifiables, une autre approche consiste à agréger homomorphiquement les chiffrés [CF85, Ben87] (e.g., comme dans ElectionGuard [BNPW24]), où le chiffré résultant est la combinaison (multiplicative ou additive) de tous les chiffrés d'entrée, c'est-à-dire le résultat de l'élection. Cette approche évite l'hypothèse d'un unique composant de confiance pour assurer la confidentialité des mélanges vérifiables[2]. En revanche, la mise en œuvre de règles de vote arbitraires au moyen de l'agrégation homomorphique (y compris les preuves nécessaires de validité des bulletins) est délicate [CGY22, HKLR24]. Enfin, soit le chiffré unique issu de l'agrégation homomorphique, soit les chiffrés mélangés et donc anonymisés provenant d'un mélange vérifiable, sont déchiffrés par les membres du bureau de vote (décrits, par exemple, dans Belenios [CGG19]).

Il existe encore de nombreux autres mécanismes permettant d'étendre davantage les garanties de confidentialité et de vérifiabilité d'un système. Afin d'éviter la production de reçus permettant à l'électeur de prouver à un adversaire comment il a voté, des signatures re-randomisables [CCFG16, CFL19] ont été proposées. Celles-ci restent valides sur une représentation aléatoirement modifiée du bulletin, rendant ce bulletin re-randomisé publiable tout en préservant de fortes propriétés d'authentification. Pour améliorer la résistance à la coercition, des identifiants factices permettent aux électeurs de déposer un bulletin qui ne sera pas comptabilisé [JCJ05, CCM08]. En outre, certains systèmes autorisent le revote (e.g., [Val22]), permettant aux électeurs de remplacer leur bulletin à tout moment. Afin de combiner le revote avec un tableau d'affichage numériques public, la mise à jour dénégable du vote [LQT20, MPP24, KTV15] a été proposée, dans laquelle l'adversaire ne peut pas savoir si un vote donné a été mis à jour. Pour atteindre une notion pratique de confidentialité perpétuelle, l'utilisation d'engagements à dissimulation inconditionnelle (*unconditionally hiding commitments*) est proposée (e.g., Pedersen [Ped92]); ces derniers devant être intégrés dans un protocole de vote par Internet approprié (e.g., [CPP13]). Ces engagements continueront de masquer le choix de l'électeur même si la cryptographie venait à être compromise à l'avenir.

La réduction de la confiance dans le système inclut également, dans le contexte du vote à distance, l'appareil (personnel) de vote utilisé par l'électeur. En particulier dans les élections politiques, ces appareils ne sont souvent pas supposés pleinement fiables (e.g., en Estonie [Est14] et en Suisse [Swi13]). Diverses méthodes ont donc été proposées pour permettre à l'électeur de vérifier que son vote a été correctement exprimé par son propre dispositif. Dans l'approche *audit-or-cast* (également appelée *Benaloh challenge* [Ben06]),

[2]Bien que les mécanismes de vérification garantissent que les bulletins d'entrée correspondent aux bulletins de sortie, un composant malhonnête pourrait néanmoins choisir un ordre prévisible des bulletins (e.g., la fonction identité), ce qui empêcherait l'établissement de garanties de confidentialité.

l'appareil de vote de l'électeur s'engage sur un bulletin, puis l'électeur choisit soit d'auditer, soit de déposer ce bulletin. Cette approche est notamment implémentée dans Helios [Adi08], bien que cette méthode fondée sur des défis semble difficile à utiliser pour les électeurs [MKRV18]. Dans l'approche *cast-then-audit*, les électeurs ont la possibilité d'auditer le bulletin déposé. Cette méthode est utilisée dans le contexte estonien [HW14] ou, en améliorant l'absence de reçu grâce à l'utilisation de preuves interactives à divulgation nulle de connaissance, dans POLYAS [Tru23, MT23].[3] Dans les contextes où aucun second dispositif ne peut être supposé, mais où un canal sécurisé du système vers l'électeur est disponible, des codes de retour sont proposés [AC11]. Une feuille de vote comportant un code de retour unique par candidat est envoyée à l'électeur avant l'élection. Ensuite, après que l'électeur a exprimé son vote, il vérifie que les codes de retour affichés par l'appareil de vote correspondent aux codes reçus via le canal sécurisé. Cette approche est notamment retenue en Suisse [Swi24c, HKLD17], ainsi que dans d'autres contextes [AC12, Com19a]. Alternativement, lorsque la vérification lors de la phase de dépouillement est acceptable, des codes de suivi peuvent être utilisés [KMST16, RRI16, DRRR24], où tous les votes comptabilisés sont publiés en clair avec un code de suivi. L'électeur vérifie alors que son vote, identifiable grâce à son code de suivi personnel, est correct.

Si ces méthodes de vérification garantissent que l'appareil de vote (et d'autres parties du système) n'a pas modifié le candidat sélectionné, elles ne protègent pas la confidentialité face à un dispositif de vote malveillant. À cette fin, des codes de vote [Cha01] ont été proposés. L'électeur ne saisit plus son choix en clair, mais un code représentant ce choix, dont l'attribution est inconnue du dispositif de vote. Certaines approches utilisent des codes de vote pouvant être saisis manuellement [Cha01, CZZ$^+$16], mais comme ceux-ci peuvent être facilement devinés par un dispositif malveillant, ces approches doivent être combinées avec un mécanisme de *cast-as-intended*. D'autres propositions utilisent des codes plus longs, impossibles à deviner par l'appareil [RT09, HKD23, CFL19], au prix du compromis que l'électeur doit scanner un code QR (ou un mécanisme similaire) pour sélectionner le bulletin correspondant.

Compte tenu de l'ensemble de ces approches et systèmes, il convient de souligner qu'il n'existe pas de système unique optimal. Les propriétés précises à viser (e.g., la notion de confidentialité) ainsi que les hypothèses de confiance jugées acceptables dépendent fortement du contexte électoral. Cela conduit à des conceptions de systèmes très différentes. Par exemple, les systèmes estonien [Val22] et de la Poste suisse [Swi24c] sont tous

[3] L'audit dans POLYAS est effectué sur une ré-chiffrement du bulletin déposé. Cet audit est convaincant pour l'électeur, car une preuve interactive à divulgation nulle de connaissance est réalisée avec le serveur attestant que la ré-chiffrement est correct, mais cette preuve n'est pas convaincante pour un tiers.

deux conçus pour des élections politiques nationales. Toutefois, les hypothèses concernant les électeurs diffèrent sensiblement : en Estonie [Val22], les électeurs sont supposés être en possession exclusive d'une carte d'identité numérique personnelle et avoir accès à un second dispositif fiable. Le système s'attend donc à ce que les électeurs signent leur bulletin à l'aide de leur identité numérique et vérifient leur bulletin via l'approche *cast-then-audit* sur leur second dispositif. En revanche, dans le contexte suisse, il n'existe pas encore d'identité numérique, mais un composant d'initialisation de confiance et un canal privé (via le courrier postal) sont considérés comme acceptables. L'électeur reçoit ainsi par courrier une feuille de codes comportant un code d'authentification pour déposer son bulletin et des codes de retour pour vérifier le bulletin [Swi24c].

Box 6: Il est trivial de satisfaire uniquement la confidentialité ou uniquement la vérifiabilité

Compte tenu de l'ensemble de ces mécanismes et de leur complexité intrinsèque — qui doivent en outre être combinés de manière cohérente pour former un système complet — on peut se demander s'il n'existe pas de solutions plus simples. En effet, il est trivial de construire des systèmes qui ne fournissent que la vérifiabilité ou que la confidentialité. Par exemple, un système peut publier en clair qui a voté pour quels candidats, ou inversement ne rien publier du tout à l'exception du résultat de l'élection.

Cependant, atteindre des notions acceptables à la fois de confidentialité et de vérifiabilité demeure un défi et requiert l'utilisation combinée de plusieurs des mécanismes décrits. Cette complexité se reflète également dans les systèmes réels, qui nécessitent des centaines de pages pour documenter entièrement leur approche de manière détaillée ; voir par exemple la spécification du système de vote de la Poste suisse [Swi24c].

C.3 Le contexte suisse

La Chancellerie fédérale suisse [Swi13] impose des exigences très élevées en matière de sécurité pour le vote par Internet. Dans ce contexte, des feuilles de vote sont distribuées aux électeurs par courrier postal ; ces feuilles sont générées par un composant d'initialisation de confiance, en collaboration avec des composants de contrôle. Lorsqu'un électeur souhaite voter, il saisit son choix de candidat sur l'appareil de vote, qui chiffre le vote et envoie le chiffré aux composants de contrôle. Ceux-ci jouent intuitivement le rôle d'un serveur de vote distribué et calculent conjointement les codes de retour correspondants au vote exprimé. Ces codes sont renvoyés à l'électeur, qui utilise la feuille de vote pour vérifier leur validité, puis ce dernier confirme

son vote. Selon la définition de la Chancellerie fédérale suisse, la vérifiabilité doit être garantie même si l'attaquant contrôle le réseau, tous les composants de contrôle sauf un, ainsi que l'appareil de vote. En ce qui concerne la confidentialité du vote, les hypothèses de confiance sont similaires, à l'exception du fait que l'appareil de vote est alors également supposé de confiance.

Au-delà de la défense contre un adversaire aussi puissant, le contexte suisse se distingue par ses exigences élevées en matière de transparence et par une collaboration étroite avec la recherche. La spécification du système doit être entièrement publique, tout comme son code source. L'examen public est encouragé au moyen de programmes de récompense pour la découverte de failles (*bug bounty*). En outre, la Chancellerie fédérale suisse mandate régulièrement des experts internationaux pour examiner le système, et ces rapports sont eux aussi rendus publics [Swi24a]. Dans le cadre de ces efforts, divers problèmes ont été mis au jour et corrigés [HLPT20, CDG22, HR25], et une planification à long terme vise à améliorer le système au cours des prochaines années [Ste23].

Dans ce cadre, deux systèmes entièrement spécifiés sont proposés. Le Swiss Post Voting System [Swi24c, Swi24b] est actuellement le seul système pleinement développé et conforme à ce cadre ambitieux. Le système davantage académique, CHVote, fournit une spécification et une implémentation des primitives cryptographiques (mais, par exemple, pas d'interface utilisateur) [HKLD17]. La principale différence entre les systèmes réside dans la manière dont les codes de retour sont générés tout en préservant la confidentialité. Intuitivement, le système de vote de la Poste suisse calcule un haché du vote chiffré (de manière déterministe), puis utilise une table de hachage pour retrouver le code de retour correspondant [Swi24c]. Le système CHVote utilise des transferts aveugles (*oblivious transfer*) k-parmi-n afin de récupérer les k codes de retour correspondant aux k candidats sélectionnés parmi les n candidats au total [HKLD17]. Les deux protocoles présentent un niveau de complexité élevé comparable et reposent sur des primitives cryptographiques spécifiques. Bien qu'il soit difficile de quantifier précisément cette complexité, les deux spécifications dépassent chacune les 300 pages pour décrire en détail la cryptographie employée et les messages échangés entre les différents composants [HKLD17, Swi24c, Swi24d, Swi25a]. Cette complexité rend difficile une analyse intuitive du système et des différents états dans lesquels il peut se trouver (et éventuellement ne devrait pas se trouver) ; par conséquent, des méthodes plus avancées sont nécessaires pour analyser ce type de systèmes.

