Olivier Pereira
UCLouvain

December 6, 2022

1 Introduction

The problem of voting may be seen as the most elementary secure function evaluation problem. Each voter $V_i \in \{V_1, \ldots, V_n\}$ has a secret input x_i, which is 0 or 1 depending on the voter intent, and the voters want to jointly compute the election result as $\rho(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$, while not revealing any information that cannot be derived from this election result.

Low-tech solutions can be used to solve this problem. One traditional protocol goes as follows:

1. Election authorities define who has the right to vote, and on which question the voters should voter;
2. Voters enter a voting booth that gives them privacy while they write their vote on a ballot, which they fold before leaving the booth;
3. Voters gather around a ballot box and check that it is empty;
4. Voters, in-turn, place their ballot in the ballot box;
5. The ballot box is shaken, then opened, and the votes are publicly counted;
6. The election authorities announce the tally.

This protocol has many benefits.

Correctness. Regarding correctness, this protocol has the following appealing properties:

- **Cast-as-Intended** voters can verify that their ballot captures their vote intent;
- **Recorded-as-Cast** voters can verify that their ballot is on-record in the ballot box, and is not removed or tampered with;
- **Eligibility Verifiability** anyone can verify that only legitimate voters insert one ballot, and only one ballot, in the ballot box;

- **Counted-as-Recorded** anyone can verify that the ballots contained in the ballot box are proprely counted.

The first two properties are often associated to the idea of *individual verifiability*, while the last two properties are often associated to the idea of *universal verifiability*. A voting system that offers all these properties is called *end-to-end verifiable*, or simply *verifiable*, following an idea introduced by Benaloh and Fischer in 1985 [CF85]. In particular, a successful verification process of a verifiable election guarantees that the tally is correct independently of any trust assumption in any (subset of a) designated group of persons (election authorities, observers, auditors, ...), or of any trust assumption in any piece of software. In many verifiable voting systems, the computational complexity of the verification process requires using a computer and a verification software. This does not create a contradiction with the previous trust model, as long as the verification does not depend on trusting any specific (subset of a) designated group of computers or pieces of software: the auditor is free to pick any computing device, or possibly several computing devices, and to use his own verification software if he wishes to do so. Phrased in another way, the use of software in an election would not contradict end-to-end verifiability as long as the election remains *software independent*, meaning that "an undetected change or error in its software cannot cause an undetectable change or error in an election outcome" [Riv08].

Privacy. Regarding privacy, this protocol keeps individual votes completely secret until the tallying starts, guarantees that votes are cast independently of each other, and eventually leaks nothing more about the votes than what can be derived from the election result. What is actually leaked by the election result then depends on the voters' choices: an unanimous choice would leak everyone's vote intent. It also depends on the correctness of the ballot box content: if someone manages to remove all the ballots from the ballot box before the last ballot is inserted, and if that last ballot becomes the only one to be tallied, then that last vote won't be secret, even if a large number of ballots supporting the different choices were initially inserted in the ballot box.

The use of a voting booth is also expected to offer some form of receipt freeness or coercion resistance, that is, to make it impossible for a voter to demonstrate how he voted, even if he wishes to do so. This guarantees the free expression of the vote, as a voter may not be coerced into voting in a specific way, nor willingly decide to sell his vote, since the buyer would not be able to verify what he paid for. Unless, again, individual votes can be derived from the election result.

However, the extent to which this protocol guarantees coercion resistance may be limited in reality, and depend on many extra security measures. When voters write the name of their selected candidate on the ballot, they may be given carbon paper to record how they voted [CML18]. Or voters may be given a stolen pre-filled ballot before entering the voting booth, and be required to cast that ballot and bring back a blank ballot, which will be used to coerce

another voter [Cru13]. Smartphones or cheap spy cameras can also be used to ask a voter to film himself inside the voting booth, in order to offer live evidence of the content of a ballot [New03]. When talliers are also coercers or vote buyers, they may also require voters to place a small mark on a ballot, or to make a pin hole at a specific position, or to fold the ballot in a specific way, so that it can be tracked [Sch02]. When there are many ways of voting, e.g., when it is possible to approve any number of candidates from a long list, or when a large number of candidates can be ranked, a voter may simply be instructed to vote according to a most probably unique pattern [DC07]. In such a case, even electronic cast vote records can be used for coercion, and coercion could then be performed by anyone if these cast vote records are made public (e.g., in order to support recounts).

The chain of custody. The security of this protocol crucially depends on the quality of the chain of custody of the ballots. In particular, universal verifiability depends on the possibility for a voter, or for an observer, to continuously watch the ballot box in order to verify who inserts a ballot, that the ballot box is not tampered with, and that the counting process is honest. If this continuous watch is lost, the chain of custody is broken, and it cannot be repaired. In particular, any recount is useless if there is no trust that the right ballots are counted. And maintaing an unbroken chain of custody on the ballot boxes can become extremely challenging as soon as an election must be organized at a moderately large scale.

The reliance on a chain of custody has become necessary because of the requirement of secret ballots. If there were no privacy requirement, it would be an option to have a public bulletin board on which one could display the list of voters together with the ballot that they submitted. Verifying the content of this bulletin board could be done by anyone, at any time, and any change would be visible. A good custody would of course still be useful in order to make sure that the election process succeeds (anyone erasing the bulletin board might be able to cause the cancellation of the election), but the role of this custody is not anymore to guarantee that the announced result is correct.

Cryptography comes in. These observations highlight why secret ballot elections raise a fundamental tension between correctness and privacy requirements. As these requirements are the core business of cryptography, it is quite appealing to explore how cryptography could reconcile them.

Of course, introducing cryptographic operation in an election also creates new difficulties: introducing the requirement of a cryptographic expertise in order to understand that their vote was properly recorded and counted may contradict the requirement of organizing public elections: similar arguments were used in the March 3, 2009 judgement by the German Federal Constitutional Court on the use of voting computers [Fed09]. Other democratic countries have a different view, and Internet voting making heavy use of cryptographic protocols has been routinely used in government elections in Estonia and Switzerland, for

instance, during the last 20 years.

But the adoption of cryptography in an election process is not a synonym of a reliance on cryptography for a correct election result. Cryptographic techniques have indeed been designed and deployed, including in government elections, with the purpose of offering an additional layer of verifiability rather than as a substitution to traditional security measures [CCC+10, CRST15, BFS+21].

Chapter organization In the rest of this chapter, we will describe various cryptographic techniques that have been proposed in support of verifiable secret ballot elections.

We will start by presenting a simple voting system, then describe security definitions for verifiability and ballot privacy, and eventually discuss two problems for which we remain fairly far from having satisfactory solutions.

2 Single pass voting

Most voting schemes follow a common pattern of a single pass voting process [BCG+15], running in four phases, based on six algorithms: Setup, Vote, Tally, $\mathsf{Publish}$, $\mathsf{VerifyBallot}$, $\mathsf{VerifyTally}$.

1. In a **setup** phase, a well identified authority, possibly supported by a group of trustees, runs a Setup protocol that generates the data needed to run the election, based on a security parameter λ and an election description ED, which includes the questions and choices that are offered on the ballot, the list of voters, the voting period and the place at which ballots must be submitted. $\mathsf{Setup}(1^\lambda, ED)$ outputs a pair (pp, sk) where the public parameter pp contains ED augmented with cryptographic material such as public keys and the description of the trustees. The secret elements in sk are kept by the authority and trustees – each trustee T_i may keep his own secret element sk_i, part of sk. We assume that pp is provided as input to all the other algorithms.

2. In a **voting** phase, voters access the public paramters pp from the setup phase, and produce a ballot b encoding their voting choice v, together with a tracker t for that ballot by running $(b, t) \leftarrow \mathsf{Vote}(id, v)$. The pair (id, b) is then submitted to the authority, possibly with some credentials, while the tracker t is kept by the voter for auditing purpose. The authority keeps the sequence (id, b) of submitted ballots in an internal bulletin board BB.

