Chapter 69

Verifiable Voting Systems

Thea Peacock[1], Peter Y. A. Ryan[1], Steve Schneider[2] and Zhe Xia[2]

[1] University of Luxembourg [2] University of Surrey

1 Introduction

The introduction of technology into voting systems can bring a number of benefits, such as improving accessibility, remote voting and efficient, accurate processing of votes. A voting system which uses electronic technology in any part of processing the votes, from vote capture and transfer through to vote tallying, is known as an 'e-voting' system. In addition to the undoubted benefits, the introduction of such technology introduces particular security challenges, some of which are unique to voting systems because of their specific nature and requirements. The key role that voting systems play in democratic elections means that such systems must not only be secure and trustworthy, but must be seen by the electorate to be secure and trustworthy. It is a challenge to reconcile the secrecy of the ballot with demonstrable correctness of the result.

There are many general challenges involved in running a voting system securely, that are common to any complex secure system, and any implementation will need to take account of these. Over and above these, we introduce a particular approach to addressing the challenge of demonstrating trustworthiness, around the key idea of 'end-to-end verifiability'. This means that every step of the processing of the votes, from vote casting through to vote tallying, can be independently verified by some agent independent of the voting system itself. In particular, the output of the system can be checked, and so the integrity of the election does not need to rely on the trustworthiness or competence of the system and its programmers, but can be demonstrated independently. 'Individual

verification' of a step is when an individual voter is able to perform some check that their vote was handled correctly. 'Universal verifiability' of a step is when any external party is able to check that the step has been carried out correctly. Typically, end-to-end verifiability of a system will include both of these kinds of verifiability. Verifiability provides an assurance that the result of the election is correct and accurate.

This chapter shows how verifiability can be provided within a voting system, by introducing several verifiable voting schemes that have been proposed. Section 2 first discusses the many security requirements on voting systems and the relationships between them. As well as verifiability, requirements include ballot secrecy, integrity, coercion-resistance and usability among others. These requirements are often in tension, and part of the challenge of designing an e-voting system is to find ways of reconciling them. Section 3 then introduces the different kinds of verifiable voting schemes. Cryptographic mechanisms are used by many schemes in order to achieve particular goals, and this is the topic of Section 4. A simple example would be to provide the voter with a receipt of how they voted in order to allow individual verifiability. In order to maintain secrecy of the ballot (even if the voter wants to prove how she voted, for example to sell the vote), the receipt could be encrypted to mask the information. The section introduces more sophisticated cryptographic mechanisms that are used, including: secret sharing to distribute trust (no individual party has total access to a critical secret such as the election secret key); zero-knowledge proofs for verification (allowing a check that an operation on a secret has been done correctly without giving away secret information); mixnets, to shuffle and decrypt a set of votes so no decrypted vote can be linked to its original encryption. An overview of several noteworthy schemes are then introduced in Section 5, and the development of a particular scheme, Prêt à Voter, is introduced in more detail in Section 6 to illustrate in more detail the considerations that go into the design of a verifiable e-voting scheme. We consider threats to such schemes in Section 7 before concluding in Section 8.

2 Security Requirements

While e-voting systems often vary widely in design and operation, they generally converge on a standard set of security requirements. These requirements are difficult to capture

and there is no consensus as yet on precise definitions. Intuitive definitions of some important security properties are as follows.

- Ballot secrecy: only the voter should know how she voted.
- Legitimacy: only registered voters may vote.
- Eligibility: a voter may vote at most once and all votes cast are genuine.
- Individual verifiability: the voter should be able to check that her vote is accurately recorded for tabulation.
- Universal verifiability: the final tally should be verifiable by any third party.
- Accuracy (Integrity): the announced tally should reflect the true count of all legitimate, cast votes.
- Receipt-freeness: a voter should not be able to prove her vote.
- Coercion resistance: A voting system is coercion resistant if the voter can vote the way she wants to, even while appearing to cooperate with the coercer.
- Robustness: the system should be able to deliver the correct result even in the event of certain, suitably defined, levels of failure of corruption.
- Availability: users should be able to access all features of a fully-functioning system during the election.

2.1 Inter-relationships and conflicts

These properties are not wholly independent. For example, ballot secrecy and anonymity can be regarded as special cases of confidentiality or privacy; individual and universal verifiability together imply integrity.

Coercion-resistance is a stronger form of receipt-freeness, which can be described as the inability of a voter to prove how she voted. For coercion-resistance to hold, the voter must be able to vote for her chosen candidates even if she appears to cooperate with the adversary during the whole voting process, from the time before the vote is cast until the final result is published.

Further explanation and discussion of these properties can be found in [57].

Observe also that tensions exist between certain properties, most markedly ballot secrecy and individual verifiability. While the requirement exists for ballot secrecy, it should also be possible to publicly verify the accuracy of both vote recording and tallying. It is a challenge to reconcile these two requirements.