C.4 Méthodes formelles

Les méthodes formelles visent à analyser précisément les garanties de sécurité concrètes des systèmes. Étant données une spécification, des propriétés de sécurité visées et un modèle de menace, les méthodes formelles cherchent à déterminer si la spécification satisfait effectivement ces propriétés. Même pour des protocoles « simples », une analyse précise peut révéler un degré de nuance surprenant (e.g., des protocoles d'établissement de clés partagées peuvent satisfaire de nombreuses variantes de propriétés de sécurité subtilement distinctes [GHS+20]). Pour des protocoles complexes, tels que ceux requis par les systèmes de vote par Internet, ces méthodes permettent un raisonnement rigoureux indispensable pour comprendre exactement les propriétés obtenues malgré la complexité. Dans le contexte suisse, afin d'être certifié, tout système doit recourir à des méthodes formelles afin de démontrer la sécurité du protocole [Swi13].

La tâche principale dans l'application des méthodes formelles consiste à traduire la spécification et les propriétés en un modèle exact et fidèle. Dans ce modèle, un raisonnement rigoureux devient possible. La qualité des garanties obtenues dependront alors de la qualité, i.e., fidélité du modèle avec la réalité. Deux grandes approches de modélisation formelle existent : les modèles calculatoires et les modèles symboliques. Les modèles calculatoires supposent un attaquant qui s'exécute en temps polynomial et réduisent la sécurité d'un système concret à des hypothèses cryptographiques précises, telles que l'hypothèse du logarithme discret.[4] Cette approche repose généralement sur des preuves papier-crayon (e.g., [Swi25c, HKK+22, BCG+18a]), bien que certains outils semi-automatisés existent, tels que EasyCrypt [BDG+12], CryptoVerif [Bla07] et Squirrel [BDJ+24]. Les modèles symboliques sont plus abstraits et supposent une cryptographie parfaite (e.g., si l'adversaire ne connaît pas la clé de déchiffrement, il ne peut rien apprendre du message en clair). En contrepartie, ces abstractions permettent une automatisation accrue. Cela permet de prouver la sécurité de systèmes plus complexes, et des outils de vérification symbolique matures tels que ProVerif [Bla16, BCC22] et Tamarin [MSCB13] ont été appliqués avec succès à l'analyse de protocoles industriels à grande échelle, tels que TLS [BBK17, CHSvdM16, CHH+17], la 5G [BDH+18], Signal [KBB17] ou le protocole de paiement EMV [BHSTP24]. En particulier, ProVerif a été utilisé pour analyser des protocoles de vote [Swi25d, CCD23, CDGL24], et constitue également l'outil sur lequel nous nous concentrons dans ce travail.

Les protocoles sont modélisés dans ProVerif à l'aide du π calcul appliqué [AF01]. Les propriétés peuvent être exprimées sous forme de requêtes

[4] L'hypothèse du logarithme discret stipule qu'il est difficile de retrouver x à partir d'un $y = g^x$ donné. Ici, y est un élément d'un grand groupe cyclique multiplicatif fini, et g en est le générateur.

d'atteignabilité (e.g., déterminer si l'adversaire apprend une valeur) ou de requêtes de correspondance (e.g., vérifier qu'une relation donnée entre des états du protocole est satisfaite). En outre, une notion d'équivalence, ou indistinguabilité, peut être démontrée, ce qui est particulièrement utile pour les propriétés de confidentialité. ProVerif modélise un attaquant de type Dolev–Yao [DY03], c'est-à-dire un attaquant capable de créer, supprimer ou modifier des messages, et de dériver toute valeur à partir de sa connaissance, mais incapable de casser la cryptographie. Pour effectuer une preuve, ProVerif évalue si l'attaquant peut violer les propriétés dans le modèle du protocole. En interne, ProVerif transforme le protocole et le modèle d'attaquant en un ensemble de clauses de Horn noté $\mathbb{C}$. Ces clauses sont ensuite saturées en $\mathbb{C}_{sat}$, c'est-à-dire combinées entre elles afin de déduire d'autres clauses logiquement impliquées. Ce processus se poursuit jusqu'à ce qu'aucune nouvelle clause (au sens sémantique) n'émerge. Enfin, il est vérifié dans $\mathbb{C}_{sat}$ si les propriétés considérées sont satisfaites. ProVerif est prouvé correct (*sound*) : s'il ne trouve pas d'attaque dans le modèle, il n'en existe pas. En revanche, ProVerif peut produire de fausses attaques ou ne pas terminer. Dans ces cas, ProVerif fournit des mécanismes permettant de guider (de manière sound) son exécution, lesquels ont notamment été étendus récemment pour prendre en charge des lemmes [BCC22]. Les lemmes spécifient des propriétés intermédiaires que ProVerif peut utiliser durant son exécution, mais qui sont néanmoins préalablement prouvées vraies par ProVerif (c'est-à-dire que des lemmes non valides conduisent à l'échec de la preuve).

Les preuves formelles de la sécurité des protocoles de vote par Internet poussent les outils existants à leurs limites. La vérifiabilité de bout en bout (E2E-verifiability) relie les votes inclus dans le résultat aux votes effectivement exprimés par les électeurs, ce qui requiert intuitivement de compter les votes. Cette tâche est difficile pour les outils de preuve automatique en général, et plus encore dans le contexte complexe du vote électronique, où le modèle vise à être non borné en nombre d'élections, d'électeurs, de bulletins et même de revotes. Récemment, un framework de modélisation en ProVerif a été proposé pour faciliter les preuves de vérifiabilité [CCD23], en formulant la vérifiabilité E2E comme une propriété de correspondance. Ce framework encode un contexte électoral extensible (non borné) dans lequel des modèles de protocoles concrets peuvent être instanciés. Il fournit en outre des lemmes génériques afin d'aider à la terminaison et à la précision des preuves. Ce framework a été utilisé avec succès pour démontrer la vérifiabilité de plusieurs protocols, notamment Belenios [CGG19], Swiss Post Voting System [Swi24c] et CHVote [HKLD17]. Outre la vérifiabilité, les protocoles de vote par Internet nécessitent également une preuve de confidentialité. La confidentialité doit être modélisée comme une propriété d'équivalence, laquelle est notoirement plus difficile à vérifier que les pro-

priétés de correspondance [BS16, BDS15]. Les preuves de confidentialité existantes sont souvent modélisées séparément de la vérifiabilité [BCG+18a, HC19] et peuvent inclure des simplifications supplémentaires (e.g., l'omission de la vérification par l'électeur [HC19]).

C.5 Contributions

C.5.1 Contributions présentées dans cette thèse

Dans cette thèse, nous présentons les contributions suivantes.

Systematization of knowledge : mécanismes utilisés en pratique Alors que de nombreux mécanismes on été proposés concevoir des protocoles de vote par par Internet, à la fois respectueux de la confidentialité et vérifiables, nous nous intéressons aux mécanismes de sécurité effectivement utilisés aujourd'hui par les fournisseurs. Nous menons une étude de marché et passons en revue la littérature académique afin d'identifier les systèmes utilisés en pratique. Cela conduit à la collecte de 82 systèmes candidats, parmi lesquels nous isolons 29 systèmes de vote par Internet qui sont à la fois en usage actif et revendiquent une forme de vérifiabilité. Parmi ceux-ci, nous caractérisons et systématisons les 18 systèmes qui fournissent suffisamment d'informations pour permettre l'extraction de leurs mécanismes de sécurité.

Notre travail met en évidence que la majorité des systèmes proposant le vote par Internet sont en pratique des boîtes noires, dont le fonctionnement interne est inconnu, y compris des systèmes qui revendiquent pourtant une certaine forme de vérifiabilité. En outre, les quelques systèmes plus transparents présentent parfois un écart significatif entre les mécanismes qu'ils utilisent et ce qui peut être considéré comme l'état de l'art (par exemple, la génération de clés distribuée n'est utilisée que par environ la moitié de ces systèmes). Parmi les 18 systèmes analysés en détail, nous considérons que 8 sont bien documentés, et ces systèmes mettent globalement en œuvre des mécanismes conformes à l'état de l'art. Nous détaillons notre méthode et nos résultats dans le Chapter 2. Ce travail a été publié à E-Vote-ID 2024, en collaboration avec Michael Kirsten et Felix Dörre [MKD24].

Codes de vote courts La Chancellerie suisse a développé des exigences élevées en matière de sécurité pour le vote électronique. Il est intéressant de noter que les hypothèses de confiance concernant l'appareil de vote sont quelque peu incohérentes : alors que, pour la vérifiabilité, l'appareil de vote ne peut pas être supposé digne de confiance, il l'est en revanche pour la confidentialité. Nous cherchons à éliminer cette hypothèse de confiance, et donc à garantir la confidentialité du vote même face à un dispositif de vote malhonnête.

Nous atteignons cet objectif en utilisant le vote par codes (*code voting*). De plus, nous faisons en sorte que les codes soient très courts (généralement un ou deux chiffres), et donc adaptés à une saisie manuelle par l'électeur. Le protocole reste simple — en particulier, il ne repose que sur des primitives standards — surtout en comparaison avec d'autres protocoles conçus dans le dans le même contexte [HKD23, HKLD17, Swi24c]. Nous détaillons le protocole dans le Chapter 3 et en prouvons sa sécurité dans le Chapter 4. Ce travail a été publié à ESORICS 2024, en collaboration avec Véronique Cortier et Alexandre Debant [CDM24].

Framework de preuve Afin de prouver la vérifiabilité, des travaux récents ont proposé un cadre (framework) permettant de modéliser des protocoles de votes dans un contexte électroral extensible [CCD23]. Ce framework fournit en outre des outils limitant les problèmes de fausse attaque ou non-terminaison lors des preuves, en trouvant un juste équilibre entre terminaison et précision. Toutefois, il ne permet pas de prouver la confidentialité, laquelle requiert une approche de preuve fondamentalement différente (équivalence plutôt qu'atteignabilité) et repose généralement sur des techniques de modélisation différentes.

Nous nous appuyons sur ces travaux et proposons un framework permettant de traiter à la fois la vérifiabilité et la confidentialité. Celui-ci modélise un contexte électoral avec un nombre non borné d'élections concurrentes, d'électeurs et de votes. Il fournit en outre une bibliothèque de lemmes permettant d'équilibrer la terminaison et la précision du modèle. De manière cruciale, ce framework permet de réutiliser le même modèle du protocole pour les preuves de confidentialité et de vérifiabilité. Il est par ailleurs rétrocompatible avec les modèles développés dans le framework de vérifiabilité existant. Nous appliquons ce framework de preuve à plusieurs protocoles issus de la littérature et de l'industrie, notamment Belenios, CHVote, Swiss Post Voting System et la proposition des Codes de vote courts présenté dans le Chapter 3. Nous détaillons le framework et les études de cas dans le Chapter 4.