3. In a **tallying** phase, the authority and the trustees jointly compute the election result v, and evidence π of the correctness of this result, by jointly computing $(v, \pi) \leftarrow \mathsf{Tally}(sk, BB)$. They publish a public version of BB as $PBB \leftarrow \mathsf{Publish}(BB)$ together with v and π.

4. In the **audit** phase, which may in practice be split into different steps that are performed in parallel of, or in-between the previous three election phases, based on (PBB, v, π). Trustees, when they exist, are invited

to verify that their secret key elements have been properly included in the election public parameters by computing $\mathsf{VerifySetup}(T_i, sk_i)$, which returns $\top$ or $\bot$ depending on whether the verification is successful. Voters themselves are invited to verify that their ballot made it to PBB by computing $\mathsf{VerifyBallot}(PBB, id, t)$, which returns $\top$ or $\bot$ as before. Furthermore, anyone, including external auditors, is invited to verify that the announced result v is consistent with PBB by computing $\mathsf{VerifyTally}(PBB, v, \pi)$, which also returns $\top$ or $\bot$.

This single pass process, in which voters only need receive data from the setup phase information, and then cast their ballot by submitting a single message (that may include paper and electronic data), offers a lot of convenience. It is rarely realistic, except in smaller scale elections like board elections, to require voters to communicate with each other or interact in multiple rounds in order to submit their ballot, even though this may also offer security benefits, like the absence of any trustees on which privacy would rest, or dispute freeness [KY02, HRZ10].

2.1 Voting based on an additively homomorphic encryption scheme

We describe one of the most common and successful approaches to verifiable single pass voting schemes was pioneered by Benaloh [Ben87]. In the setup phase, trustees jointly generate an election public key for an additively homomorphic encryption scheme.

In the voting phase, voters use that public key to encrypt their vote, expressed as an encryption of a 0 or a 1 for each candidate, prove in non-interactive zero-knowledge that their encrypted vote is valid, and submit the resulting ballot, made of the encrypted vote and validity proofs, to the election authority. Voters also keep a hash of their ballot as tracker.

In the tallying phase, the trustees compute an encryption of the election result by taking advantage of the homomorphic property of the encryption scheme, and verifiably decrypt that election result.

In the audit phase, the election authority publishes the list of voter id and ballot pairs, together with the election result decryption proof. Voters can then use the ballot hash that they recorded in order to see that the right ballot is associated to their id, or that there is no ballot at all if they did not vote. Eventually, everyone can check that all the ballots are associated to legitimate voters, that all the recorded ballots are valid, and that the announced election result is indeed the decryption of the homomorphic aggregation of all the recorded ballots.

Benaloh's protocol was based on his own encryption scheme, which relies on a higher-order residue class decision problem modulo an RSA modulus. Here, we present a different instance of Benaloh's approach, similar to the protocol introduced by Cramer, Gennaro and Schoenmakers [CGS97], and which works in any DDH group. Close variants of this protocol have been adopted in numerous

voting systems and libraries, including VoteBox [SDW08], Helios [AdMPQ09], Belenios [CGG19] and ElectionGuard [Ben22]. The protocol proceeds as follows.

1. **Setup.** Based on the election description ED, the election authority runs the protocol $\mathsf{Setup}(1^\lambda, ED)$. It proceeds by (i) designating a set of trustees $\{T_1, \ldots, T_m\}$, (ii) selecting a DDH group $(\mathbb{G}, g, q)$, where g generates the group $\mathbb{G}$ of prime order q (iii) asking each trustee T_i to pick a secret key $x_i \leftarrow \mathbb{Z}_q$ and return the corresponding public key $y_i = g^{x_i}$, together with a Schnorr proof of knowledge of x_i, computed as (e_i, f_i) where $e_i = H(\ell_i, y_i, g^{r_i})$ for a random $r_i \in \mathbb{Z}_q$ and $f_i = r_i + e_i x_i$. Here, H is a hash function, and ℓ_i designates context information, including T_i, ED, and the group description $(\mathbb{G}, g, q)$. The election public parameters pp are made of ED, the DDH group $(\mathbb{G}, g, q)$, the list of trustees and associated public keys and proofs (T_i, y_i, e_i, f_i), and the election public key computed as $(g, y = \prod_{i=1}^m y_i = g^{\sum_{i=1}^m x_i})$. They are made public by the election authority. The secret elements in sk are the x_i decryption keys, which each trustee T_i keeps secret, together with ℓ_i.

If the identity of the trustees submitting their public key is properly verified, and if the proofs are all valid, such a key generation process guarantees that the contribution of all the trustees will be necessary to decrypt a message freshly encrypted with the ElGamal public key (g, y). However, a single missing key would make it impossible to compute an election result, and robustness is often desired. Sometimes, this concern can be addressed in very simple ways: if one wishes to have 3 trustees, and 2 of them needed to tally an election, it is sufficient to ask each trustee T_i to share a copy of their secret key with trustee $T_{i+1 \bmod 3}$ before the election starts. General threshold solutions can also be obtained using variants of a protocol proposed by Pedersen [Ped91] that have been implemented in Belenios [CGG19] and ElectionGuard [Ben22] for instance. They however bring the practical difficulty of requiring rounds of communication between the trustees.

2. **Voting.** Each voter obtains pp from the the authority, including the ballot description and election public key (g, y). Now, Vote computes, for a ballot with a single option $v \in \{0, 1\}$, an ElGamal encryption of the voting choice together with a proof that the ciphertext is indeed an encryption of a bit. An encryption of choice v is computed as $(a, b) = (g^r, y^{r+v})$ for $r \leftarrow \mathbb{Z}_q$, and the proof as (e, f_r, f_a, f_b) where $e = H(\ell, a, b, a', b', c, d)$ with $a' = g^s, b' = y^{w+s}, c = g^t, d = y^{t+vw}$ for $s, t, w \leftarrow \mathbb{Z}_q$ and $f_r = s + er$, $f_a = w + ev$ and $f_b = t + r(e - f_a)$. As before, ℓ contains context information, including pp. The ciphertext and proof are serialized into a ballot b and submitted to the authority, together with the voter identity. The tracker t is computed as $H(b)$.

The Vote algorithm performs encryption using exponential ElGamal, which is by far the most commonly used encryption mechanism in verifiable election systems, because its key generation process can be distributed easily [Ped91].

The situation is indeed considerably more complicated with an encryption scheme based on an RSA group, like Paillier's scheme for instance [Pai99], because of the complexity of generating an RSA modulus in a distributed way. Exponential ElGamal comes with an inefficient decryption process since a discrete logarithm must be extracted, but this discrete logarithm is normally small enough to be of no practical concern: exhaustive tables are often pre-computed, or a baby-step giant-step algorithm can be used if needed.

The proof that the ElGamal ciphertext is an encryption of $v \in \{0, 1\}$ actually shows that $v(1-v) = 0 \bmod q$, and is analyzed by Groth and Kohlweiss [GK15] for instance. It offers the convenience of being compatible with several batching techniques, saving computational work when many proofs need to be computed [DPP22a]. Another option, used in many practical systems, is based on the Chaum and Pedersen discrete logarithm equality proof [CP92], made disjunctive thanks to the generic transformation of Cramer et al. [CDS94].

Such a Vote algorithm is tailored for a single choice: the encryption hides the choice, and the proof guarantees to a third party that the ciphertext contains a valid choice. Approval voting on multiple candidates can be performed trivially, by computing as many ciphertexts and proofs as the number of candidates, and all these ciphertexts and proofs can be serialized in a single ballot. Different validity proofs may also be included in elections where the voter can only approve a limited number of candidates [CGS97, DJ01].