2.2 Achieving system security

At a high level, any verifiable supervised scheme can be divided into two parts. At the *front-end*, voters interact with the voting client to generate their encrypted votes and then submit their votes to the voting system. At the *back-end*, the received votes are tallied and the election result is announced. Note that voters only need to be involved in the front-end, while they need not be concerned with the back-end.

2.2.1 Challenges

The privacy, receipt-freeness and integrity properties need to be considered in both the front-end and the back-end. In the front-end, the voter's intent should not be leaked, even if the voter wants to prove it to others. This requires the encrypted vote to be generated at some supervised and controlled environment, such as a voting booth. Otherwise, adversaries who see how the encrypted vote is generated will learn the voter's intent. Moreover, the receipt should only contain the encrypted vote, but not the plaintext intent. In the back-end, if each vote is decrypted individually at the end of the tally, the relationships between the received encrypted votes and the decrypted votes have to be kept private, e.g., using mixnets. Or the homomorphic property can be used to combine received encrypted votes so that no individual vote will be decrypted at the end of the tally.

In the majority of verifiable schemes, the encrypted votes are encoded using encryption algorithms. Although different key lengths can be carefully selected based on different

Note that the Farnel scheme [2] has introduced another interesting design philosophy for the receipt. Instead of each voter being provided with a receipt which contains her own encrypted vote, the voter will be given a random receipt which contains another voter's vote. Hence the receipt can contain the plaintext intent. However, because some receipts may not be given to any of the voters and they can be removed without being detected, Farnel is not fully verifiable, and we do not discuss it further in this chapter.

security requirements, this only provides computational privacy. If adversaries have unlimited computational resources, they are able to decrypt the encrypted votes on the WBB directly. Therefore, as the computational power increases and better breaking algorithms are introduced, today's encrypted votes might be decrypted some time later without using the secret key. For this reason, some researchers advocate the *everlasting privacy* property which ensures unconditional privacy that does not depend on the strength of encryption. To achieve this property, the encrypted vote could be encoded using unconditionally hiding bit commitments instead of encryption.

2.2.2 Compromises

An e-voting system may only satisfy a subset of the desirable security properties. Similarly, a system may only partially satisfy a certain property or it may satisfy a weaker form of the property.

For example, a system may be receipt-free but not coercion-resistant: a coercer may be able to obtain a voter's credentials and vote in her place. Therefore, even if the scheme is receipt-free it is not coercion-resistant. Coercion-resistance is also difficult to achieve in remote e-voting schemes if the entire voting process is unsupervised, as there is no sure way to exclude outside influence during voting. A few solutions to the problem have been devised and are discussed in Section 5.6. However, their implementation is not straightforward, and in some cases such as for complex voting methods like STV, they may be computationally infeasible in practice.

If Internet voting is required the voting administrators may have to accept the possibility of coercion. However, coercion may not be considered a serious threat in the particular voting community. It is a case of balancing system requirements against the achievable level of security, recognising and accepting the possible threats.

Absolute ballot secrecy is another strong requirement that is not always possible to achieve, for example when the outcome of voting is unanimous. Likewise a compromise may need to be reached between ballot secrecy and for example, introducing human assistance and/or audio/visual aids for disabled voters. The threat then arises of an official and/or device "learning" a vote. If legislation demands increased accessibility, then there is no choice but to implement the (typically) strong safeguards on equipment that becomes necessary.

3 Verifiable Voting Schemes

In the literature, although a large number of verifiable voting schemes have been introduced and various techniques have been used in these schemes, many of them share similar design philosophies. In this section, we review some of the design philosophies for these verifiable voting schemes. Our focus is verifiable supervised schemes, but we will also briefly explain verifiable remote voting and its limitations.

3.1 Verifiable supervised schemes

The verifiability property consists of three components: cast-as-intended, recorded-as-cast and counted-as-recorded. The first two components are related to the front-end, ensuring that the voter's encrypted vote is not only correctly generated but also properly recorded by the election system. The last component is related to the back-end to ensure that all received encrypted votes are correctly tallied. We now explain how these three components can be designed in verifiable supervised schemes.

To achieve the cast-as-intended property, the individual voter needs to verify that the encrypted vote contains her intended vote. One typical strategy is to use the cut-and-choose method. For example, after an encrypted vote is generated by the voting client, the voter randomly decides whether to audit it or cast it. Note that if the encrypted vote is audited, the voter should not be allowed to cast it as her vote. The voter can repeat the audit process as many times as she like, each time using an independently generated encrypted vote. After she is satisfied, she requests another encrypted vote and submits it without auditing. The cut-and-choose method provides probabilistic assurance that the encrypted vote is correctly generated without cheating on the part of the system. Another typical strategy requires the voting client to generate a cryptographic proof that the encryption is correctly performed, and an honest voter will accept the proof if the encrypted vote is indeed properly generated. This is the approach taken by MarkPledge discussed in Section 5.4. Different from the cut-and-choose method, this direct audit gives a much higher assurance that the voting client is honest, and the voter can cast the vote which has been audited. Auditing is however, normally more complex than the

Note that the voter should not be able to transfer this proof to others. Otherwise, this proof also proves how this voter has voted.

cut-and-choose method.