C.5.2 Contributions ne faisant pas partie de cette thèse

Nous avons choisi de ne pas présenter les contributions suivantes dans le manuscrit.

BSI StuVe Commandée par le BSI, l'Office fédéral allemand pour la sécurité de l'information, l'étude explore l'état de l'art actuel des mécanismes utilisés dans le vote par Internet vérifiable. Sur la base d'une étude de la littérature

et du marché[5], huit mécanismes utilisés dans le vote par Internet vérifiable ont été sélectionnés par le BSI pour être évalués. Les mécanismes retenus comprennent le chiffrement à clé publique malléable, les mises en gage malléables, l'agrégation homomorphe, les réseaux de mélange vérifiables, les signatures numériques, les mécanismes *audit-or-cast*, *cast-then-audit* et les codes de retour.

L'étude permet de comparer ces mécanismes entre eux sur la base de critères clairement définis couvrant la confidentialité, la vérifiabilité, l'utilisabilité et la praticabilité. Nous présentons des variantes des mécanismes lorsqu'elles permettent d'atteindre des compromis différents entre ces propriétés. Pour la confidentialité et la vérifiabilité, nous recensons les hypothèses de confiance nécessaires à l'établissement des garanties. Pour la confidentialité, nous évaluons en outre si le mécanisme présente des propriétés intéressantes vis-à-vis de l'absence de reçu ou de la confidentialité pérenne. Pour évaluer l'utilisabilité, nous réalisons une évaluation expertise visant à déterminer si les mécanismes sont faciles à exécuter pour l'électeur, ce que nous considérons comme acquis lorsque le mécanisme est à la fois efficient (peu d'étapes à exécuter) et efficace (ces étapes sont simples à réaliser). Pour évaluer la praticabilité, nous mettons en évidence l'implémentabilité (difficulté de développement) et l'efficacité (coûts computationnels et de communication).

L'étude conclut qu'au regard de l'état de l'art actuel, il est praticable de mettre en œuvre un dépouillement vérifiable préservant la confidentialité, couplé à une vérification du dispositif de vote de l'électeur. Elle souligne en outre que, lors de l'évaluation d'un système de vote et de ses hypothèses de confiance, le système doit être évalué dans son ensemble, et non pas uniquement les mécanismes pris isolément. Enfin, elle documente que la recherche a déjà identifié des approches permettant de préserver la confidentialité lors d'un dépouillement vérifiable face à des adversaires quantiques.

L'étude a été publiée par le BSI en 2024 [MMC+24] et constitue un travail conjoint avec Johannes Müller, Véronique Cortier, Alexandre Debant, Pierrick Gaudry, Anselme Goetschmann, Ralf Küsters et Melanie Volkamer. En outre, un article court faisant référence à cette étude a été publié à E-Vote ID 2025 afin de lui donner davantage de visibilité tout en mettant en évidence ses forces et ses faiblesses [CDK+25], en collaboration avec Véronique Cortier, Alexandre Debant, Ralf Küsters, Johannes Müller et Melanie Volkamer. L'étude a également conduit à la publication d'un guide destiné aux organisateurs d'élections sur la manière d'organiser des élections vérifiables, rédigé et publié par le BSI [Fed25b], le brouillon du guide ayant été préparé dans le cadre du projet BSI StuVe. Le projet com-

[5] L'étude de marché a été intégrée dans la systématisation des connaissances sur les mécanismes utilisés en pratique, présentée dans cette thèse au Chapter 2.

prenant l'étude et l'ébauche du guide a été co-dirigé par Johannes Müller et moi-même.

Recommandations pour la mise en œuvre d'une vérifiabilité individuelle indépendante Le mécanisme *cast-then-audit* permet à l'électeur d'auditer les bulletins émis par son dispositif de vote. Cet audit doit être réalisé sur un autre dispositif, indépendant de celui potentiellement compromis de l'électeur. Un niveau de confiance supplémentaire est obtenu en exécutant l'audit au moyen d'implémentations indépendantes, afin de se prémunir contre un opérateur du système principal malhonnête. J'ai mis en œuvre une telle implémentation indépendante, publiée à l'adresse famoser/polyas-verification, pour les élections de la Gesellschaft für Informatik e.V. qui utilisent le système POLYAS [Tru23]. Cette implémentation a été mise à disposition des électeurs lors des élections annuelles depuis 2023, en complément d'autres implémentations. À notre connaissance, il s'agit du premier projet de ce type offrant aux électeurs une vérification *cast-as-intended* indépendante.

Sur la base des expériences réalisées en 2023, nous formulons des recommandations à l'intention de futurs projets souhaitant mettre en œuvre des outils de vérification indépendants [MGH+24]. Ce travail rappelle le contexte du projet, puis formule des recommandations concrètes couvrant l'implémentation, les tests, la qualité et les responsabilités. Chaque recommandation est contextualisée par des expériences correspondantes issues du projet. Ces recommandations ont été publiées à E-Vote ID 2023, dans le cadre d'un travail conjoint avec Rüdiger Grimm, Tobias Hilt, Michael Kirsten, Christoph Niederbudde et Melanie Volkamer.

Développement et évaluation expertise d'une vidéo informative L'usage continu de systèmes opaques et non vérifiables suggère que les organisateurs d'élections peuvent ne pas être conscients des défis de sécurité liés au vote par Internet, ni des stratégies d'atténuation qui ont été développées. Pour répondre à ce problème, nous avons développé une vidéo informative sur ce thème à destination d'organisateurs d'élections non experts en vote par Internet. Dans cette vidéo, nous expliquons les défis généraux et les solutions existantes, tout en soulignant qu'il n'existe pas d'approche unique optimale, chacune présentant ses propres avantages et inconvénients. Comme fil conducteur, nous utilisons un système de vote par codes conçu pour être simple à expliquer et à illustrer. Nous simplifions le sujet de manière à le rendre compréhensible pour des non-experts, tout en veillant à ce que cette simplification ne conduise pas à des malentendus ni à des affirmations incorrectes.

Nous documentons la méthode de création de la vidéo et faisons évaluer celle-ci par 19 experts germanophones du vote par Internet [HMMV25]. Sur la base de leurs retours, nous proposons des améliorations visant à

renforcer la justesse, la clarté et l'exhaustivité de la vidéo. Par ailleurs, le développement de la vidéo suivi de son évaluation experte a fourni de précieux enseignements, par exemple le fait qu'une grande partie (mais pas la totalité) des retours aurait pu être résolue dès l'analyse du seul script textuel. Ce travail a été publié à E-Vote-ID 2025 et a reçu le prix du *Distinguished Paper*, dans le cadre d'un travail conjoint avec Tobias Hilt et Melanie Volkamer.

Bug de concurrence dans le système de vote de la Poste suisse Le système de vote de la Poste suisse est un système développé dans le contexte suisse. Le code du système est public, et la Poste suisse invite à la relecture et à la divulgation responsable des vulnérabilités dans le cadre d'un programme de récompense pour la découverte de failles [Swi25e].

Lors de l'inspection du code, nous avons découvert un bug permettant d'exécuter certains algorithmes plus souvent que spécifié, par exemple pour contourner le stockage d'un seul bulletin par électeur ou pour effectuer des attaques par force brute sur des secrets. Concrètement, pour chaque requête reçue via Internet, le serveur de vote exposé au réseau (non digne de confiance) prépare une commande contenant le contenu de la requête et l'envoie aux composants de contrôle [Swi25b] (dont l'un est supposé digne de confiance). Le composant de contrôle vérifie ensuite si une commande sémantiquement identique a déjà été exécutée et sinon procède à son exécution. Cependant, la vérification visant à déterminer si une commande avait déjà été exécutée incluait une valeur définie par le serveur de vote (non digne de confiance), ce qui permettait à ce dernier de faire exécuter par un composant de contrôle honnête une commande sémantiquement équivalente un nombre arbitraire de fois. Nous avons confirmé l'existence du bug en implémentant un test unitaire correspondant dans la base de code de la Poste suisse.

Lors de l'analyse de l'impact du bug, parmi les six algorithmes potentiellement affectés, nous avons étudié en détail deux exploitations concrètes possibles, qui n'ont toutefois pas conduit à une attaque réussie :

- `partialDecryptPCC` : cette fonction stocke le bulletin de l'électeur et déclenche la génération des codes de retour. L'invocation multiple de cette fonction pourrait permettre de stocker plusieurs bulletins pour un même électeur. Le bug n'a pas pu être exploité en raison d'une contrainte de clé primaire dans la base de données, qui empêchait le stockage de plusieurs bulletins par électeur.
- `createLVCCShare` : cette fonction vérifie la validité du code de confirmation, avec une limite supérieure sur le nombre de tentatives autorisées par électeur. L'invocation multiple de cette fonction pourrait permettre de tester davantage de codes de confirmation que ce qui est

configuré. Le nombre de tentatives est stocké dans la base de données, avec une gestion des versions assurée par la couche d'abstraction de la base de données (attribut `@VersionId` géré par Hibernate [Hib25]). Ce mécanisme de versionnement a empêché l'exploitation, car lors de l'enregistrement (c'est-à-dire l'écrasement) du nombre de tentatives issues d'exécutions concurrentes, l'implémentation a levé une exception `StaleStateException` dans toutes les exécutions sauf la première à se terminer.

Le bug a été divulgué de manière responsable et classé comme de sévérité élevée. Il a été corrigé en l'espace d'une semaine, et la Poste suisse a versé une récompense de 10 000 €. De plus amples détails concernant le bug et les exploitations potentielles sont disponibles à l'adresse gitlab.com:swisspost-evoting/e-voting/e-voting#9.

C.5.3 Contributions logicielles

En outre, notre travail a conduit à plusieurs résultats pratiques. Nous avons utilisé ProVerif de manière intensive et avons proposé puis implémenté certaines améliorations, soit directement dans l'outil, soit dans l'écosystème qui l'entoure. Par ailleurs, lors de la mise en œuvre de la vérification individuelle pour le système POLYAS, nous avons découvert et corrigé des problèmes dans PHP.

Améliorations de ProVerif Certaines améliorations superficielles de la grammaire la rendent plus permissive (#aa34a4c9, #3e9e6af7, #f03b9136), supprimant ainsi certaines sources d'irritation liées à l'autorisation (ou non) des points-virgules et des points dans les modèles. Nous avons également ajouté une option en ligne de commande *-parse-only* permettant d'analyser un modèle sans lancer immédiatement son exécution (#a6a4f059, #d2776e8a).[6] Comme modification moins superficielle, nous avons contribué au développement d'une amélioration facilitant le débogage des preuves de type équivalence : lorsque ProVerif n'est pas en mesure de prouver une équivalence, il fournit désormais des informations supplémentaires expliquant pourquoi la preuve échoue (#f03b9136).[7] Enfin, nous avons implémenté une amélioration rendant les lemmes plus expressifs, que nous détaillerons au Chapter 4. Toutes ces améliorations ont déjà été intégrées dans la branche *master* et seront donc incluses dans les prochaines versions de ProVerif.