It is obviously crucial to make sure that the voter obtains authentic election public parameters: using an incorrect ballot format, ballot submission process or election public key would have a disastrous effect. Ensuring the use of authentic parameters may be highly non-trivial to address in practice, depending on the election setting. In the case of Internet voting, the public parameters may be delivered from a web server managed by the authority. One then needs to make sure that the voters connect to the correct web server, but also that the web server distributes authentic parameters. Signing these parameters offers limited help here, because voters would typically need access to another root of trust in order to obtain access to authentic hashes or signature verification keys – this may only be feasible to expert users. And even if an authentic signature verification key may be of limited interest if the signature verification software comes from the same potentially compromised web server. Potential ways to address these issues include the use of independent ballot preparation systems rather than relying on code delivered by the server and the publication of (a hash of) the public parameters through independent channels – newspapers for instance. Note that these challenges are not unique to Internet voting: they sometimes appear in paper-based elections as well [TM20].

3. **Tallying** Once the voting phase is over, and after the content of the bulletin board has been frozen, the election result can be computed using the Tally protocol.

   The simplest option, used by default in most of the systems referenced above [SDW08, AdMPQ09, CGG19, Ben22], is to take advantage of the additive homomorphism offered by ElGamal ciphertexts: the ciphertexts

$(g^{r_1}, y^{r_1+v_1})$, ..., $(g^{r_n}, y^{r_n+v_n})$ included in each ballot can be multiplied together, resulting in the ciphertext $(a, b) = (g^r, y^{r+v}) = (g^{\sum_{i=1}^{n} r_i}, y^{\sum_{i=1}^{n} r_i+v_i})$, which encrypts the election result. Each trustee T_i can then compute the decryption factor $d_i = a^{x_i}$, and prove that this factor was correctly computed, using the Chaum-Pedersen protocol: the proof (e_i, f_i) is computed as $e_i = H(\ell_i, y_i, a, b, g^{s_i}, a^{s_i})$ and $f_i = s_i + e_i x_i$. Here, ℓ_i designates context information, including T_i and the hash of BB and $s_i \in \mathbb{Z}_q$ is selected uniformly at random.

When all the decryption factors have been collected and published, it is possible to compute $y^v = b / \prod_{i=1}^{m} d_i$. Now, the election result v can be found by extracting the discrete logarithm of y^v in base y. This discrete logarithm is upper-bounded by the number of voters.

The trustees then post $BB = \mathsf{Publish}(BB)$ (so, $\mathsf{Publish}$ is the identity function here), the election result v and a proof of the validity of this result made of all the decryption factors d_i and associated decryption proofs (e_i, f_i).

The tallying method proposed here is extremely simple and efficient for the trustees: they need to perform one single ElGamal decryption and to compute one Chaum-Pedersen proof per candidate on the ballot, with a total cost close to the one of 3 exponentiations. In particular, provided that the encrypted election result is provided to the trustees, the computational load of each trustee is independent of the number of voters. The main limitation of this approach is that it may become fairly challenging to use with large or complex ballots: it requires one ciphertext and validity proof per candidate on the ballot, and can hardly be adapted to ballots that use ranked choices or other richer choice expression rules.

Tallying approaches based on mix-nets [Cha81, SK95] can offer the level of flexibility that is needed. Here, encrypted votes are verifiably re-encrypted and shuffled by the trustees before they are individually decrypted. The main convenience of this approach is that the validity of a ballot can be verified in clear, but this time only after decryption (even though ciphertext independence may need to be verified early [Wik08]). As a result, there is no need to provide a proof of the validity of a ballot at submission time, at least when it is acceptable to verify the validity of the ballots after decryption. This is of course paid by a more cumbersome tallying process: the computational load of the trustees now grows linearly with the number of voters, and a few exponentiations will typically be needed per ciphertext that needs to be mixed (rather than a few exponentiations per candidate on the ballot). In terms of process, a mix-net is also an inherently sequential process that requires more coordination, since each trustee needs to mix the outputs provided by the previous trustee. In practice however, most of the computation can be performed in parallel.

One commonly deployed mix-net is the one of Wikström [Wik, Wik09], which has been used, among others, in Helios [BGP11], Wombat [BFL+12], and also in municipal elections in Norway [Gjo] and national elections in Es-

tonia [HMVW16] for instance. The other common solution is randomized partial checking as proposed by Jakobsson et al. [JJR02], which has been used in Scantegrity for municipal elections in the US [CCC+10], and in vVote for state elections in Australia [CRST15] for instance.

One downside to using a mix-net is that the decrypted ballots, which are now used to tally the election, may leak too much information about individual choices, which in turn could be used for coercing voters: a coercer may threaten to harm a voter is a ballot containing a unique pattern of choices does not show up. This threat can be handled using more advanced techniques: see Benaloh et al. [BMN+09] and the Ordinos protocol [KLM+20] for instance.

4 **Audit.** There are three audit steps, performed by different parties: trustees run VerifySetup, voters run VerifyBallot and anyone can run VerifyTally algorithms.

(a) In VerifySetup, each trustee T_i checks the presence of a (T_i, y_i, e_i, f_i) tuple in pp and outputs $\top$ only if $y_i = g^{x_i}$ – $(\mathbb{G}, g, q)$ was saved during Setup.

(b) In VerifyBallot, a voter with identifier id who voted and kept a tracker t verifies the presence of a (id, b) pair on PBB and outputs $\top$ only if $t = H(b)$. Registered voters who did not vote verify that there is no ballot associated to their id on PBB.

(c) In VerifyTally, anyone performs a number of global verification steps.

i. The DDH group $(\mathbb{G}, g, q)$ provided by the election authority offers the desired security level.

ii. The (T_i, y_i, e_i, f_i) tuples offered for each trustee in pp are such that $e_i = H(\ell_i, y_i, g^{f_i - e_i x_i})$ and $y = \prod_{i=1}^{m} y_i$.

iii. The ballot style in the election description includes a single choice.

iv. The VerifySetup and VerifyBallot steps have been performed by the trustees and listed voters respectively.

v. All the id's listed on PBB with a registered ballot are on the list of voters from the election description.

vi. Every recorded ballot (a, b, e, f_r, f_a, f_b) encodes a valid vote, by checking that $(a, b) \in \mathbb{G}^2$, and that e can be recomputed, that is: $e = H(\ell, a, b, g^{f_r} a^{-e}, y^{f_r + f_a} b^{-e}, g^{f_b} a^{f_a - e}, h^{f_b} b^{f_a - e})$.

vii. Every recorded ballot is independent of the others, by checking that each e value contained in a ballot is unique across all recorded ballots.

viii. Each decryption factor d_i is correct, by recomputing the encrypted election result (a, b) as in the Tally function, and verifying the each Chaum-Pedersen proof (e_i, f_i) is such that $e_i = H(\ell_i, y_i, a, b, g^{f_i} y_i^{-e_i}, a^{f_i} d_i^{-e})$.

ix. The announced election result v is consistent with the encrypted election result and decryption factors, that is, $y^v = b / \prod_{i=1}^{m} d_i$.

We see that VerifySetup, VerifyBallot and VerifyTally have three different roles.

The first two algorithms aim at making sure that the contribution of various parties to the protocol are properly taken into account. In VerifySetup, the trustees make sure that their public key share has been properly recorded on their behalf, and was not substituted with a public key whose private counterpart would be known by someone else. This is needed for voters to be able to trust that the vote that they encrypt is indeed protected by the whole group of trustees. In VerifyBallot, the voters make sure that their ballot has been properly recorded on their behalf, or that no ballot was recorded for them if they did not vote. Unless trust is delegated to third parties, trustees and voters are the only ones who can perform these steps. A level of indirection can be obtained by asking trustees to sign their public key, and voters to sign their ballot, but one then needs to trust the mechanism by which the signature verification keys are authenticated (which could be a PKI authority, for instance).