The two auditing methods can be illustrated using a simple analogy: consider the problem of ensuring an intact fortune cookie contains a fortune. Using cut-and-choose, when given a fortune cookie you can either break it open (but then it cannot be used later as a fortune cookie) and check it contains a fortune, or you can decide to accept it. If you choose to break open several and confirm they all contain a fortune, then you can be confident when you decide to accept one that this one will also contain a fortune. Alternatively to cut-and-choose, you may use high-tech X-ray equipment to scan a cookie. If the result confirms the presence of a fortune, you will have very high assurance that there is a fortune in the cookie. In this way, no cookie needs to be opened, but it needs more advanced technology than the previous method.

To achieve the recorded-as-cast property, two requirements are necessary. Firstly, there needs to be an append-only web bulletin board (WBB) that can be read by the public but can only be appended by authorised parties. Once some information is written to it, it cannot be altered or removed. In verifiable schemes, after voters submit their encrypted votes to the election system, all these votes will be published on the WBB. Secondly, each voter will be provided with a receipt which contains her encrypted vote. To verify that her vote has been recorded by the election system, the voter can later check her receipt against the WBB to verify that her vote has been correctly recorded by the election system. If not, she can use the receipt to support a complaint to the authorities. Note that this check is optional. But since the attackers do not know which votes will be checked, if they remove a few votes before they reach the WBB, this cheating behaviour will be caught with high probability.

The counted-as-recorded property is achieved by designing the tally phase so that its entire process is publicly verifiable. In other words, no vote can be added, altered or removed without being detected. For example, if some invalid votes need to be removed from the tally, it can be verified by the public that all invalid votes have been removed and no valid vote has been removed. Moreover, the election result is calculated using the remaining votes in a publicly verifiable way. Mixnets and homomorphic encryption are two typical techniques used in the tally phase. They not only ensures the counted-as-recorded property, but also protects ballot secrecy.

A similar notion to verifiability is *software independence* [53]: a voting system is

software independent when the result it reports does not depend at all on the correctness of its software. In other words, an undetected error or deliberate change in the software cannot cause an undetected error or change to the election result. Hence the software does not need to be trusted in order to have confidence in the election result, because its output can be verified for correctness. Since the software is generally the most complex and intricate element of an e-voting system, obtaining software-independence is an important counterbalance to over-reliance on the correctness of the software.

3.2 Verifiable remote schemes

The design philosophy for verifiability is similar in both verifiable supervised and remote voting schemes. However, because voters will cast their votes in an uncontrolled environment, e.g., via Internet or post, the receipt-freeness property becomes trickier to achieve. This is because adversaries may observe the voter when she is casting her vote and find out how she has voted.

In a low coercion environment, i.e., coercion and vote buying are not serious concerns, the verifiable remote scheme can be directly designed based on a verifiable supervised scheme, just ignoring the receipt-freeness property. For example, the voter generates her encrypted vote and then submits it to the election system through a remote channel. She can still check that her vote is correctly generated, but without the receipt-freeness protection, she might be coerced to vote the candidate favoured by adversaries. The Helios system [1] is an example of a verifiable remote scheme designed for a low coercion environment.

The receipt-freeness property can be achieved in verifiable remote schemes, but there needs to be an untappable channel between the voter and the election system, and information transmitted through this channel is kept private from others. A popular design principle is to add a registration phase before the election day. This phase is carried out within some controlled environment, but the voter can participate in it at any convenient time. Here we briefly explain the design philosophy of the JCJ/Civitas scheme [35, 18]; its technical details will be further explained in Section 5.6. Each voter will register a credential in the registration phase. In the voting phase, the voter should first encrypt her credential and her preferred candidate (as well as some zero-knowledge proofs), and then submit them to the election system. Because adversaries cannot distinguish a fake

credential from a real one, when the voter is being coerced, she can use a fake credential to cast a vote. Note that all votes with fake credentials will be removed from the tally in a publicly verifiable way after mixing. Later, she can cast her vote again using her real credential when she is not being coerced. However, this type of scheme also has some limitations. Firstly, voters need to perform complicated cryptographic calculations both in the registration phase and in the voting phase. Either voters need to use some trusted device or they need to possess special knowledge to perform these tasks. Secondly, if the credential is learnt by any other party, e.g., by the voting client or by adversaries using social engineering, this party can cast another vote at a later time to overwrite the voter's vote.

4 Building Blocks

In this section, we briefly describe some building blocks which are commonly used to build verifiable voting schemes. These include encryption schemes, secret sharing and threshold techniques, zero knowledge proofs, mixnets and some other useful techniques, such as blind signature, designated verifier proof, plaintext equivalent test and proxy re-encryption.