[6] Cette option est notamment utilisée dans le serveur de langage VSCode, décrit dans le paragraphe suivant.

[7] En interne, ProVerif raisonne sur des clauses. Lorsqu'une clause conduit à l'échec d'une preuve d'équivalence, cette modification garantit que ProVerif indique comment cette clause a été dérivée.

Outillage autour de ProVerif Au cours de notre travail avec ProVerif, nous avons développé des outils afin de rendre notre flux de travail plus efficace et plus sûr. Un effort majeur a été consacré au développement d'un serveur de langage ProVerif pour VSCode, qui apporte une coloration syntaxique sémantique, des fonctionnalités de navigation et des renommages sûrs (voir ProVerif/vscode-proverif-language-service). Cette extension est disponible publiquement [Pro25a, Pro25b]. Pour diagnostiquer les problèmes de non-terminaison, nous avons développé famoser/proverif-terminator, un outil capable d'identifier des cas simples de non-terminaison et d'afficher de manière sélective la sortie de ProVerif. Au cours du développement du framework, nous avons également développé un cadre de test, qui s'est avéré utile à la fois pour tester efficacement les itérations successives du framework et pour documenter les exécutions des modèles.

Correctifs PHP Dans le cadre de la mise en œuvre de la vérification individuelle pour les élections de la Gesellschaft für Informatik e.V. qui utilisent le système POLYAS [Tru23], des bugs ont été découverts dans le langage PHP, choisi pour l'implémentation. Nous avons activement contribué au diagnostic et à la résolution de ces bugs, ainsi qu'à l'ajout de tests unitaires appropriés en PHP afin d'éviter que des problèmes similaires ne se reproduisent. Nous décrivons ces bugs et notre travail au Section A.1. PHP est le langage le plus utilisé côté serveur.[8]

[8] Voir par exemple les statistiques de w3tech.

Bibliography

[AC11] Jordi Puiggalí Allepuz and Sandra Guasch Castelló. Internet voting system with cast as intended verification. In *International Conference on E-Voting and Identity*, pages 36–52. Springer, 2011.

[AC12] Jordi Puiggalí Allepuz and Sandra Guasch Castelló. Cast-as-intended verification in Norway. In *Proc. 5th Conf. Electron. Voting*, pages 49–63. Citeseer, 2012.

[ACKR13] Myrto Arapinis, Véronique Cortier, Steve Kremer, and Mark Ryan. Practical everlasting privacy. In *International Conference on Principles of Security and Trust*. Springer, 2013.

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security 2008: 17th USENIX Security Symposium*, pages 335–348, San Jose, CA, USA, July 28 – August 1, 2008. USENIX Association. URL: http://www.usenix.org/events/sec08/tech/full_papers/adida/adida.pdf.

[Adi12] Ben Adida. Helios v4. Technical report, Helios, August 2012. URL: https://web.archive.org/web/20240417102302/https://documentation.heliosvoting.org/verification-specs/helios-v4.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. *ACM Sigplan Notices*, 36(3):104–115, 2001.

[Ann24a] Annee Politique Suisse. Pilotbetrieb für E-Collecting mit der E-ID-Vertrauensinfrastruktur. https://anneepolitique.swiss/prozesse/68241, Sep 2024.

[Ann24b] Annee Politique Suisse. Missbräuchliche Unterschriftensammlungen. https://anneepolitique.swiss/prozesse/68174, Sep 2024.

[Ass23] Assembly Voting. Electa: Documentation of the cryptographic protocol (version 2.0). Technical report, Assembly Voting, 3 2023. URL: https://web.archive.org/web/20240714102041/https://downloads.assembly-voting.com/download/marketing/electa_-_documentation_of_the_cryptographic_protocol.pdf.

[Ass24] Assembly Voting. Assembly voting SDKs, 2024. URL: https://github.com/aion-dk.

[BBH+17] Matthew Bernhard, Josh Benaloh, J Alex Halderman, Ronald L Rivest, Peter YA Ryan, Philip B Stark, Vanessa Teague, Poorvi L Vora, and Dan S Wallach. Public evidence from secret ballots. In *International Joint Conference on Electronic Voting*, pages 84–109. Springer, 2017.

[BBK17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy*, pages 483–502, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press. doi:10.1109/SP.2017.26.

[BBMP20] Sevdenur Baloglu, Sergiu Bursuc, Sjouke Mauw, and Jun Pang. Election verifiability revisited: Automated security proofs and attacks on helios and belenios. Cryptology ePrint Archive, Report 2020/982, 2020. URL: https://eprint.iacr.org/2020/982.

[BCC22] Bruno Blanchet, Vincent Cheval, and Véronique Cortier. ProVerif with lemmas, induction, fast subsumption, and much more. In *2022 IEEE Symposium on Security and Privacy*, pages 69–86, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press. doi:10.1109/SP46214.2022.9833653.

[BCG+18a] David Bernhard, Véronique Cortier, Pierrick Gaudry, Mathieu Turuani, and Bogdan Warinschi. Verifiability analysis of CHVote. report, Bernhard, David and Cortier, Véronique and Gaudry, Pierrick and Turuani, Mathieu and Warinschi, Bogdan, 2018.

[BCG+18b] David Bernhard, Véronique Cortier, Pierrick Gaudry, Mathieu Turuani, and Bogdan Warinschi. Verifiability analysis of CHVote. Cryptology ePrint Archive, Report

2018/1052, 2018. URL: https://eprint.iacr.org/2018/1052.

[BDG+12] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. *International School on Foundations of Security Analysis and Design*, pages 146–166, 2012.

[BDH+18] David A. Basin, Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5G authentication. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1383–1396, Toronto, ON, Canada, October 15–19, 2018. ACM Press. doi:10.1145/3243734.3243846.

[BDJ+24] David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, and Joseph Lallemand. The squirrel prover and its logic. *ACM SIGLOG News*, 11(2):62–83, 2024.

[BDS15] David A. Basin, Jannik Dreier, and Ralf Sasse. Automated symbolic proofs of observational equivalence. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 1144–1155, Denver, CO, USA, October 12–16, 2015. ACM Press. doi:10.1145/2810103.2813662.

[Bel24a] Belenios. Belenios code, 2024. URL: https://gitlab.inria.fr/belenios/belenios/.

[Bel24b] Belenios. Belenios documentation, 2024. URL: https://web.archive.org/web/20240708065438/https://www.belenios.org/documentation.html.

[Ben87] J. Benaloh. *Verifiable secret-ballot elections*. PhD thesis, Yale University, 1987.

[Ben06] Josh Benaloh. Simple verifiable elections. *EVT*, 2006.

[BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237

of *Lecture Notes in Computer Science*, pages 263–280, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-29011-4_17.

[BGS+16] Christian Bull, Kristian Gjøsteen, Katrine Stemland Skjelsvik, Ida Sofie, and Gebhardt Stenerud. The imperfect storm. In *The International Conference on Electronic Voting: E-Vote-ID*, pages 116–120, 2016.

[BHSTP24] David Basin, Xenia Hofmeier, Ralf Sasse, and Jorge Toro-Pozo. Getting chip card payments right. In *International Symposium on Formal Methods*, pages 29–51. Springer, 2024.

[Big24a] BigPulse. Election vote count verification protocol, 2024. URL: https://www.bigpulsevoting.com/about/vote-count-verification-protocol/.

[Big24b] BigPulse. Secure misc voting system, 2024. URL: https://www.bigpulsevoting.com/.

[Bla07] Bruno Blanchet. Cryptoverif: Computationally sound mechanized prover for cryptographic protocols. In *Dagstuhl seminar "Formal Protocol Verification Applied*, volume 117, page 156, 2007.

[Bla16] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and ProVerif. *Foundations and Trends in Privacy and Security*, 1:1–135, 2016.

[Bla23] Bruno Blanchet. ProVerif 2.05: automatic cryptographic protocol verifier, user manual and tutorial. Technical report, INRIA, October 2023. URL: https://bblanche.gitlabpages.inria.fr/proverif/manual.pdf.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-45682-1_30.

[BNOV13] Jurlind Budurushi, Stephan Neumann, Maina M Olembo, and Melanie Volkamer. Pretty Understandable Democracy - A Secure and Understandable Internet Voting Scheme. In *International Conference on Availability, Reliability and Security (ARES)*, 2013.

[BNPW24] Josh Benaloh, Michael Naehrig, Olivier Pereira, and Dan S. Wallach. ElectionGuard: a cryptographic toolkit to enable verifiable elections. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/benaloh.

[BRR+15] Josh Benaloh, Ronald Rivest, Peter YA Ryan, Philip Stark, Vanessa Teague, and Poorvi Vora. End-to-end verifiability. *arXiv preprint arXiv:1504.03778*, 2015.

[BS16] Bruno Blanchet and Ben Smyth. Automated reasoning for equivalences in the applied pi calculus with barriers. In Michael Hicks and Boris Köpf, editors, *CSF 2016: IEEE 29th Computer Security Foundations Symposium*, pages 310–324, Lisbon, Portugal, June 27–1, 2016. IEEE Computer Society Press. doi:10.1109/CSF.2016.29.

[BT94] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *26th Annual ACM Symposium on Theory of Computing*, pages 544–553, Montréal, Québec, Canada, May 23–25, 1994. ACM Press. doi:10.1145/195058.195407.

[Bun13] Schweizerischer Bundesrat. Bericht des Bundesrates zu Vote électronique. report, Schweizerischer Bundesrat, June 2013. Accessed at 2021-05-18. URL: https://www.bk.admin.ch/dam/bk/de/dokumente/pore/Vote--electronique/bericht_des_bundesrateszuvoteelectronique-auswertungdereinfuehru.pdf.download.pdf/bericht_des_bundesrateszuvoteelectronique-auswertungdereinfuehru.pdf.

[Bun19a] Schweizerische Bundeskanzlei. E-Voting: Bundesrat richtet Versuchsbetrieb neu aus und stellt Einführung

als ordentlicher Stimmkanal zurück., June 2019. Accessed at 2021-10-11. URL: https://www.bk.admin.ch/bk/de/home/dokumentation/medienmitteilungen.msg-id-75615.html.

[Bun19b] Schweizerische Bundeskanzlei. Medienmittelung: Bundeskanzlei nimmt Standortbestimmung zum E-Voting vor., March 2019. Accessed at 2021-05-19. URL: https://www.bk.admin.ch/bk/de/home/dokumentation/medienmitteilungen.msg-id-74508.html.

[Bun21] Schweizerische Bundeskanzlei. Teilrevision der Verordnung über die politischen Rechte und Totalrevision der Verordnung der BK über die elektronische Stimmabgabe (Neuausrichtung des Versuchsbetriebs): Stellungnahmen. report, Schweizerische Bundeskanzlei, August 2021. Accessed at 2021-10-22. URL: https://fedlex.data.admin.ch/filestore/fedlex.data.admin.ch/eli/dl/proj/2021/61/cons_1/doc_8/de/pdf-a/fedlex-data-admin-ch-eli-dl-proj-2021-61-cons_1-doc_8-de-pdf-a.pdf.