VerifyTally focuses on the steps that are needed to ensure that the result announced by the election authority is correct, and privacy is maintained. *(i.)* The use of a secure DDH group is needed both for correctness and privacy. For instance, a small group could make it easy to decrypt every ballot posted on the bulletin board, and also make it easy to forge ZK proofs, since they only offer computational soundness. *(ii.)* It is necessary to make sure that each trustee knows its own private key: the last trustee could pick $y = g^{x_m}$ and set $y_m = y / \prod_{i=1}^{m-1} y_i$ otherwise, making it possible for him to decrypt all the ballots alone. The election public key must also be built form each of the trustees individual public keys – the authority could simply bypass all the trustees otherwise. *(iii.)* The cryptographic format of the ballot matches the election description: voters were not offered to vote for more or fewer questions than expected. In case of a more complex election, this may require checking the order of the questions, etc. *(iv.)* The trustees validated that their key contribution has been taken included in the election public key, and the voters confirmed that their ballot is on the bulletin board when they voted, or that there is no recorded ballot for them if they did not vote. This step may require interactions with the trustees and the voters, or possibly a trust delegation mechanism if some voters want to accept one. *(v.)* All the ballots included in the tally are associated to a voter. *(vi.)* All the ballots included in the tally encrypt a valid choice. For instance, no voter encoded multiple votes in a single ballot. *(vii.)* No voter copied a ballot from another voter. This is important when the independence of the choices is desired, which may also have an impact on privacy [CGMA85, PP89]. *(viii.)* and *(ix.)* The tally is computed correctly from the set of ballots displayed on the bulletin board.

There are several observations that need to be made about this verification process.

First, we can see how importantly these verification mechanisms rely on everyone being aware of the election description, and the possibility to access a single public bulletin board. If this is not the case, then a honest voter may be tricked into casting a ballot using an incorrect list of candidates, an incorrect data format or to submit their ballot to an incorrect voting server for

instance. If the voter verifies that his ballot was properly recorded or counted based on incorrect election setup parameters, then the verification steps simply cannot offer any guarantee: the voter cannot even be sure that he is even actually participating in the election. This affects correctness but, similarly, if an adversary can trick a voter into encrypting his vote using an incorrect public key, then no privacy can be achieved.

Second, we observe that the purpose of this audit process is to identify when something went wrong during the election. As such, it requires a proper channel to file complains, possibly by addressing a message to the election authorities or by making public statements. Dealing with complains may be challenging, though: for instance, if a voter claim that their ballot is missing from the bulletin board, it will be hard to decide if the voter is lying in order to discredit the election, or if the bulletin board is compromised and actually suppressed a valid vote. Addressing such concerns requires *accountability*, which may be very difficult to obtain in many circumstances [KTV10].

Third, we can observe that an important feature of this audit is that it is not "just" a cryptographic verification process that can be automated: several steps require interactions with multiple people in order to verify the authenticity of data. This is central in order to achieve end-to-end verifiability, as described above. Variants of this protocol would additionally rely on signatures and on a public key infrastructure: trustees could sign their public key and decryption factors, voters could sign their ballots, and the whole bulletin board could be signed as well [CGS97, CGG19]. Such a process can certainly help avoiding that anyone submits ballots on anyone's behalf for instance. In the election process described above, authenticity is simply provided by writing the voter's name next to their ballot, and letting voters complain if they are not happy: this is sufficient to detect problems, but it makes the election process quite sensitive to people who would like to make an election fail by performing (detected) ballot stuffing for instance.

On the other hand, the reliance on a certification authority for the signature public keys as a replacement of the above in-person audit process also completely changes the trust model and would make the verification process insufficient to guarantee end-to-end verifiability. Indeed, as soon as a PKI is taken as a root of trust, the certification authority becomes capable of impersonating the trustees, the voters, and the bulletin board, which is incompatible with the end-to-end verifiability requirement.

The non-reliance on a certification authority remains challenging in practice, as it complicates a lot the verification of the authenticity of election data. In specific deployments, election organizers decided to have the hashes of important election data (e.g., the bulletin board) published in local newspapers, assuming that it would serve as a broadcast channel independent of the Internet infrastructure [AdMPQ09, CRST15].

2.2 Correctness

The protocol described above aims at offering end-to-end verifiability in a strong adversarial model: provided that voters can access an authentic bulletin board and are able to run algorithms faithfully, a successful verification process guarantees to each voter that their own ballot has been correctly included in the tally, that the tally includes all the ballots submitted by legitimate voters, and only those, and that the tally is computed correctly.

These guarantees do not cover one central element, that is, whether the ballot submitted by voters reflect their vote intent. If a voter device is malicious, or if a voter uses malicious code in order to prepare a ballot (e.g., because it is voting using a web page served by a malicious server), then the voter cannot be sure that the ciphertext that the device prepares accurately reflects her vote intent. This issue can be addressed using methods that will be discussed later in this chapter.

A simple definition a verifiability expresses that if honest voters follow the voting protocol in order to vote and perform the expected verification steps, then they are guaranteed that the election result is consistent with their votes and with an assignment of votes made for the voters for which a ballot appears on the bulletin board but who may not follow the voting protocol or follow the verification steps.

Definition 1. *A voting protocol* Π *is* verifiable *w.r.t. to the approval voting tallying rules for a single election choice if, for every probabilistic polynomial-time adversary* $\mathcal{A}$*, there is a negligible function* ϵ *such that*

$$\Pr[\mathsf{Verify}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \epsilon(\lambda),$$

where $\mathsf{Verify}_{\mathcal{A},\Pi}(\lambda)$ *is defined as the result of the following experiment.*

The election verifiability experiment $\mathsf{Verify}_{\mathcal{A},\Pi}(\lambda)$:

1. Create an empty bulletin board BB.
2. On input 1^λ, the adversary $\mathcal{A}$ publishes setup data pp.
3. The adversary $\mathcal{A}$ performs a sequence of queries to a $\mathcal{O}\mathsf{Vote}$ oracle, to which he provides pairs (id, v_{id}) made of a voter id and a vote intent v_{id}, and from which it receives ballots $(b_{id}, t_{id}) = \mathsf{Vote}(id, v_{id})$ computed from pp (as available on BB). Each of the (id, v_{id}, t_{id}) tuple is added in a set Q_H. Only one query can be made per voter id.
4. $\mathcal{A}$ returns PBB, together with an election tally v and a proof π that this tally is correct w.r.t. PBB.
5. The output of the experiment is defined to be 1 if:
   - $\mathsf{VerifyBallot}(PBB, id, t_{id}) = \top$ for every tuple $(id, *, t_{id}) \in Q_H$.
   - $\mathsf{VerifyTally}(PBB, v, \pi) = \top$.
   - $v \notin [v_h, v_h + v_c]$ where $v_h = \sum_{(id,v_{id},t_{id})\in Q_H} v_{id}$, that is, the tally restricted to the set of honest votes, and v_c is the number of corrupted voters, that is the number of ballots that appear on PBB and that do not correspond to an id included in a tuple in Q_H.

Various observations can be made about this definition.

First, our verifiability definition is parameterized by a specific tallying function: approval on one question in our case. The same definition can natually be generalized to more complex tallying functions, which may require more complex notations.

Second, we can see that this definition considers a fairly powerful adversary who is trying to manipulate the election result: this adversary can select all the keys, decide how the honest voters are voting, decide to create ballots on behalf of other voters in any way he wants, and run the tallying operations at will. Many definitions would not let the adversary choose the keys, and would keep some secret keys away from him. This is particularly important in voting systems in which voters are expected to sign their ballot and where the verification of that ballots on the bulletin board consists in verifying these signatures: such a verification becomes ineffective is the adversary can obtain everyone's signing key. The fact that we let the adversary choose all the secret keys also makes the VerifySetup algorithm irrelevant for this definition.