4.1 Encryption schemes

In public key encryption, anyone can encrypt a message using the public key and the encrypted message can only be decrypted by the party who possesses the corresponding secret key. Moreover, some schemes also enjoy the additive homomorphic property which is a very handy feature in verifiable voting schemes. It allows the received encrypted votes to be aggregated into a single ciphertext. To tally the election result, only this single ciphertext is decrypted so that no individual vote will be revealed.

4.1.1 RSA cipher

The RSA cipher [54] works as follows: let p and q be two large primes, where $n = pq$ and $\phi = (p-1)(q-1)$. We first select a random value e, such that $1 < e < \phi$ and $gcd(e, \phi) = 1$. Then by applying the extended Euclidean algorithm, we can compute a value d such that $1 < d < \phi$ and $ed \equiv 1 \pmod{\phi}$. Now, the RSA public key is (n, e) and

the corresponding secret key is d. To encrypt a plaintext $m \in Z_n$, we can compute the ciphertext as $c = m^e \pmod{n}$. To decrypt c, the party who knows the secret key d can compute $c^d = m^{ed} = m^{k\phi+1} = m \pmod{n}$. RSA enjoys the multiplicative homomorphic property. For example, if $E(m_1)$ and $E(m_2)$ are two RSA ciphertexts with plaintexts m_1 and m_2, then we have $E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2)$. RSA is a deterministic public-key encryption scheme, and its security is based on the factoring problem.

4.1.2 ElGamal cipher

The ElGamal cipher [25] works as follows: let p, q be two large primes such that $q|p-1$. We denote G_q as the subgroup of Z_p^* with order q. Let g be a generator of G_q. The secret key is an element $x \in Z_q$ and the corresponding public key is $y = g^x \pmod{p}$. In this chapter, if we apply the ElGamal parameters, we assume all arithmetic to be modulo p where applicable, unless otherwise stated. To encrypt a plaintext $m \in G_q$, we choose a random blinding factor $r \in Z_q$ and compute the ciphertext $E(m, r) = (G, M) = (my^r, g^r)$. Note that an ElGamal ciphertext is a pair of values of G_q. To decrypt the ElGamal ciphertext (G, M), we compute $m = G/M^x$. ElGamal enjoys the multiplicative homomorphic property, and it is a probabilistic public-key encryption scheme, which is semantically secure if the decision Diffie-Hellman assumption holds in the group G_q.

ElGamal re-encryption: Given an ElGamal ciphertext $(G, M) = (my^r, g^r)$, a party can efficiently compute a new ciphertext (G', M') that decrypts to the same plaintext as (G, M). We denote that the ciphertext (G', M') is a re-encryption of (G, M). To re-encrypt a ciphertext, the party chooses a value $s \in Z_q$ uniformly at random and computes $(G', M') = (G \cdot y^s, M \cdot g^s)$. We note that this does not require the knowledge of the secret key x, only the public parameters y and g are needed.

Exponential ElGamal cipher: This is a variant of the ElGamal cipher with an additional parameter h which is also a generator of the group G_q. To encrypt a plaintext $m \in Z_q$, we randomly choose a blinding factor $r \in Z_q$ and calculate the ciphertext as $E(m, r) = (G, M) = (h^m y^r, g^r)$. The decryption process is the same as in the ElGamal cipher, but

In deterministic encryption, the same plaintext will always be encrypted to the same ciphertext. In contrast, the same plaintext can be encrypted to different ciphertexts using probabilistic encryption.

there is no efficient algorithm to retrieve the plaintext m from h^m. Instead, if m is known to be restricted within some field, we can search the field or use pre-computed lookup tables to retrieve m.

The exponential ElGamal cipher can be re-encrypted in the same way. But differently to the ElGamal cipher, it enjoys the additive homomorphic property. For example, if $E(m_1)$ and $E(m_2)$ are two exponential ElGamal ciphertexts with plaintexts m_1 and m_2, then we have $E(m_1) \cdot E(m_2) = E(m_1 + m_2)$ and $E(m_1)^k = E(k \cdot m_1)$.

4.1.3 Paillier cipher

The Paillier cipher [48] works as follows: let n be an RSA modulus $n = pq$, where p, q are large primes. Let g be an integer of order a multiple of n modulo n^2. The public key is (g, n), and the secret key is $\lambda = lcm((p-1),(q-1))$. To encrypt a message $m \in Z_n$, we randomly choose $x \in Z_n^*$ and compute the ciphertext $c = g^m x^n \pmod{n^2}$. To decrypt c, we compute $m = L(c^\lambda \bmod n^2)/L(g^\lambda \bmod n^2) \pmod{n}$, where the L-function takes input values from the set $S_n = \{u < n^2 | u = 1 \pmod{n}\}$ and computes $L(u) = (u-1)/n$. Clearly the Paillier cipher also enjoys the additive homomorphic property, but is superior to the exponential ElGamal cipher in that it is able to retrieve the plaintext directly after the decryption.