[CCD23] Vincent Cheval, Véronique Cortier, and Alexandre Debant. Election verifiability with ProVerif. In *CSF 2023: IEEE 36th Computer Security Foundations Symposium*, pages 43–58, Dubrovnik, Croatia, July 10–14, 2023. IEEE Computer Society Press. doi:10.1109/CSF57540.2023.00032.

[CCDM26] Vincent Cheval, Veronique Cortier, Alexandre Debant, and Florian Moser. Simultaneously proving privacy and verifiability: A proverif framework for internet voting. In *IEEE Computer Security Foundations Symposium*, 2026.

[CCFG16] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. BeleniosRF: A non-interactive receipt-free electronic voting scheme. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1614–1625, Vienna, Austria, October 24–28, 2016. ACM Press. doi:10.1145/2976749.2978337.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *2008*

IEEE Symposium on Security and Privacy, pages 354–368, Oakland, CA, USA, May 18–21, 2008. IEEE Computer Society Press. doi:10.1109/SP.2008.32.

[CCT18] Vincent Cheval, Véronique Cortier, and Mathieu Turuani. A little more conversation, a little less action, a lot more satisfaction: Global states in ProVerif. In Steve Chong and Stephanie Delaune, editors, *CSF 2018: IEEE 31st Computer Security Foundations Symposium*, pages 344–358, Oxford, UK, July 9–12, 2018. IEEE Computer Society Press. doi:10.1109/CSF.2018.00032.

[CDE+25] Véronique Cortier, Alexandre Debant, Olivier Esseiva, Pierrick Gaudry, Audhild Høgåsen, and Chiara Spadafora. A practical and fully distributed e-voting protocol for the swiss context. *Cryptology ePrint Archive*, 2025.

[CDG22] Véronique Cortier, Alexandre Debant, and Pierrick Gaudry. A privacy attack on the Swiss Post e-voting system. In *Real World Crypto Symposium (RWC'22)*, Amsterdam, Netherlands, 2022. IACR.

[CDG25] Véronique Cortier, Alexandre Debant, and Pierrick Gaudry. Chvote: breaking privacy. In *European Symposium on Research in Computer Security (Esorics'25*. Springer, 2025.

[CDGG23] Véronique Cortier, Alexandre Debant, Pierrick Gaudry, and Stéphane Glondu. Belenios with cast as intended. In *International Conference on Financial Cryptography and Data Security*, pages 3–18. Springer, 2023.

[CDGL24] Véronique Cortier, Alexandre Debant, Pierrick Gaudry, and Léo Louistisserand. Vote&check: Secure postal voting with reduced trust assumptions. Cryptology ePrint Archive, Report 2024/1951, 2024. URL: https://eprint.iacr.org/2024/1951.

[CDK+25] Veronique Corter, Alexandre Debant, Ralf Küsters, Florian Moser, Johannes Müller, and Melanie Volkamer. On a study of mechanisms for end-to-end verifiable online voting (stuve). In *EVote-ID 2025-10th International Joint Conference on Electronic Voting*, 2025.

[CDM24] Véronique Cortier, Alexandre Debant, and Florian Moser. Code voting: when simplicity meets security. In *European Symposium on Research in Computer Security*, pages 410–429. Springer, 2024.

[CF85] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *26th Annual Symposium on Foundations of Computer Science*, pages 372–382, Portland, Oregon, October 21–23, 1985. IEEE Computer Society Press. doi:10.1109/SFCS.1985.2.

[CFL19] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. BeleniosVS: Secrecy and verifiability against a corrupted voting device. In Stephanie Delaune and Limin Jia, editors, *CSF 2019: IEEE 32nd Computer Security Foundations Symposium*, pages 367–381, Hoboken, NJ, USA, June 25–28, 2019. IEEE Computer Society Press. doi:10.1109/CSF.2019.00032.

[CGG19] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. Belenios: A simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows*, volume 11565 of *LNCS*, pages 214–238. Springer, 2019. doi:10.1007/978-3-030-19052-1_14.

[CGGI13] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Distributed ElGamal à la Pedersen: Application to Helios. In Ahmad-Reza Sadeghi and Sara Foresti, editors, *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*, pages 131–142. ACM, 2013. doi:10.1145/2517840.2517852.

[CGGI14] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for helios under weaker trust assumptions. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014: 19th European Symposium on Research in Computer Security, Part II*, volume 8713 of *Lecture Notes in Computer Science*, pages 327–344, Wroclaw, Poland, September 7–11, 2014. Springer, Cham, Switzerland. doi:10.1007/978-3-319-11212-1_19.

[CGK+16] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Mueller, and Tomasz Truderung. SoK: Verifiability notions for E-voting protocols. In *2016 IEEE Symposium on Security and Privacy*, pages 779–798, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press. doi:10.1109/SP.2016.52.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118, Konstanz, Germany, May 11–15, 1997. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-69053-0_9.

[CGY22] Véronique Cortier, Pierrick Gaudry, and Quentin Yang. A toolbox for verifiable tally-hiding E-voting systems. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022: 27th European Symposium on Research in Computer Security, Part II*, volume 13555 of *Lecture Notes in Computer Science*, pages 631–652, Copenhagen, Denmark, September 26–30, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-17146-8_31.

[Cha01] David Chaum. Sure Vote: Technical Overview. In *Workshop on trustworthy elections (WOTE)*, 2001. URL: http://www.vote.caltech.edu/wote01/pdfs/surevote.pdf.

[Che23] Benoit Chenon. MEAE - Transparence et vérifiabilité V2 (version 2.04). Technical report, Voxaly, 2 2023. URL: https://web.archive.org/web/20240308073039/https://www.voxaly.com/wp-content/uploads/VOXALY-LEG2023-Transparence-et-Verifiabilite-Specifications-publiques-v2-04.pdf.

[CHH+17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1773–1788, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. doi:10.1145/3133956.3134063.

[CHSvdM16] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *2016 IEEE Symposium on Security and Privacy*, pages 470–485, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press. doi:10.1109/SP.2016.35.

[CL18] Véronique Cortier and Joseph Lallemand. Voting: You can't have privacy without individual verifiability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 53–66, Toronto, ON, Canada, October 15–19, 2018. ACM Press. doi:10.1145/3243734.3243762.

[CLD05] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property: How to get rid of some algebraic properties. In *Term Rewriting and Applications, 16th International Conference, (RTA'05)*, 2005.

[CLW20] Véronique Cortier, Joseph Lallemand, and Bogdan Warinschi. Fifty shades of ballot privacy: Privacy against a malicious board. In Limin Jia and Ralf Küsters, editors, *CSF 2020: IEEE 33rd Computer Security Foundations Symposium*, pages 17–32, Boston, MA, USA, June 22–26, 2020. IEEE Computer Society Press. doi:10.1109/CSF49147.2020.00010.

[CMFP+10] Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. On some incompatible properties of voting schemes. In *Towards Trustworthy Elections*, pages 191–199. Springer, 2010.

[Com19a] New South Wales (AU) Electoral Commission. NSW Electoral Commission iVote and Swiss Post e-voting, March 2019. Accessed at 2021-09-23. URL: https://www.elections.nsw.gov.au/About-us/Media-centre/News-media-releases/NSW-Electoral-Commission-iVote-and-Swiss-Post-e-vo.

[Com19b] Commission nationale de l'informatique et des libertés. Délibération n° 2019-053 du 25 avril 2019 portant adoption d'une recommandation relative à la sécurité des systèmes de vote par correspondance électronique, notamment via Internet. Technical report, Journal Officiel de la République Française, 4 2019. URL: https:

//www.legifrance.gouv.fr/jorf/id/JORFTEXT000038661239.

[Com19c] Commission nationale de l'informatique et des libertés. Sécurité des systèmes de vote par internet: la CNIL actualise sa recommandation de 2010, 7 2019. URL: https://www.cnil.fr/fr/securite-des-systemes-de-vote-par-internet-la-cnil-actualise-sa-recommandation-de-2010.

[Com23] NSW Electoral Commission. Electoral commissioner's determination - ivote will not be used for 2023 nsw state election. https://elections.nsw.gov.au/about-us/media-centre/news-and-media-releases/electoral-commissioner-ivote-determination, 2023.

[Con24] Conventex. Digitale Wahlen, 2024. URL: https://conventex.com/digitale-wahlen/.

[CPP13] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. Election verifiability or ballot privacy: Do we need to choose? In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 481–498, Egham, UK, September 9–13, 2013. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-40203-6_27.

[CS11] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. In Michael Backes and Steve Zdancewic, editors, *CSF 2011: IEEE 24th Computer Security Foundations Symposium*, pages 297–311, Domaine de l'Abbaye des Vaux de Cernay, France, June 27–29, 2011. IEEE Computer Society Press. doi:10.1109/CSF.2011.27.

[CZZ+16] Nikos Chondros, Bingsheng Zhang, Thomas Zacharias, Panos Diamantopoulos, Stathis Maneas, Christos Patsonakis, Alex Delis, Aggelos Kiayias, and Mema Roussopoulos. D-DEMOS: A distributed, end-to-end verifiable, internet voting system. In *36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016, Nara, Japan, June 27-30, 2016*, pages 711–720. IEEE Computer Society, 2016. doi:10.1109/ICDCS.2016.56.

[Cé23] Cécile. Privacy policy - V8TE. Technical report, V8TE, 1 2023. URL: https://web.archive.org/web/20240822085749/https://drive.usercontent.google.com/download?id=1SOxRXNdHsTazUjcJt1EOcYw3hA6fy5JG&export=download&authuser=0.

[Dec24a] DecentraVote. Decentralized e-voting protocol powered by blockchain, 2024. URL: https://decentra.vote/.

[Dec24b] DecentraVote. Decentravote-core, 2024. URL: https://github.com/DecentraVote-eVoting/DecentraVote-Core.

[DH23] Alexandre Debant and Lucca Hirschi. Reversing, breaking, and fixing the french legislative election E-voting protocol. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 6737–6752, Anaheim, CA, USA, August 9–11, 2023. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/debant.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.

[DMSTW12] Ardita Driza-Maurer, Oliver Spycher, Geo Taglioni, and Anina Weber. E-voting for Swiss Abroad: A Joint Project between the Confederation and the Cantons. In *5th International Conference on Electronic Voting 2012 (EVOTE2012)*. Gesellschaft für Informatik eV, 2012.

[Dou] Douglas Wikström. Open Verificatum. https://www.verificatum.org/.

[DRRR24] Aditya Damodaran, Simon Rastikian, Peter B. Rønne, and Peter Y A Ryan. Hyperion: Transparent end-to-end verifiable voting with coercion mitigation. Cryptology ePrint Archive, Report 2024/1182, 2024. URL: https://eprint.iacr.org/2024/1182.

[DY03] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 2003.

[Ele24a] Electis. Electis core, 2024. URL: https://gitlab.com/electisNGO/electis-core/.

[Ele24b] Electis. Vote électronique CSE, 2024. URL: https://www.tessi.eu/.

[Ele24c] Electric paper Wahlsysteme. Wahlen sicher und effizient durchführen, 2024. URL: https://wahlen-organisieren.de/.