Third, there is an important limitation placed on the power of the adversary: it cannot corrupt the voter's device. It is assumed that the expected vote is encrypted, and that VerifyBallot and VerifyTally are executed honestly.

Fourth, we can see that this definition is fairly demanding on voters: we assume that all the honest perform the required protocol verification steps. Of course, this is very unlikely to happen in practice: we can expect that some voters will vote honestly but not verify. Our definition essentially treats them

as corrupted voters. Other definitions make a separate category of voters who vote honestly but do not perform any verification steps.

Eventually, our definition is strongly related to the specific protocol structure that we described at the beginning of our chapter. In particular, all the verification steps are performed based on a public bulletin board, and we assume that all the voters and auditors are seeing the same public bulletin board. This is a very demanding assumption, and requires mechanisms that voters can use to authenticate that board. Besides, there are various protocols that depart from this structure based on a single bulletin board, and assume that voters can access specific audit data for individual verifiability, while auditors may access other data for universal verifiability. Again, such variations can be captured using other, more general, definitions.

The interested reader can find a detailed review of definitions of verifiability by Cortier et al. [CGK+16]. Verifiability has also been considered in other related contexts such as multi-party computation [BDO14].

2.3 Privacy

A typical requirement on ballot privacy states that noone should be able to derive any information about votes that have been cast honestly, apart from what can be derived from the election result, and the knowledge that one has about the votes of corrupted voters.

For instance, if the result of an election with 100 voters is that 99 of them supported Candidate A, and if a corrupted voter cast a vote for candidate B, then it is not a breach of privacy that the adversary learns that all the honest voters voted for Candidate A. It would however be a problem if the adversary could learn this information in an election protocol where the result is expected to be the identity of the winning candidate, and not the number of votes that he received. So, the actual level of secrecy of the honest votes must depend both on the voters that are under adversarial control and on the result function that has been chosen for the election.

In the context of electronic elections, an extra concern may need to be taken into account: since non-anonymous encrypted ballots are transmitted and may be seen by corrupted parties, or may even be posted on a public bulletin board as soon as they are submitted, a corrupted voter may try to cast a ballot that is computed as a function of another voter's ballot. Again, this can have an impact on privacy: if I submit a ballot that is a copy of someone else's ballot, and if the tally shows that one candidate got one single vote, then I know that the voter whose ballot I copied did not vote for that candidate.

A simple definition of privacy proceeds by letting the adversary play in a game in which, based on election public parameters, he can submit pairs of vote intents on behalf on honest voters, ballots on behalf of corrupted voters, see the election bulletin board and a tally with it proof of correctness.

Depending on a secret bit β, the bulletin board will include ballots computed from the first ($\beta = 0$) or the second vote ($\beta = 1$) intent that was provided for honest voters. The tally, however, will always be computed based on the bulletin

board that corresponds to $\beta = 0$: this guarantees that the adversary cannot tell, by looking at the bulletin board, which vote intents are encrypted there. When $\beta = 1$, the ballots posted on the bulletin board could just as well have been computed from random vote intents. So, a bulletin board that would leak any information about the content of ballots would let the adversary win this security game. Furthermore, we provide the adversary with a "real" tally, as it is an information that he is supposed to learn in an election. From this tally, the adversary can try to derive information about ballots that he might have crafted as a function of the ballots of the honest voters posted on the bulletin board that he sees.

Definition 2. *A voting protocol* Π *has* ballot privacy *if there exist algorithms* $\mathsf{SimSetup}$ *and* $\mathsf{SimProof}$ *such that, for every probabilistic polynomial-time adversary* $\mathcal{A}$*, there is a negligible function* ϵ *such that*

$$|\Pr[\mathsf{Privacy}^0_{\mathcal{A},\Pi}(\lambda) = 1] - \Pr[\mathsf{Privacy}^1_{\mathcal{A},\Pi}(\lambda) = 1]| \leq \epsilon(\lambda),$$

where $\mathsf{Privacy}^b_{\mathcal{A},\Pi}(\lambda)$ *is defined as the result of the following experiment.*

The ballot privacy experiment $\mathsf{Privacy}^\beta_{\mathcal{A},\Pi}(\lambda)$:

1. Create two empty bulletin boards BB_0 and BB_1.
2. If $b = 0$ then $pp \leftarrow \mathsf{Setup}(1^\lambda, ED)$ is given to the adversary $\mathcal{A}$, else $pp \leftarrow \mathsf{SimSetup}(1^\lambda, ED)$ is given. The corresponding secret key sk is kept.
3. The adversary $\mathcal{A}$ performs a sequence of queries to the following oracles:
   - $\mathcal{O}\mathsf{VoteLR}(id, v_0, v_1)$, based on which the ballots produced by $\mathsf{Vote}(id, v_0)$ and $\mathsf{Vote}(id, v_1)$ are appended to BB_0 and BB_1 respectively.
   - $\mathcal{O}\mathsf{Ballot}(id, b)$, based on which b is appended to BB_0 and BB_1.
   - $\mathcal{O}\mathsf{Publish}$, based on which $\mathsf{Publish}(BB_b)$ is returned.

   Each voter id can only be used in a single query.
4. The adversary $\mathcal{A}$ performs a $\mathcal{O}\mathsf{Tally}$ query. If $b = 0$, then $\mathsf{Tally}(sk, BB_0)$ is returned. If $b = 1$ then $(v, \pi) = \mathsf{Tally}(sk, BB_0)$ is computed, and the pair $(v, \mathsf{SimProof}(sk, BB_1, v))$ is returned.
5. The adversary outputs a bit β', which is the output of the experiment.

Various observations must be made about this definition.

First, as a stand-alone definition, it is insufficient to match our intuition of privacy. Indeed, we would like to be sure that the adversary cannot learn anything more about honest votes than what he would learn when the votes are counted according to the prescribed tallying rules, e.g., a count of the number of votes for each candidate in the case of approval voting. But the prescribed

tallying rules do not appear in this definition. As a result, Tally could be defined for a different tallying function, that would leak considerably more information: for instance Tally could reveal who voted for who, or count the last vote only, and the protocol could still satisfy the above definition of ballot privacy. So, this definition of privacy must only be used on top of a requirement of consistency of the Tally function w.r.t. the desired result function, and on top of verifiability requirements that guarantee that the honest votes are taken into account properly and are not dropped for instance, or rendered invalid by carefully crafted ballots submitted by the adversary.

Second, this definition assumes that the secret key material resulting from Setup remains completely out of reach of the adversary. In a real voting system, the secret key would be distributed between multiple trustees, and it would be a requirement that privacy should hold as long as a minimum number of trustees behave honestly.

Third, this definition prevents the adversary from submitting multiple ballots on behalf of a single voter. This possibility is however offered in several voting systems, including the Estonian voting system [HMVW16]: the rule is then typically that only the last vote should be counted. Revoting can make ballot privacy more tricky, as the handling rules for the revotes must now be part of the consistency rules for the counting function, and ballot copies may become much harder to detect in the presence of a malicious bulletin board.

Fourth, this definition only aims at computational privacy. However the long-term confidentiality of the votes may be a central concern, and various protocols aiming at offering a perfectly private bulletin board content have been proposed as well, typically by encoding votes into perfectly hiding commitments [MN06, CPP13].

The interested reader can find a review of definitions of privacy in by Bernhard et al. [BCG+15], and more recent discussions in [CLW20, FQ20] for instance.