Paillier re-encryption: Given a Paillier ciphertext $c = g^m x^n \pmod{n^2}$, we can re-encryption it without knowledge of the secret key λ. Firstly, we randomly select a value $t \in Z_n^*$, and then we calculate $c' = c \times t^n = g^m \{tx\}^n \pmod{n^2}$. Now, c' is an re-encryption of c.

4.2 Secret sharing and threshold techniques

In secret sharing and threshold techniques, the secret information (e.g., the secret key) is shared among several parties and a quorum of these parties can work together to recover the information. Their difference is that in secret sharing, there needs to be a trusted authority to distribute the secret information among all the parties. But in threshold techniques, no trusted authority is needed, and all parties can work together to generate the secret information and distribute it among themselves. Here, we review some basic secret sharing techniques and the threshold ElGamal. Note that the threshold RSA and

threshold Paillier are also feasible, but they are more complex. Refer to [8, 64, 27, 23] for their technical details.

4.2.1 Shamir's secret sharing

Shamir's secret sharing scheme [63] is the fundamental building block for many other secret sharing and threshold techniques. It works as follows: if we want to find out the solution of polynomial $f(z) = f_0 + f_1 z + \cdots + f_{k-1} z^{k-1}$ of degree $k-1$, we need to find out every value of $(f_0, f_1, \ldots, f_{k-1})$. Therefore, we need to find out at least k pairs of (z_i, m_i) values such that for each pair, we have $f(z_i) = m_i$. Therefore, if we set f_0 as the secret m, we can generate any number of m_i, such that $m_1 = f(1), m_2 = f(2), \ldots, m_n = f(n)$. Given any subset of k of these m_i values, we can find out all the coefficients of $f(z)$ by interpolation. But on the other hand, knowledge of at most $k-1$ of these values will not enable the calculation of f_0.

By using the Lagrange interpolation, the polynomial can be written as:

$$f(z) = \sum_{i=1}^{k} (m_i \prod_{j=1, j \neq i}^{k} \frac{z - z_j}{z_i - z_j})$$

Therefore

$$m = \sum_{i=1}^{k} m_i L_i$$

where

$$L_i = \prod_{j=1, j \neq i}^{k} \frac{j}{j - i}$$

4.2.2 Verifiable secret sharing

Verifiable secret sharing [24] is based on Shamir's secret sharing, but it enjoys an additional advantage: all parties can verify that the secret has been properly distributed. The authority first generates the ElGamal secret key $x \in Z_q$, where $y = g^x$ and then distributes x among a number of parties using the Shamir's secret sharing. Let

$$f(z) = f_0 + f_1 z + \cdots + f_{k-1} z^{k-1}$$

where $f_0 = x$. For $i = 0, 1, \ldots, k-1$, the authority also computes each $F_i = g^{f_i}$ and makes these values public. Then the authority can destroy the secret information x. At

this moment, any party can check whether her given secret share is correctly constructed. Suppose the j-th party has been assigned the share x_j. She verifies that

$$g^{x_j} = \prod_{l=0}^{k-1} F_j{}^{j^l}$$

If the share is properly constructed, the above equation will always hold because

$$g^{x_j} = g^{f_0 + f_1 \cdot j + \cdots + f_{k-1} \cdot j^{k-1}} = \prod_{l=0}^{k-1} g^{f_l \cdot j^l} = \prod_{l=0}^{k-1} F_j{}^{j^l}$$

4.2.3 Threshold ElGamal

The threshold ElGamal [50] works as follows: at the beginning, n parties $(P_1, P_2, \ldots, P_n)$ need to agree on the ElGamal parameters (p, q, g). Recall that p and q are large primes, where $q|p-1$, and g is a generator of G_q. Then they work together to implement the following processes:

1. P_i randomly chooses $x_i \in Z_q$ and computes $y_i = g^{x_i}$.

2. The public key y is computed as $y = \prod_{i=1}^{n} y_i$. Now all parties know the public key y, but they cannot find out the corresponding secret key $x = \sum_{i=1}^{n} x_i \pmod q$ unless they all work together. The next step is how to distribute x to all parties in a verifiable way that any subset of k parties can recover it.

3. P_i randomly chooses a polynomial $f_i(z) \in Z_q(z)$ of degree at most $k-1$ such that $f_i(0) = x_i$. Let

   $$f_i(z) = f_{(i,0)} + f_{(i,1)} z + \cdots + f_{(i,k-1)} z^{k-1}$$

   where $f_{(i,0)} = x_i$.

4. P_i computes $F_{(i,j)} = g^{f_{(i,j)}}$ for $j = 0, 1, \ldots, k-1$ and broadcasts $(F_{(i,j)})_{j=1,2,\ldots,k-1}$. (Note that $F_{(i,0)} = y_i$ is known beforehand.)

5. After every party broadcasts the $k-1$ values in the previous step, P_i sends $s_{ij} = f_i(j)$ and a signature secretly to every other party P_j where $j = 1, 2, \ldots, n$. (Note that in particular, P_i keeps s_{ii}.)