[Eli24] Eligo. Electronic and misc voting platform, 2024. URL: https://www.eligo.social/.

[Est14] Estonia. Riigikogu Election Act. https://www.riigiteataja.ee/en/eli/510032014001/consolide, 2014.

[Est25] Estonian National Electoral Committee. Statistics about Internet Voting in Estonia, 2025. URL: https://www.valimised.ee/en/archive/statistics-about-internet-voting-estonia.

[Eur23] European Union. European Citizens' Initiative. https://citizens-initiative.europa.eu/how-it-works_en, 2023.

[EVo24a] EVoting. Electronic voting service, 2024. URL: https://www.evoting.com/.

[EVo24b] EVoting. Encryption and voting secrecy, 2024. URL: https://www.evoting.com/en/seguridad-integral/secreto-voto/.

[Faz20] Zoltan Fazekas. DecentraVote: Electronic voting secured by blockchain (version 1.0). Technical report, DecentraVote, 5 2020. URL: https://github.com/DecentraVote-eVoting/DecentraVote-Core/blob/72c0a20ed1764fb5e80d4475934965a2f57a743e/Whitepaper_DecentraVote.pdf.

[FCC24] Tamara Finogina, Jordi Cucurull Juan, and Nuria Costa. Selective comparison of verifiable misc voting systems. *Security and Privacy*, 2024. doi:10.1002/spy2.394.

[FDL07] Laure Fouard, Mathilde Duclos, and Pascal Lafourcade. Survey on electronic voting schemes. *supported by the ANR project AVOTÉ*, 2007.

[Fed24] Federal Chancellery Switzerland. Elektronische Unterschriftensammlung für eidgenössische Volksbegehren (E-Collecting): Bericht des Bundesrates in Erfüllung des Postulates 21.3607 Staatspolitische Kommission NR vom 27. Juni 2021. Technical report, Federal Government Switzerland, Nov 2024.

[Fed25a] Federal Council Switzerland. The Federal Council renews basic licences for online voting. https://www.news.admin.ch/en/newnsb/ZLw6w1GV_UdJKDocuT0sX, 2025.

[Fed25b] Federal Office for Information Security, Germany. Ende-zu-Ende Verifizierbare Onlinewahlen: Handlungsleitfaden für Wahlorganisatoren. Technical report, Federal Office for Information Security, Germany, 2025. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Wahlen/Onlinewahlen/Handlungsleitfaden_Onlinewahlen_Wahlorganisatoren.pdf.

[fK20] Bundesamt für Kommunikation. Strategie Digitale Schweiz. report, Bundesamt für Kommunikation, June 2020. Accessed at 2021-11-25. URL: https://www.bakom.admin.ch/dam/bakom/de/dokumente/informationsgesellschaft/strategie/strategie_digitale_schweiz.pdf.download.pdf/Strategie-DS-2020-De.pdf.

[FOfIS25] Germany Federal Office for Information Security. Zertifizierte Produkte - Sonstige Produkte. https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Zertifizierung-und-Anerkennung/Zertifizierung-von-Produkten/Zertifizierung-nach-CC/Zertifizierte-Produkte-nach-CC/Sonstiges/Sonstiges_node.html, nov 2025.

[Fol24a] Followmyvote. Blockchain voting: Cryptographically secure voting, 2024. URL: https://web.archive.org/web/20240820075328/https://followmyvote.com/cryptographically-secure-voting-2/.

[Fol24b] Followmyvote. Followmyvote smartcontract code, 2024. URL: https://github.com/FollowMyVote/Pollaris-Contract.

[Fre25] French Cybersecurity Agency. Recommandations pour la mise en œuvre du vote par internet pour les élections non-politiques. Technical report, French Cybersecurity Agency, 2025.

[Ged24] Gediote. Expert en solutions de vote, 2024. URL: https://www.gedivote.fr/.

[Ger21] Micha Germann. Internet voting increases expatriate voter turnout. *Government Information Quarterly*, 38(2):101560, Apr 2021. doi:10.1016/j.giq.2020.101560.

[GHS+20] Guillaume Girol, Lucca Hirschi, Ralf Sasse, Dennis Jackson, Cas Cremers, and David A. Basin. A spectral analysis of noise: A comprehensive, automated, formal analysis of Diffie-Hellman protocols. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020: 29th USENIX Security Symposium*, pages 1857–1874. USENIX Association, August 12–14, 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/girol.

[Glo24] Stéphane Glondu. Belenios specification (version 2.5.1). Technical report, Belenios, 10 2024. URL: https://web.archive.org/web/20240708064854/https://www.belenios.org/specification.pdf.

[Gro10] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology*, 23(4):546–579, October 2010. doi:10.1007/s00145-010-9067-9.

[GS17] Micha Germann and Uwe Serdült. Internet voting and turnout: Evidence from Switzerland. *Electoral Studies*, 47:1–12, 2017.

[HBT+24] Tobias Hilt, Benjamin Berens, Tomasz Truderung, Margarita Udovychenko, Stephan Neumann, and Melanie Volkamer. Systematic user evaluation of a second device based cast-as-intended verifiability approach. In *International Conference on Financial Cryptography and Data Security*, pages 33–49. Springer, 2024.

[HC19] Lucca Hirschi and Cas Cremers. Improving automated symbolic analysis of ballot secrecy for E-voting protocols: A method based on sufficient conditions. In *2019*

IEEE European Symposium on Security and Privacy, pages 635–650, Stockholm, Sweden, June 17–19, 2019. IEEE Computer Society Press. doi:10.1109/EuroSP.2019.00052.

[Hel24] Helios. Helios-server, 2024. URL: https://github.com/benadida/helios-server/tree/c7ed0608e2cdfc75bf323a834f0ca579e6273e34.

[Hib25] Hibernate. Hibernate. https://hibernate.org/, Dez 2025.

[HKD23] Rolf Haenni, Reto E Koenig, and Eric Dubuis. Private Internet Voting on Untrusted Voting Devices. In *International Conference on Financial Cryptography and Data Security*, 2023.

[HKK+22] Nicolas Huber, Ralf Küsters, Toomas Krips, Julian Liedtke, Johannes Müller, Daniel Rausch, Pascal Reisert, and Andreas Vogt. Kryvos: Publicly tally-hiding verifiable E-voting. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 1443–1457, Los Angeles, CA, USA, November 7–11, 2022. ACM Press. doi:10.1145/3548606.3560701.

[HKLD17] Rolf Haenni, Reto E. Koenig, Philipp Locher, and Eric Dubuis. CHVote system specification. Cryptology ePrint Archive, Report 2017/325, 2017. URL: https://eprint.iacr.org/2017/325.

[HKLR24] Nicolas Huber, Ralf Küsters, Julian Liedtke, and Daniel Rausch. Zk-snarks for ballot validity: A feasibility study. In *International Joint Conference on Electronic Voting*, pages 107–123. Springer, 2024.

[HKW20] Sven Heiberg, Kristjan Krips, and Jan Willemson. Planning the next steps for Estonian Internet voting. *E-Vote-ID 2020*, page 82, 2020.

[HL08] James Heather and David Lundin. The Append-Only Web Bulletin Board. In Pierpaolo Degano, Joshua D. Guttman, and Fabio Martinelli, editors, *Formal Aspects in Security and Trust, 5th International Workshop, FAST 2008, Malaga, Spain, October 9-10, 2008, Revised Selected*

Papers, volume 5491 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2008. doi:10.1007/978-3-642-01465-9_16.

[HLPT20] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *2020 IEEE Symposium on Security and Privacy*, pages 644–660, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press. doi:10.1109/SP40000.2020.00048.

[HMMV25] Tobias Hilt, Florian Moser, Philipp Matheis, and Melanie Volkamer. Development and expert evaluation of an informative video concerning verifiable internet voting. In *International Joint Conference on Electronic Voting*, pages 107–123. Springer Nature Switzerland Cham, 2025.

[HMVW16] Sven Heiberg, Tarvi Martens, Priit Vinkel, and Jan Willemson. Improving the verifiability of the Estonian Internet Voting scheme. In *International Joint Conference on Electronic Voting*, pages 92–107. Springer, 2016.

[HR25] Thomas Haines and Jarrod Rose. Vestigial vulnerabilities in deployed verifiable e-voting systems. *Cryptology ePrint Archive*, 2025.

[HSB21] Lucca Hirschi, Lara Schmid, and David A. Basin. Fixing the achilles heel of E-voting: The bulletin board. In Ralf Küsters and Dave Naumann, editors, *CSF 2021: IEEE 34th Computer Security Foundations Symposium*, pages 1–17, Virtual Conference, June 21–24, 2021. IEEE Computer Society Press. doi:10.1109/CSF51468.2021.00016.

[HW14] Sven Heiberg and Jan Willemson. Verifiable internet voting in estonia. In Robert Krimmer and Melanie Volkamer, editors, *6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014, Lochau / Bregenz, Austria, October 29-31, 2014*, pages 1–8. IEEE, 2014. doi:10.1109/EVOTE.2014.7001135.

[IT 23] IT Security Evaluation Facility of Deutsche Telekom Security GmbH. Protection profile for e-voting systems for non-political elections: BSI-CC-PP-0121 (ver-

sion 0.9). Technical report, Federal Office for Information Security Germany - BSI, 4 2023. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Schutzprofile/BSI-CC-PP-0121-Protection-Profile-for-E-Voting-Systems.pdf.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine, editors, *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*, pages 61–70. ACM, 2005. doi:10.1145/1102199.1102213.

[JFR13] Rui Joaquim, Paulo Ferreira, and Carlos Ribeiro. EVIV: An end-to-end verifiable Internet voting system. *Computers & Security*, 2013.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In Dan Boneh, editor, *USENIX Security 2002: 11th USENIX Security Symposium*, pages 339–353, San Francisco, CA, USA, August 5–9, 2002. USENIX Association. URL: http://www.usenix.org/publications/library/proceedings/sec02/jakobsson.html.

[JMP13] Hugo Jonker, Sjouke Mauw, and Jun Pang. Privacy and verifiability in voting systems: Methods, developments and trends. Cryptology ePrint Archive, Report 2013/615, 2013. URL: https://eprint.iacr.org/2013/615.

[JRF09] Rui Joaquim, Carlos Ribeiro, and Paulo Ferreira. Veryvote: A voter verifiable code voting system. In *International Conference on E-Voting and Identity (VOTE-ID)*, 2009.

[Kan] Kanton Zürich. National- und Ständeratswahlen. https://www.zh.ch/de/politik-staat/wahlen-abstimmungen/national-staenderatswahlen.html.

[Kan22] Kanton Zürich. Abstimmungsvorlagen vom 25. September 2022. https://www.zh.ch/de/news-uebersicht/medienmitteilungen/2022/06/abstimmungs

vorlagen-vom-25-september-2022.html, September 2022.

[KBB17] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *2017 IEEE European Symposium on Security and Privacy*, pages 435–450, Paris, France, April 26–28, 2017. IEEE Computer Society Press. doi:10.1109/EuroSP.2017.38.

[Ker24] Kercia. Solutions de vote électronique sécurisées, 2024. URL: https://www.kercia.com/.