3 Going further

The protocol and definitions that we described until now are falling short to address other central requirements for a voting system. We briefly discussion two important ones.

3.1 Cast-as-intended verifiability

The protocol and security definitions that we discussed until now are defined in a model in which voters can trust their device to not disclose their vote, and to encode their vote honestly. But a malicious voting device can easily break the correctness and privacy guarantees that are typically expected in an election. Such a malicious device can leak the vote intent provided by the voter to a third party, or even encode it into the voter's ciphertext in a way that is easy to read by a third party (e.g., choose the randomness used for encryption in such a way

that the last bit of the ciphertext reveals the vote). A malicious device can also prepare a ballot for Candidate B even if the voter stated that he wants to vote for Candidate A.

Currenlty, the most convincing way to address these issues seems to remain the use of paper ballots: they offer strong guarantees that the vote intent is captured properly. Whether it is more effective to rely on hand-maked paper ballots or on ballot marking devices (BMDs) remains a difficult question: ballot marking devices can support voters and offer support to avoid casting an invalid ballot, which may be valuable when the ballot completion rules are complex, or for voters who can benefit from assistive technologies. However, they come with the risk that voters do not verify their paper ballot, and that a corrupted BMD could then get away with a paper ballot that does not reflect the voter intent. Besides, even if a cheating BMD is caught by the voter, it may be hard for a third party to decide whether the voter made a mistake or if the BMD is corrupted. Hand marked paper ballots may offer stronger guarantees that they reflect the voter intent, but they can raise usability issues and make the security of the system fully rely on the quality of the ballot chain of custody.

We will briefly discuss here the two main approaches that have been deployed to guarantee cast-as-intended verifiability when casting an electronic ballot.

3.1.1 Continuous testing

One approach that is used to detect a malicious voting device is based on continuous testing. The most famous instance of this approach is often called the "Benaloh challenge" [Ben06] – it proceeds as follows.

In a first step, the voter prepares a ballot in the usual way. However, just before casting the ballot, the voter device is required to commit on the ballot that would be cast if the voter decides to do so. This commitment can simply be the tracker provided by the Vote algorithm discussed above, and the voters takes a copy of it, by writing it down or capturing it as a QR code using a smartphone, or whatever suitable mechanism.

In a second step, the voter decides whether he wants to cast or audit the ballot that was just committed to. If the voter decides to cast the ballot, it is submitted, and the voter checks whether a ballot with the correct tracker shows on the public bulletin board. If the voter decides to audit the ballot, then the voter device must reveal to the voter the full randomness that was used to prepare the ballot. The voter can then copy that randomness and use any device(s) of his choice to verify that the ballot that was committed to reflects the expected vote intent. Audited ballots cannot be cast anymore (because of the risks for privacy), and the voter must then prepare a new ballot, that it can again choose to cast or audit.

If the voter decides to cast or audit with probability 1/2, and if the audit is performed on at least one trustworthy device, then a malicious device is caught with probability 1/2 on every attempt, which may be enough to detect any cheating at a scale that would be sufficient to modify the election result.

Running such a protocol turns out to be relatively challenging in practice.

- Voters may not be comfortable with a randomized cast-or-audit process: few of them choose to audit a ballot, and even fewer choose to audit more than one ballot – an effective cheating strategy for a ballot marking device may then be to cheat on the second prepared ballot only, which may be sufficient to change the election result when the winner margin is small [Tea21].

- The verification process fails to offer any evidence: there is no way to decide whether a complaining voter actually spotted a malicious voting device, or simply made a mistake while preparing their ballot. Of course, tests can then be repeated, and possibly filmed, but this comes with its own challenges.

- Voters may find it frustrating to prepare a ballot, audit that ballot in order to make sure that it is correct, and then be forbidden to cast that ballot. Variations on the continuous testing approcaches have been proposed, that avoid this challenge, but also come with an increased burden for the voters [aJDaPGaMT19], or with weaker privacy guarantees [HMVW16].

- Voters may find it difficult to prepare their ballot on one device and to audit it using at least one other device, while being required to transfer randomness between these devices. Various strategies have been proposed to simplify this information transfer process: some systems post the audited ballots together with their randomness on a separate bulletin board of "spoiled ballots", so that the auditing device only needs to check a bulletin board. Other systems display the randomness as a QR code, and come with a dedicated smartphone app that scans this QR code and verify the ballot [NORV14, HMVW16].

- The access to a ballot audit application may be challenging. Of course, the election organisers can provide such an application. But these organisers may also be those that serve the ballot preparation application, and the two apps could then collude easily. Independent ballot verifiers typically exist, but it is then complicated to advertise them to the voters.

Systematic studies of the effectiveness of this protocol have been perfored, including by Marky et al. for instance [MKRV18].

Overall, the continuous testing approach has the immense benefit of enabling cast-as-intended verifiability without adding any specific trust assumption. But it remains quite challenging to deploy in most practical use cases.

3.1.2 Return codes

The second main approach to cast-as-intended verifiability relies on so-called return codes, an was notably used for government elections in Norway [Gjo] and Switzerland [GGP15]. In this approach, a trusted printing service would deliver individual code sheets to the voters. The code sheet is essentially a blank ballot with two random and distinct short codes (2 alphanumeric characters, for

instance) associated to each choice on the ballot: one code corresponds to the choice being selected, the other to the choice being ignored.

Voters typically introduce a code sheet identifier into their voting device and cast their vote as usual. They must then receive random codes corresponding to their choices from the voting server, which they must compare to their code sheet. If the codes do not match, the voter complains.

Server side, the return codes are computed from the ballot by a group of trustees (without decrypting itself the ballot, of course) and, since the codes look random, even a malicious voting server cannot infer the content of a ballot from the codes.

Besides, a malicious voting client will not be able to cast a ballot with one set of choices and obtain codes corresponding to another set of codes, unless the trustees or the printing service are corrupted.

The key of the cryptographic design is to let the printing service submit to the trustees a somehow encrypted form of all the return codes in a random order, and make sure that each ballot comes with auxiliary information that makes it possible to decrypt the codes associated to the vote intent. These codes then play the role of short MACs, with additional privacy properties. Various designs are proposed and discussed by Khazaei and Wikström [KW17].

The key benefit of this approach is its improved usability compared to continuous testing [KHRV19]: voters do not need to make random choices, do not need to cast multiple ballots, and do not need to use of multiple devices.

But this comes at an important cost: trust must now be placed in specific designated system actors. If the print office leaks copies of the code sheets, then the code verification process becomes pointless. Similarly, if the trustees (and possibly the voting device) are corrupted, then any code can be computed and displayed to the voter, independently of the vote that is actually cast. This is an important departure from the verifiability trust model that is traditionally associated to end-to-end verifiability.

3.2 Receipt-freeness

In many elections, it is not sufficient to guarantee that votes remain private: the system should also make sure that voters do not get any way of proving how they voted to a third party, in order to make sure that they cannot sell their vote, or be coerced into voting in a specific way. In effect, the voting process should be *receipt-free* [BT94].

Various techniques have been proposed in order to obtain receipt-freeness, that rely on the properties of physical objects [MN06, KTR13]. We will focus here on the main cryptographic techniques.

3.2.1 Receipt-freeness by revoting

One of the main approaches to achieve receipt-freeness takes advantage of the idea of *revoting*. Estonia may be the first country to have deployed this approach, taken to its simplest form, in government elections in 2005 [CM16].

There, voters have the possibility to submit as many ballots as they want through the Internet, and only the last one will be counted. In order to prevent a vote-buyer from noticing a change in a ballot that was submitted in his presence, no public bulletin board is available (hence making it impossible for voters to verify whether their ballot is included in the tally). Voters are also allowed to vote in-person after the Internet election, which would cancel the electronic vote.