6. P_i verifies that the share s_{ji} received from P_j is consistent with the previously published values by verifying that

   $$g^{s_{ji}} = \prod_{l=0}^{k-1} F_{(j,l)}^{i^l}$$

This is because that

$$g^{s_{ji}} = g^{f_{(j,0)}+f_{(j,1)}i+\cdots+f_{(j,k-1)}i^{k-1}} = \prod_{l=0}^{k-1} g^{f_{(j,l)}i^l} = \prod_{l=0}^{k-1} F_{(j,l)}^{i^l}$$

If this check fails, P_i broadcasts that an error has been found, publishes s_{ji} and the signature, and then stops.

7. P_i computes her share of the secret key as the sum of all shares received in step 5 as

$$s_i = \sum_{j=1}^{n} s_{ji} \pmod{q}$$

As follows, P_i signs the public key y. Finally, after all parties have signed y, anyone can check whether y is agreed among all these parties.

4.3 Zero-knowledge proofs

A zero-knowledge proof allows the prover to prove some fact to the verifier without revealing the secret details of the fact. According to the definitions in *"Handbook of Applied Cryptography"* [38], it should achieve the following three properties:

- **Completeness:** Given an honest prover and an honest verifier, the protocol will succeed with overwhelming probability. The definition of overwhelming depends on the application, but generally implies that the probability of failure is not of practical significance.

- **Soundness:** If there exists an expected polynomial-time algorithm with the following property: if a dishonest prover can with non-negligible probability successfully execute the protocol with the honest verifier, then the same algorithm can be used to extract some knowledge which is essentially equivalent to the honest prover's secret.

- **Zero-knowledge:** There exists an expected polynomial-time algorithm which can produce, upon input of the assertion to be proven but without interacting with the real prover, transcripts indistinguishable from those resulting from interaction with the real prover.

4.3.1 Interactive proofs and Fiat-Shamir heuristics

Generally speaking, an interactive zero-knowledge proof works as follows:

1. The prover sends a *witness* to the verifier. The witness works as a commitment in the protocol.

2. The verifier sends a *challenge* back to the prover. The challenge could be the outcome of fair coin toss.

3. The prover sends a *response* to the verifier. The calculation of the response needs to take into account the witness, the challenge and the secret.

In the interactive zero-knowledge proof, both the prover and the verifier need to be present during the execution of the protocol. Sometimes, it will be more convenient if the prover can generate a transcript of the protocol so that the verifier can verify it at some later time. By using the Fiat-Shamir heuristic [26], this can be achieved by transferring an interactive proof into a non-interactive proof. The non-interactive zero-knowledge proof (NIZKP) normally works as follows:

1. The prover generates a *witness*.

2. The prover takes the witness as well as some other necessary information as inputs, and outputs the *challenge* using some hash function.

3. The prover calculates the *response* and then sends the transcript which includes the witness, the challenge and the response to the verifier.

The security of NIZKP, which can be proved using the *Random Oracle Model* [4], is based on the fact that the verifier cannot predict the outcome of the hash function. Otherwise, she can fabricate a proof which will be accepted by the verifier.

In the following paragraphs, we describe several zero-knowledge proofs in the interactive form. They can be transferred into non-interactive zero-knowledge proofs similarly using the Fiat-Shamir heuristics.

Here, we only illustrate the technique using examples of three-round interactive proofs. Some proofs may have more rounds, but their concept is similar.

4.3.2 Schnorr Identification Algorithm

The Schnorr Identification Algorithm [62] is widely used to prove the knowledge of the ElGamal secret key without revealing it. The basic theory is as follows: suppose p, q are two large primes where $q|p-1$. Let g be a generator of group G_q which is a subgroup of Z_p^*. Suppose $x \in Z_q$ is the secret key, and $y = g^x$ is the corresponding public key. The prover $\mathcal{P}$ can prove that she knows x without disclosing it to the verifier $\mathcal{V}$.

1. $\mathcal{P}$ randomly chooses a value $c \in Z_q$, and sends $w = g^c$ to $\mathcal{V}$.
2. $\mathcal{V}$ sends a random challenge $e \in Z_q$ back to $\mathcal{P}$.
3. $\mathcal{P}$ calculates $s = c + xe \pmod{q}$, and sends s to $\mathcal{V}$.
4. $\mathcal{V}$ checks $g^s = wy^e$.

Moreover, for an ElGamal ciphertext $(G, M) = (my^r, g^r)$, the Schnorr Identification Algorithm also can be used to prove knowledge of its plaintext m without revealing it. The protocol first proves that $\mathcal{P}$ knows the blinding factor r in g^r. Because y is a public parameter, if $\mathcal{P}$ knows r, she can retrieve m by calculating $m = G/y^r$. Therefore, the protocol also proves that $\mathcal{P}$ knows the plaintext m.