[KKL+18] Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas Zacharias. On the security properties of e-voting bulletin boards. In Dario Catalano and Roberto De Prisco, editors, *SCN 18: 11th International Conference on Security in Communication Networks*, volume 11035 of *Lecture Notes in Computer Science*, pages 505–523, Amalfi, Italy, September 5–7, 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-319-98113-0_27.

[KLM+20] Ralf Küsters, Julian Liedtke, Johannes Müller, Daniel Rausch, and Andreas Vogt. Ordinos: A verifiable tally-hiding E-voting system. In *2020 IEEE European Symposium on Security and Privacy*, pages 216–235, Genoa, Italy, September 7–11, 2020. IEEE Computer Society Press. doi:10.1109/EuroSP48549.2020.00022.

[KMST16] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A lightweight verifiable remote voting system. In Michael Hicks and Boris Köpf, editors, *CSF 2016: IEEE 29th Computer Security Foundations Symposium*, pages 341–354, Lisbon, Portugal, June 27–1, 2016. IEEE Computer Society Press. doi:10.1109/CSF.2016.31.

[KNBV17] Oksana Kulyk, Stephan Neumann, Jurlind Budurushi, and Melanie Volkamer. Nothing comes for free: How much usability can you sacrifice for security? *IEEE Security & Privacy*, 15(3):24–29, 2017.

[KR05] Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi calculus. In *European*

Symposium on Programming (ESOP'05), pages 186–200. Springer, 2005.

[KRS10] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS 2010: 15th European Symposium on Research in Computer Security*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404, Athens, Greece, September 20–22, 2010. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-15497-3_24.

[KS19] Christian Killer and Burkhard Stiller. The Swiss postal voting process and its system and security analysis. In *International Joint Conference on Electronic Voting*, pages 134–149. Springer, 2019.

[KTV10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010: 17th Conference on Computer and Communications Security*, pages 526–535, Chicago, Illinois, USA, October 4–8, 2010. ACM Press. doi:10.1145/1866307.1866366.

[KTV12] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash attacks on the verifiability of e-voting systems. In *2012 IEEE Symposium on Security and Privacy*, pages 395–409, San Francisco, CA, USA, May 21–23, 2012. IEEE Computer Society Press. doi:10.1109/SP.2012.32.

[KTV15] Oksana Kulyk, Vanessa Teague, and Melanie Volkamer. Extending helios towards private eligibility verifiability. In Rolf Haenni, Reto E. Koenig, and Douglas Wikström, editors, *E-Voting and Identity - 5th International Conference, VoteID 2015, Bern, Switzerland, September 2-4, 2015, Proceedings*, volume 9269 of *Lecture Notes in Computer Science*, pages 57–73. Springer, 2015. doi:10.1007/978-3-319-22270-7_4.

[KWC+13] Tyler Kaczmarek, John Wittrock, Richard Carback, Alex Florescu, Jan Rubio, Noel Runyan, Poorvi L Vora, and Filip Zagórski. Dispute resolution in accessible voting systems: The design and use of audiotegrity. In *International Conference on E-Voting and Identity*, pages 127–141. Springer, 2013.

[KY02] Aggelos Kiayias and Moti Yung. Self-tallying elections and perfect ballot secrecy. In David Naccache and Pascal Paillier, editors, *PKC 2002: 5th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 141–158, Paris, France, February 12–14, 2002. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-45664-3_10.

[Leg24] LegaVote. Vous avez un projet de scrutin, nous avons une solution de vote électronique, 2024. URL: https://www.legavote.fr/.

[LHK19] Philipp Locher, Rolf Haenni, and Berner Fachhochschule Koenig, Reto E. Analysis of the Cryptographic Implementation of the Swiss Post Voting Protocol. report, Berner Fachhochschule, July 2019. Accessed at 2021-05-19. URL: https://www.bk.admin.ch/dam/bk/de/dokumente/pore/E-Voting_Report_Locher_Haenni_Koenig_Juli%202019.pdf.download.pdf/E-Voting_Report_Locher_Haenni_Koenig_Juli%202019.pdf.

[LKZ14] Huian Li, Abhishek Reddy Kankanala, and Xukai Zou. A taxonomy and comparison of remote voting schemes. In *23rd International Conference on Computer Communication and Networks (ICCCN 2014)*, pages 1–8. IEEE Computer Society, 2014. doi:10.1109/icccn.2014.6911807.

[LQT20] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. VoteAgain: A scalable coercion-resistant voting system. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020: 29th USENIX Security Symposium*, pages 1553–1570. USENIX Association, August 12–14, 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/lueks.

[Mar24] Markus Häfliger. Das ist ein Fiasko für unsere Demokratie. https://www.tagesanzeiger.ch/304329075925, Sep 2024.

[MGH+24] Florian Moser, Rüdiger Grimm, Tobias Hilt, Michael Kirsten, Christoph Niederbudde, and Melanie Volkamer. Recommendations for implementing independent

individual verifiability in internet voting. In *EVote-ID 2024-9th International Joint Conference on Electronic Voting*, 2024.

[MKD24] Florian Moser, Michael Kirsten, and Felix Dörre. Sok: Mechanisms used in practice for verifiable internet voting. In *E-Vote-ID 2024-9th International Joint Conference on Electronic Voting*, 2024.

[MKRV18] Karola Marky, Oksana Kulyk, Karen Renaud, and Melanie Volkamer. What did i really vote for? on the usability of verifiable e-voting schemes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2018.

[MMC+24] Florian Moser, Johannes Müller, Véronique Cortier, Alexandre Debant, Pierrick Gaudry, Anselme Goetschmann, Ralf Küsters, and Melanie Volkamer. A Study of Mechanisms for End-to-End Verifiable Online Voting. Technical report, Federal Office for Information Security, Germany, 2024. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Cryptography/End-to-End-Verifiable_Online-Voting.pdf.

[MN06] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 373–392, Santa Barbara, CA, USA, August 20–24, 2006. Springer Berlin Heidelberg, Germany. doi:10.1007/11818175_22.

[Moo20] Larry Moore. Voatz mobile voting platform – An overview: Security, identity, auditability (version 1.1). Technical report, Voatz, Inc., 2 2020. URL: https://web.archive.org/web/20240822073616/https://voatz.com/wp-content/uploads/2020/07/voatz-security-whitepaper.pdf.

[MPP24] Johannes Müller, Balázs Pejó, and Ivan Pryvalov. DeVoS: Deniable yet verifiable vote updating. *Proceedings on Privacy Enhancing Technologies*, 2024(1):357–378, January 2024. doi:10.56553/popets-2024-0021.

[MSCB13] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The Tamarin Prover for the Symbolic Analysis of Security Protocols. In *International Conference on Computer Aided Verification (CAV)*, 2013.

[MSLM19] Karola Marky, Martin Schmitz, Felix Lange, and Max Mühlhäuser. Usability of code voting modalities. In *Conference on Human Factors in Computing Systems, CHI 2019*. ACM, 2019.

[MT23] Johannes Müller and Tomasz Truderung. CAISED: A protocol for cast-as-intended verifiability with a second device. In *8th International Joint Conference on Electronic Voting (E-Vote-ID 2023)*, volume 14230 of *LNCS*, pages 123–139. Springer, 2023. doi:10.1007/978-3-031-43756-4_8.

[Mül22] Johannes Müller. Breaking and fixing vote privacy of the estonian E-voting protocol IVXV. In Shin'ichiro Matsuo, Lewis Gudgeon, Ariah Klages-Mundt, Daniel Perez Hernandez, Sam Werner, Thomas Haines, Aleksander Essex, Andrea Bracciali, and Massimiliano Sala, editors, *FC 2022 Workshops*, volume 13412 of *Lecture Notes in Computer Science*, pages 325–334, Grenada, May 6, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-32415-4_22.

[MZR+21] Karola Marky, Marie-Laure Zollinger, Peter Roenne, Peter YA Ryan, Tim Grube, and Kai Kunze. Investigating usability and user experience of individually verifiable internet voting schemes. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 28(5):1–36, 2021.

[Neo24] Neovote. Neovote, 2024. URL: https://www.neovote.com/.

[Neu24] Neuvote. Misc voting, 2024. URL: https://www.neuvote.com/.

[NKV+24] Christina Nissen, Oksana Kulyk, Melanie Volkamer, Lara Elisabeth Fredrich, and Helena Hermansen. Tracking code-based verification-design and evaluation. In *E-Vote-ID 2024*, pages 85–100. Gesellschaft für Informatik, 2024.

[NSW23] NSW Electoral Commission. Technology assisted voting - paper 3 final review report. https://elections.nsw.gov.au/getmedia/fa175615-1679-4ac0-85a1-ee0daac53d8b/tav-final-review-report.pdf, 2023.

[nVo24] nVotes. Secure misc voting software, 2024. URL: https://nvotes.com/.

[oE17] Council of Europe. Recommendation CM/Rec(2017)51 of the Committee of Ministers to member States on standards for e-voting, June 2017.

[OEAMiEOfDIR08] Organisation for Security OSCE/ODIHR Election Assessment Mission, Cooperation in Europe's Office for Democratic Institutions, and Human Rights. Swiss confederation federal elections: 21. october 2007. report, Organisation for Security and Cooperation in Europe's Office for Democratic Institutions and Human Rights, April 2008.

[ØK12] Bjarte M Østvold and Edvard K Karlsen. Public Review of E-Voting Source Code: Lessons learnt from E-Vote 2011. *Norsk informatikkonferanse*, 2012.

[Onl24] Onlz. Electronic voting you can trust., 2024. URL: https://www.onlz.com/.

[PCKO24] Seongho Park, Jaekyoung Choi, Jihye Kim, and Hyunok Oh. zkVoting: Zero-knowledge proof based coercion-resistant and E2E verifiable e-voting system. *IACR Cryptology ePrint Archive*, 2024. arXiv:2024/1003.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-46766-1_9.

[Per22] Olivier Pereira. Individual verifiability and revoting in the estonian internet voting system. In Shin'ichiro Matsuo, Lewis Gudgeon, Ariah Klages-Mundt, Daniel Perez Hernandez, Sam Werner, Thomas Haines, Aleksander Essex, Andrea Bracciali, and Massimiliano Sala, editors, *FC 2022 Workshops*, volume 13412

of *Lecture Notes in Computer Science*, pages 315–324, Grenada, May 6, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-32415-4_21.

[POL24] POLYAS. Polyas, 2024. URL: https://www.polyas.de/.

[Pos19] Schweizerische Post. Die Post setzt ausschliesslich auf das neue System mit universeller Verifizierbarkeit, July 2019. Accessed at 2021-05-21. URL: https://www.post.ch/de/ueber-uns/medien/medienmitteilungen/2019/die-post-setzt-ausschliesslich-auf-das-neue-system-mit-universeller-verifizierbarkeit.

[Pro25a] ProVerif. Open VSX Registry: ProVerif Language. https://open-vsx.org/extension/ProVerif/vscode-proverif, Dez 2025.

[Pro25b] ProVerif. Visual Studio Code Marketplace: ProVerif Language. https://marketplace.visualstudio.com/items?itemName=ProVerif.vscode-proverif, Dez 2025.