The revoting approach has been refined in several papers, one famous example being the protocol of Juels et al. [JCJ05], implemented in Civitas [CCM08]. Here, voters are assumed to perform a trusted registration procedure, during which they receive several credentials: one is real, while the others are fake but indistinguishable from the real one, and can be used by a voter who is under coercion. Ballots are posted on a public bulletin board, and a special tallying process is proposed that only counts the ballots associated to real credentials, but without letting any observer know which of the ballots have been really counted. These systems require voters to maintain multiple credentials or cryptographic tokens in order to vote, which may be quite a usability challenge in practice [KN20].

Other variations aim at improving usability by relying on a single credential per voter, but offer weaker forms of receipt-freeness. For instance, Caveat Coercitor [GRBR13] only aims at canceling all the votes that may have been submitted under coercion, rather than letting voters vote the way they want. The common aspect of all these protocols is that voters can demonstrate to a third party what is contained in any of the ballots they submit, but are unable to demonstrate which of their ballot is tallied, if any.

Receipt-freeness by hiding randomness The second main approach to receipt-freeness designs protocols in such a way that voters cannot demonstrate the content of the ballot that they are submitting, avoiding the requirement for voters to handle multiple voter credentials. In a first generation of protocols [BT94], encrypted votes for all possible options (assuming that there aren't too many of them) are prepared by multiple election authorities and transmitted to the voters, who receive the sets of ciphertexts and are informed of which ciphertext encrypt which option in a non-transferable way. The voter would then simply pick which ciphertext he wants to submit as his vote, while being ignorant of the randomness used to compute it.

This approach was refined by Hirt [Hir10], who proposed that voters should prepare an encryption of their own vote, and send it to a randomization server who is only trusted for receipt-freeness. This server re-randomizes the encrypted vote, publishes the resulting ciphertext, and sends to the voter a designated verifier proof that the new ciphertext is indeed a re-encryption of the original one. The need of an interactive ballot submission process using a designated verifier proof was removed through the design of mechanisms that make it possible for a voter to produce a signed encrypted vote in such a way that the ciphertext and signature can be re-randomized, and the signature will remain valid as long

as the content of the ciphertext is not modified [BFPV11]. The characteristics of such mechanisms were recently captured in a dedicated primitive, traceable receipt-free encryption (TREnc) [DPP22b]. However, such techniques are not sufficient to protect from a coercer who would choose and keep the signing key to himself and demand that the voter submits a given signed ballot: the voter would then be able to replace the signed ballot with another one. So, in order to support a fully non-interactive ballot submission process, current solutions still require a trusted setup phase, during which voters register their signing key [CCFG16].

4 Conclusion

This chapter scratched the surface of some of the cryptographic approaches that have been developed during the last 40 years towards having verifiable elections that keep ballots secret.

If we stick to purely electronic elections, we remain far from our goals: there remain considerable unsolved challenges to be solved before having verifiable schemes that offer receipt-free elections under broadly accepted trust assumptions while being usable by non-experts.

A large part of the research efforts of the last 20 years focused on the design of schemes that combine paper and cryptography, typically for in-person voting: we can think of Prêt-à-Voter[CRS05, CRST15], Scantegrity [CCC$^+$10] or STAR-Vote [BBE$^+$13] for instance. These schemes offer considerable security benefits, but are also relatively challenging to deploy and are not routinely used. An appealing direction that calls for further research is the security of vote-by-mail, which keeps being increasingly used in several democraties, and offers an alternative to Internet voting.

References

[AdMPQ09] Ben Adida, Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of helios. In *2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '09*. USENIX Association, 2009.

[aJDaPGaMT19] Véronique Cortier Jannik Dreier Pierrick Gaudry Mathieu Turuani. A simple alternative to benaloh challenge for the cast-as-intended property in helios/belenios. `https://hal.inria.fr/hal-02346420/`, 2019.

[BBE$^+$13] Josh Benaloh, Michael D. Byrne, Bryce Eakin, Philip T. Kortum, Neal McBurnett, Olivier Pereira, Philip B. Stark, Dan S. Wallach, Gail Fisher, Julian Montoya, Michelle

Parker, and Michael Winn. Star-vote: A secure, transparent, auditable, and reliable voting system. In *2013 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '13*. USENIX Association, 2013.

[BCG+15] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. Sok: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 499–516. IEEE Computer Society, 2015.

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In *Security and Cryptography for Networks - 9th International Conference, SCN 2014*, volume 8642 of *LNCS*, pages 175–196. Springer, 2014.

[Ben87] Josh Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, 1987.

[Ben06] Josh Benaloh. Simple verifiable elections. In *2006 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06*. USENIX Association, 2006.

[Ben22] Josh Benaloh. Electionguard specification v1.0. `https://www.electionguard.vote/`, January 2022.

[BFL+12] Jonathan Ben-Nun, Niko Fahri, Morgan Llewellyn, Ben Riva, Alon Rosen, Amnon Ta-Shma, and Douglas Wikström. A new implementation of a dual (paper and cryptographic) voting system. In *5th International Conference on Electronic Voting 2012, (EVOTE 2012)*, volume P-205 of *LNI*, pages 315–329. GI, 2012.

[BFPV11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In *Public Key Cryptography - PKC 2011*, volume 6571 of *LNCS*, pages 403–422. Springer, 2011.

[BFS+21] Josh Benaloh, Kammi Foote, Philip B. Stark, Vanessa Teague, and Dan S. Wallach. Vault-style risk-limiting audits and the inyo county pilot. *IEEE Secur. Priv.*, 19(4):8–18, 2021.

[BGP11] Philippe Bulens, Damien Giry, and Olivier Pereira. Running mixnet-based elections with helios. In *2011 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '11*. USENIX Association, 2011.

[BMN+09] Josh Benaloh, Tal Moran, Lee Naish, Kim Ramchen, and Vanessa Teague. Shuffle-sum: coercion-resistant verifiable tallying for STV voting. *IEEE Trans. Inf. Forensics Secur.*, 4(4):685–698, 2009.

[BT94] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 544–553. ACM, 1994.

[CCC+10] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II municipal election at takoma park: The first E2E binding governmental election with ballot privacy. In *19th USENIX Security Symposium*, pages 291–306. USENIX Association, 2010.

[CCFG16] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. Beleniosrf: A non-interactive receipt-free electronic voting scheme. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1614–1625. ACM, 2016.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy (S&P 2008)*, pages 354–368. IEEE Computer Society, 2008.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology - CRYPTO '94*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

[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. IEEE Computer Society, 1985.

[CGG19] Véronique Cortier, Pierrick Gaudry, and Stephane Glondu. Belenios: a simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning*, volume 11565 of *LNCS*, pages 214–238. Springer, 2019.

[CGK+16] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. Sok: Verifiability notions for e-voting protocols. In *IEEE Symposium on Security and*

Privacy, SP 2016, pages 779–798. IEEE Computer Society, 2016.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science*, pages 383–395. IEEE Computer Society, 1985.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology - EUROCRYPT '97*, volume 1233 of *LNCS*, pages 103–118. Springer, 1997.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[CLW20] Véronique Cortier, Joseph Lallemand, and Bogdan Warinschi. Fifty shades of ballot privacy: Privacy against a malicious board. In *33rd IEEE Computer Security Foundations Symposium, CSF 2020*, pages 17–32. IEEE, 2020.

[CM16] Dylan Clarke and Tarvi Martens. *Real-World Electronic Voting: Design, Analysis and Deployment*, chapter E-voting in Estonia, pages 129–141. CRC Press, 2016.

[CML18] Tristan A Canare, Ronald U Mendoza, and Mario Antonio Lopez. An empirical analysis of vote buying among the poor: Evidence from elections in the philippines. *South East Asia Research*, 26(1):58–84, 2018.