4.3.3 Chaum-Pedersen protocol

The Chaum-Pedersen protocol [16] is used to prove the equality of discrete logarithm. Suppose (g, y) is the ElGamal public key pair and the secret key is $x = log_g y$. By using the Chaum-Pedersen protocol, the prover $\mathcal{P}$ can prove to the verifier $\mathcal{V}$ that a pair (m, n) achieves the following property: $log_g y = log_m n = x$. We denote such a proof as $\mathcal{CP}(g, y, m, n)$.

1. $\mathcal{P}$ randomly chooses a value $c \in Z_q$, then he sends $U = g^c$ and $V = m^c$ to $\mathcal{V}$.
2. $\mathcal{V}$ sends a random challenge $e \in Z_q$ back to $\mathcal{P}$.
3. $\mathcal{P}$ calculates $s = c + xe \pmod{q}$, and sends s to $\mathcal{V}$.
4. $\mathcal{V}$ checks $g^s = Uy^e$ and $m^s = Vn^e$.

The Chaum-Pedersen protocol also can be used to prove that an ElGamal ciphertext $(G', M') = (Gy^s, Mg^s)$ is a re-encryption of $(G, M) = (my^r, g^r)$ without revealing the randomisation factor s. The proof is $\mathcal{CP}(y, G'/G, g, M'/M)$ which implies that there exists a value s such that $log_y(G'/G) = log_g(M'/M)$. Moreover, the Chaum-Pedersen protocol can be used to prove that an ElGamal ciphertext has been correctly decrypted.

4.3.4 Cramer-Damgård-Schoenmakers protocol

The witness hiding/indistinguishable protocol was introduced by Cramer, Damgård and Schoenmakers in [20], therefore it is also known as the CDS protocol. It can be used to prove that a party knows the solution of k out of n problems without revealing which problems she can solve. This protocol is normally used in verifiable voting schemes to prove that a ciphertext is an encryption of one value within a subset of different values. Here, we only introduce the basic theory of the CDS protocol; for its technical details, please refer to [20].

For example, there exists n different questions $Q_1, Q_2, \ldots, Q_n$. The prover $\mathcal{P}$ wants to prove to the verifier $\mathcal{V}$ that she knows the solution of one question. But $\mathcal{P}$ does not want $\mathcal{V}$ to find out which solution she knows. $\mathcal{P}$ can execute the CDS protocol with $\mathcal{V}$ as follow:

1. Suppose $\mathcal{P}$ knows the solution δ_i of the i-th question Q_i, $\mathcal{P}$ first randomly selects r_i and calculates the genuine witness t_i. $\mathcal{P}$ then randomly chooses fake challenges c_j, fake responses s_j and uses them to fabricate the witnesses t_j, where $j \neq i$. $\mathcal{P}$ sends all these witnesses $(t_1, t_2, \ldots, t_n)$ to $\mathcal{V}$.

2. $\mathcal{V}$ randomly selects a challenge c^* and sends it to $\mathcal{P}$.

3. $\mathcal{P}$ calculates $c_i = c^* - \sum_{j \neq i} c_j$. Then she calculates the real response s_i, using r_i, c_i and her knowledge δ_i. After that, $\mathcal{P}$ sends $(c_1, c_2, \ldots, c_n)$ and $(s_1, s_2, \ldots, s_n)$ to $\mathcal{V}$.

4. $\mathcal{V}$ checks that $c^* = \sum_{k=1}^{n} c_k$ and for all the questions, each of their proofs is satisfied. However, $\mathcal{V}$ will be unable to distinguish the real proof from the fake proofs.

4.4 Mixnets

A mixnet is a cryptographic building block implemented by a number of mix servers. It takes a list of encrypted values as input, and outputs a list of values (encrypted or decrypted depending on the type of mixnet) corresponding to the input list, but permuted so the links between individual inputs and outputs are hidden. When the mixnet receives a number of encrypted values as inputs, each mix server will either partially decrypt (in a decryption mix) or re-encrypt (in a re-encryption mix) each of the encrypted values and output the results to the next mix server in a permuted order. Therefore, if there exists at least one honest mix server, the relationships between the mixnet inputs and outputs will be kept private. However, the main challenge is how to efficiently prove that the mixnet has generated the correct outputs without revealing the inputs and outputs relationships.

In the literature, there are two types of mixnets: *decryption mixnets* and *re-encryption mixnets*. Their difference is that in decryption mixnets, each mix server will partially decrypt the received encrypted list and the final mixnets outputs are plaintext values. But in re-encryption mixnets, each mix server re-encrypts the received encrypted list and the final mixnets outputs are still encrypted values. In general, re-encryption mixnets are more robust and versatile, because their re-encryption phase and the decryption phase are separated. And the mix servers only need public information to carry out the re-encryption phase.