[PT19] Olivier Pereira and Vanessa Teague. Report on the SwissPost-Scytl e-voting system, trusted-server version. report, Pereira, Olivier and Teague, Vanessa, July 2019. Accessed at 2021-05-19. URL: https://www.bk.admin.ch/dam/bk/de/dokumente/pore/E-Voting_Report_Pereira_Teague_Juli%202019.pdf.download.pdf/E-Voting_Report_Pereira_Teague_Juli%202019.pdf.

[Rob22] Eduardo Robles. Sequent voting systemarchitecture overview (work in progress). Technical report, Sequentech, 4 2022. URL: https://web.archive.org/web/20240830084927/https://sequentech.github.io/documentation/assets/files/2022-04-10-arch-1-5bc31f49172dc628367719c7785abe36.pdf.

[RRI16] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 176–192, Christ Church, Barbados, February 26, 2016. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-53357-4_12.

[RT09] Peter YA Ryan and Vanessa Teague. Pretty good democracy. In *International Workshop on Security Protocols*, 2009.

[Rue21] David Ruescas. Sequent tech cryptographic protocol. Technical report, Sequentech, 3 2021. URL: https://web.archive.org/web/20240830085125/https://sequentech.github.io/documentation/assets/files/2021-03-19-proto-1-18fd36c9669813aadcf59e672f0f7a84.pdf.

[Scy24] Scytl. Secure misc voting, 2024. URL: https://scytl.com/.

[Seq22] Sequent. Bringing Sequent's End-To-End Verifiable Voting Solution to the German Market. https://sequentech.io/case-study/bringing-sequents-end-to-end-verifiable-voting-solution-to-the-german-market/, 2022.

[Seq24a] Sequent. Open source end-to-end verifiable misc voting, 2024. URL: https://sequentech.io/.

[Seq24b] Sequent. Sequent, 2024. URL: https://github.com/sequentech/.

[Seq24c] Sequent. Sequentech docs, 2024. URL: https://sequentech.github.io/documentation.

[SFD+14] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. Security analysis of the estonian internet voting system. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 703–715, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. doi:10.1145/2660267.2660315.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press. doi:10.1109/SFCS.1994.365700.

[SHR23] Anggrio Sutopo, Thomas Haines, and Peter B. Rønne. On the auditability of the estonian IVXV system - and

an attack on individual verifiability. In Aleksander Essex, Shin'ichiro Matsuo, Oksana Kulyk, Lewis Gudgeon, Ariah Klages-Mundt, Daniel Perez, Sam Werner, Andrea Bracciali, and Geoff Goodell, editors, *FC 2023 Workshops*, volume 13953 of *Lecture Notes in Computer Science*, pages 19–33, Bol, Brač, Croatia, May 5, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-48806-1_2.

[Sim24a] Simply Voting. Checking, interpreting, and publishing results, 2024. URL: https://web.archive.org/web/20240711091037/https://help.simplyvoting.com/docs/checking-interpreting-and-publishing-results.

[Sim24b] Simply Voting. Simply voting, 2024. URL: https://www.simplyvoting.com/.

[SJT08] Karen Scarfone, Wayne Jansen, and Miles Tracy. Guide to general server security. Technical Report Special Publication 800-123, National Institute of Standards and Technology (NIST), 2008. doi:10.6028/nist.sp.800-123.

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology – EUROCRYPT'95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403, Saint-Malo, France, May 21–25, 1995. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-49264-X_32.

[SRF21] SRF Schweizer Radio und Fernsehen. Frauenfelder Ex-Stadtschreiber schuldig gesprochen. https://www.srf.ch/news/schweiz/verurteilt-wegen-wahlfaelschung-frauenfelder-ex-stadtschreiber-schuldig-gesprochen, 2021.

[SRKK10] Ben Smyth, Mark Ryan, Steve Kremer, and Mounira Kourjieh. Towards automatic analysis of election verifiability properties. In *Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*, pages 146–163. Springer, 2010.

[Sta22] Stadt Zürich. 25. September 2022: fünf kommunale Sachvorlagen . https://www.stadt-zuerich.ch/portal/de/index/politik_u_recht/abstimmungen_u_wahlen/archiv_abstimmungen/vergangene_termine/220925.html, September 2022.

[Ste23] Steuerungsausschuss Vote électronique. Vote électronique - Massnahmenkatalog von Bund und Kantonen. https://www.bk.admin.ch/dam/bk/de/dokumente/pore/Vote--lectronique/Vote%20%C3%A9lectronique_%20Massnahmenkatalog%20von%20Bund%20und%20Kantonen%20vom%2004.08.2023.pdf.download.pdf/Vote%20%C3%A9lectronique_%20Massnahmenkatalog%20von%20Bund%20und%20Kantonen%20vom%2004.08.2023.pdf, 2023.

[SW24] Daniel Stockemer and Michael Wigginton. The (complex) effect of internet voting on turnout: Theoretical and methodological considerations. *Policy & Internet*, 16(3):607–627, 2024.

[Swi13] Swiss Federal Chancellery. Federal Chancellery Ordinance on Electronic Voting. ordinance, Swiss Federal Chancellery, 12 2013. URL: https://www.fedlex.admin.ch/eli/cc/2022/336/en.

[Swi22] Swiss Post. Swiss Post Voting System: System Specification. Version 1.1.1. Technical report, Swiss Post, October 2022. URL: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/afbd7555b3f87092596a5203f022c00dcc0fd390/System/System_Specification.pdf.

[Swi24a] Swiss Federal Chancellery. Examination of systems, 2 2024. URL: https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting/ueberpruefung_systeme.html.

[Swi24b] Swiss Post. E-voting: Misc voting and elections, 2024. URL: https://digital-solutions.post.ch/en/e-government/digitization-solutions/e-voting.

[Swi24c] Swiss Post. Swiss post voting system – system specification (version 1.4.1). Technical report, Swiss Post, 6 2024. URL: https://gitlab.com/swisspost-evoting

/e-voting/e-voting-documentation/-/blob/a3e1063615e2efcdf2692e8a47697daf7ccdb2d1/System/System_Specification.pdf.

[Swi24d] Swiss Post. Swiss post voting system verifier specification (version 1.5.2). Technical report, Swiss Post, 6 2024. URL: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/a3e1063615e2efcdf2692e8a47697daf7ccdb2d1/System/Verifier_Specification.pdf.

[Swi24e] Swiss Post. swisspost-evoting, 2024. URL: https://gitlab.com/swisspost-evoting.

[Swi25a] Swiss Post. Cryptographic primitives of the swiss post voting system (version 1.5.1). Technical report, Swiss Post, 8 2025. URL: https://gitlab.com/swisspost-evoting/crypto-primitives/crypto-primitives/-/blob/b4ac0ca02765929a5847a96ad57893ba35b02ade/Crypto-Primitives-Specification.pdf.

[Swi25b] Swiss Post. Infrastructure whitepaper of the swiss post voting system. Technical report, Swiss Post, 11 2025. URL: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/509ffba0cfc609bd952587521379b380a9957fba/Operations/Infrastructure%20whitepaper%20of%20the%20Swiss%20Post%20voting%20system.md.

[Swi25c] Swiss Post. Protocol of the Swiss Post Voting System: Computational Proof of Complete Verifiability and Privacy. Version 1.4.0. Technical report, Swiss Post, June 2025. URL: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/blob/b7eef7459787dfcddb6559fe151ec8ecf13ecd1e/Protocol/Swiss_Post_Voting_Protocol_Computational_proof.pdf.

[Swi25d] Swiss Post. Swiss Post Voting System: Symbolic Analysis of the Swiss Post Voting System. Version 1.4.0. Technical report, Swiss Post, 2025. URL: https://gitlab.com/swisspost-evoting/e-voting/e-voting-documentation/-/tree/master/Symbolic-models?ref_type=heads.

[Swi25e] Swiss Post. YesWeHack: Swiss Post - E-Voting. https://yeswehack.com/programs/swiss-post-evoting, Dez 2025.

[Tru23] Tomasz Truderung. POLYAS 3.0 verifiable e-voting system (version 1.3.2). Technical report, POLYAS GmbH, 7 2023. URL: https://github.com/polyas-voting/core3-verifiable-doc/blob/d23fdb8d2627e17b181e06f97fb44e38865bf157/pdf/polyas3.0-verifiable.pdf.

[Tru24] Tomasz Truderung. POLYAS-core3 second device protocol (version 1.1). Technical report, POLYAS GmbH, 2 2024. URL: https://github.com/polyas-voting/core3-verifiable-doc/blob/432434bb85ee24e5aa9caaacf3e5e24bc6d50708/pdf/second-device-spec.pdf.

[TVK22] Paul Tim Thürwächter, Melanie Volkamer, and Oksana Kulyk. Individual verifiability with return codes: manipulation detection efficacy. In *International Joint Conference on Electronic Voting*, pages 139–156. Springer, 2022.

[TW10] Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 10: 3rd International Conference on Cryptology in Africa*, volume 6055 of *Lecture Notes in Computer Science*, pages 100–113, Stellenbosch, South Africa, May 3–6, 2010. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-12678-9_7.

[V8T24] V8TE. Misc voting platform 100% self-service, 2024. URL: https://www.v8te.com/.

[Val22] Valimised.ee. IVXV protocols: Specification (version 1.8.0). Technical report, Valimised.ee, 12 2022. URL: https://web.archive.org/web/20240623062715/https://www.valimised.ee/sites/default/files/2023-02/IVXV-protocols.pdf.

[Val24] Valimised Eestis. IVXV, 2024. URL: https://github.com/valimised/ivxv/.

[VKLF22] Melanie Volkamer, Oksana Kulyk, Jonas Ludwig, and Niklas Fuhrberg. Increasing security without decreasing usability: A comparison of various verifiable voting

systems. In *Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022)*, pages 233–252, 2022.

[Voa24] Voatz. Voatz secure and convenient voting anywhere, 2024. URL: https://voatz.com/.

[Vox24] Voxaly. Vote électronique - Voxaly vous accompagne, 2024. URL: https://www.voxaly.com/vote/electronique/.

[VT19] The University of Melbourne Vanessa Teague. Faking an iVote decryption proof: Why the decryption proof flaw identified in the Swiss Post system affects the iVote system too. report, The University of Melbourne, November 2019.

[VV08] Melanie Volkamer and Roland Vogt. Common criteria protection profile for basic set of security requirements for misc voting products: BSI-PP-0037 (version 1.0). Technical report, Federal Office for Information Security Germany - BSI, 4 2008. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Reporte/ReportePP/pp0037b_engl_pdf.pdf.

[WeC24] WeChooz. Vote électronique - Solutions de vote par internet sécurisées - WeChooz, 2024. URL: https://www.wechooz.fr/solution-de-vote-electronique/.

[Wik04] Douglas Wikström. A universally composable mix-net. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 317–335, Cambridge, MA, USA, February 19–21, 2004. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-24638-1_18.

[Wik24] Douglas Wikström. Open verificatum, 2024. URL: https://www.verificatum.org/.