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Advances in Cryptology - CRYPTO '92*, volume 740 of *LNCS*, pages 89–105. Springer, 1992.

[CPP13] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. Election verifiability or ballot privacy: Do we need to choose? In *Computer Security - ESORICS 2013*, volume 8134 of *LNCS*, pages 481–498. Springer, 2013.

[CRS05] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139. Springer, 2005.

[CRST15] Chris Culnane, Peter Y. A. Ryan, Steve A. Schneider, and Vanessa Teague. vVote: A verifiable voting system. *ACM Trans. Inf. Syst. Secur.*, 18(1):3:1–3:30, 2015.

[Cru13] Cesi Cruz. Social networks and the targeting of illegal electoral strategies. In *American Political Science Association 2013 Annual Meeting*, 2013.

[DC07] Roberto Di Cosmo. On privacy and anonymity in electronic and non electronic voting: the ballot-as-signature attack. `https://hal.archives-ouvertes.fr/hal-00142440`, April 2007.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, 2001.

[DPP22a] Henri Devillez, Olivier Pereira, and Thomas Peters. How to verifiably encrypt many bits for an election? In *Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security*, volume 13555 of *LNCS*, pages 653–671. Springer, 2022.

[DPP22b] Henri Devillez, Olivier Pereira, and Thomas Peters. Traceable receipt-free encryption. In *Advances in Cryptology - ASIACRYPT 2022*, volume 13793 of *LNCS*. Springer, 2022.

[Fed09] Federal Constitutional Court of Germany. Use of voting computers in 2005 Bundestag election unconstitutional. Press Release No. 19/2009 of 03 March 2009, 2009.

[FQ20] Ashley Fraser and Elizabeth A. Quaglia. Protecting the privacy of voters: New definitions of ballot secrecy for e-voting. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*, volume 12804 of *Lecture Notes in Computer Science*, pages 670–697. Springer, 2020.

[GGP15] David Galindo, Sandra Guasch, and Jordi Puiggali. 2015 neuchâtel's cast-as-intended verification mechanism. In *E-Voting and Identity - 5th International Conference, VoteID 2015*, volume 9269 of *LNCS*, pages 3–18. Springer, 2015.

[Gjo] Kristian Gjøsteen. The norwegian internet voting protocol. Cryptology ePrint Archive, Report 2013/473, 2013. `https://ia.cr/2013/473`.

[GK15] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *Advances in Cryptology - EUROCRYPT 2015*, volume 9057 of *LNCS*, pages 253–280. Springer, 2015.

[GRBR13] Gurchetan S. Grewal, Mark Dermot Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. Caveat coercitor: Coercion-evidence in electronic voting. In *2013 IEEE Symposium on Security and Privacy, SP 2013*, pages 367–381. IEEE Computer Society, 2013.

[Hir10] Martin Hirt. Receipt-free K-out-of-L voting based on elgamal encryption. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, volume 6000 of *LNCS*, pages 64–82. Springer, 2010.

[HMVW16] Sven Heiberg, Tarvi Martens, Priit Vinkel, and Jan Willemson. Improving the verifiability of the estonian internet voting scheme. In *Electronic Voting - First International Joint Conference, E-Vote-ID 2016*, volume 10141 of *LNCS*, pages 92–107. Springer, 2016.

[HRZ10] Feng Hao, Peter Y. A. Ryan, and Piotr Zielinski. Anonymous voting by two-round public discussion. *IET Inf. Secur.*, 4(2):62–67, 2010.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005*, pages 61–70. ACM, 2005.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proceedings of the 11th USENIX Security Symposium*, pages 339–353. USENIX, 2002.

[KHRV19] Oksana Kulyk, Jan Henzel, Karen Renaud, and Melanie Volkamer. Comparing "challenge-based" and "code-based" internet voting verification implementations. In *Human-Computer Interaction - INTERACT 2019*, volume 11746 of *LNCS*, pages 519–538. Springer, 2019.

[KLM+20] Ralf Küsters, Julian Liedtke, Johannes Müller, Daniel Rausch, and Andreas Vogt. Ordinos: A verifiable tally-hiding e-voting system. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020*, pages 216–235. IEEE, 2020.

[KN20] Oksana Kulyk and Stephan Neumann. Human factors in coercion resistant internet voting – a review of existing solutions

and open challenges. In *Electronic Voting - 5th International Joint Conference, E-Vote-ID 2020*, 2020.

[KTR13] Dalia Khader, Qiang Tang, and Peter Y. A. Ryan. Proving prêt à voter receipt free using computational security models. In *2013 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '13*. USENIX Association, 2013.

[KTV10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010*, pages 526–535. ACM, 2010.

[KW17] Shahram Khazaei and Douglas Wikström. Return code schemes for electronic voting systems. In *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017*, volume 10615 of *LNCS*, pages 198–209. Springer, 2017.

[KY02] Aggelos Kiayias and Moti Yung. Self-tallying elections and perfect ballot secrecy. In *Public Key Cryptography - PKC 2003*, volume 2274 of *LNCS*, pages 141–158. Springer, 2002.

[MKRV18] Karola Marky, Oksana Kulyk, Karen Renaud, and Melanie Volkamer. What did I really vote for? In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018*, page 176. ACM, 2018.

[MN06] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *Advances in Cryptology - CRYPTO 2006*, volume 4117 of *LNCS*, pages 373–392. Springer, 2006.

[New03] BBC News. Mafia turns to 3g video phones. `http://news.bbc.co.uk/2/hi/technology/3033551.stm`, May 2003.

[NORV14] Stephan Neumann, Maina M. Olembo, Karen Renaud, and Melanie Volkamer. Helios verification: To alleviate, or to nominate: Is that the question, or shall we have both? In *Electronic Government and the Information Systems Perspective - Third International Conference, EGOVIS 2014*, volume 8650 of *LNCS*, pages 246–260. Springer, 2014.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

[Ped91] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91*, volume 547 of *LNCS*, pages 522–526. Springer, 1991.

[PP89] Birgit Pfitzmann and Andreas Pfitzmann. How to break the direct rsa-implementation of mixes. In *Advances in Cryptology - EUROCRYPT '89*, volume 434 of *LNCS*, pages 373–381. Springer, 1989.

[Riv08] Ronald Rivest. On the notion of 'software independence' in voting systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3759–3767, 2008.

[Sch02] Frederic Charles Schaffer. What is vote buying? In *Trading Political Rights: The Comparative Politics of Vote Buying*, 2002.

[SDW08] Daniel Sandler, Kyle Derr, and Dan S. Wallach. Votebox: A tamper-evident, verifiable electronic voting system. In *Proceedings of the 17th USENIX Security Symposium*, pages 349–364. USENIX Association, 2008.

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth. In *Advances in Cryptology - EUROCRYPT '95*, volume 921 of *LNCS*, pages 393–403. Springer, 1995.

[Tea21] Vanessa Teague. Which e-voting problems do we need to solve? In *Advances in Cryptology - CRYPTO 2021*, volume 12825 of *LNCS*, pages 3–7. Springer, 2021.

[TM20] Glenn Thrush and Jennifer Medina. California republican party admits it placed misleading ballot boxes around state. `https://www.nytimes.com/2020/10/12/us/politics/california-gop-drop-boxes.html`, December 2020.

[Wik] Douglas Wikström. Verificatum. `https://www.verificatum.org/`.

[Wik08] Douglas Wikström. Simplified submission of inputs to protocols. In *Security and Cryptography for Networks, SCN 2008*, volume 5229 of *LNCS*, pages 293–308. Springer, 2008.

[Wik09] Douglas Wikström. A commitment-consistent proof of a shuffle. In *Information Security and Privacy, 14th Australasian Conference, ACISP*, volume 5594 of *LNCS*, pages 407–421. Springer, 2009.