Moreover, there are also two types of methods to verify the correctness of mixnets: *cut-and-choose* and *efficient proofs*. The cut-and-choose method can be used in both decryption mixnets and re-encryption mixnets. For example, we can randomly require half of the links of the mixnet to be opened in order to check whether the partial decryption or re-encryption is done properly. However, the challenge is how to design an architecture so that the opened links still do not reveal the relationships between inputs and outputs—i.e., there are no chains of opened links that relate inputs to outputs. Another issue is that if only one value is altered within the mixnets, a single round of the cut-and-choose method only gives 50% probability to detect the cheating. To increase the probability of detecting such cheating, we need to run the audit a number of times, but this will make

Normally, this attack does not aim to dramatically change the election result, but to find out how a voter has voted.

the verification process expensive. Efficient proofs are more complex and they only work for re-encryption mixnets. Each mix server generates a transcript proof for the shuffle she has done. The proof proves that no value is added, removed or altered during the shuffle and it can be publicly verified. Otherwise, even if a single value is altered, the proof will fail with overwhelming probability.

In the following paragraphs, we briefly describe two mixnet examples: one decryption mixnet verified using the cut-and-choose method, and one re-encryption mixnet verified using the efficient proofs.

4.4.1 Chaum's mixnet and Randomised Partial Checking

Chaum's mixnet [12] works as follows: suppose $\{(K_1, K_1^{-1}), (K_2, K_2^{-1}), \ldots, (K_m, K_m^{-1})\}$ are a number of key pairs, where K_i is the public key and K_i^{-1} is the corresponding secret key (for $i = 1, 2, \ldots, m$). The public keys are all made public and each secret key is held by an individual mix server. The mixnet inputs are a list of ciphertexts $L_0 = (l_{01}, l_{02}, \ldots l_{0n})$, where the i-th ciphertext is

$$l_{0i} = K_1(K_2(\ldots(K_{m-1}(K_m(m_i, r_m), r_{m-1})\ldots), r_2), r_1)$$

This ciphertext is commonly known as an *onion* due to its layered structure. When receiving the mixnet inputs, the first mix server will use her secret key K_1^{-1} to decrypt each of the onions in L_0, and she then removes the randomisation values, shuffles the remaining values and outputs the result list L_1 onto the WBB. At this moment, there should be a value $K_2(\ldots(K_{m-1}(K_m(m_i, r_m), r_{m-1})\ldots), r_2)$ in the list L_1. But because of the shuffle, its index will be changed. As follows, the next mix server downloads the list L_1 from the WBB, decrypts each of the ciphertext using her secret key K_2^{-1}, removes the randomisation values, shuffles the remaining values and outputs the result list L_2 to the WBB. This process is continued until the ciphertext list is decrypted by all the mix servers. Finally, the last mix server will output the list L_m which contains all the plaintexts.

Chaum's mixnet can be verified using Randomised Partial Checking (RPC) [32]. To enable this, the mixnet needs to be implemented in a slightly different way. Each mix server needs to implement two shuffles and every two adjacent mix servers are paired together, as show in Figure 1. To audit the mixnet, each pair of the mix servers is verified separately as follows:

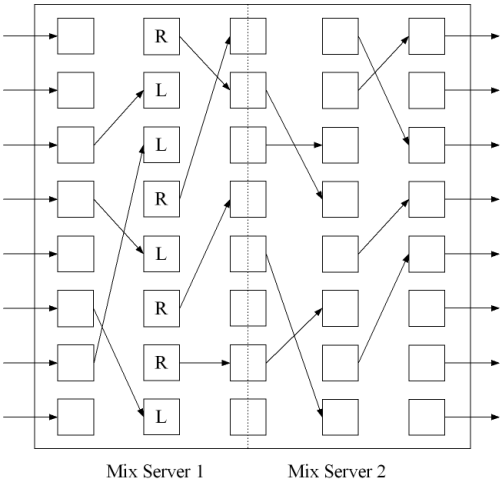

Figure 1: Randomised Partial Checking

1. For the left mix server, the auditor will go down the middle column and randomly assigns half units L and the other half units R.

2. For units assigned L, the auditor requires the left mix server to reveal the corresponding links in her first shuffle (incoming links).

3. For units assigned R, the auditor requires the left mix server to reveal the corresponding links in her second shuffle (outgoing links).

4. For the right mix server, for exactly half of the inputs she receives, their incoming links have already been revealed. We denote that these units are in the group G_1 and the other units are in the group G_2. Then the auditor randomly assigns half units in G_1 and half units in G_2 and requires the right mix server to reveal their outgoing links.

5. In the last shuffle, for the units whose incoming links have not been revealed, the right mix server is required to revealed their outgoing links.

To open a link, the mix server needs to reveal either the source of the link (for incoming links) or the destination of the link (for outgoing links) as well as the randomisation value. Therefore, the auditor who has access to the public key can re-calculate the link using the revealed information. Thanks to the above architecture, although half of the links have been audited, the remaining links still ensure that the inputs and outputs relationships

are kept private if there exists at least one pair of honest mix servers. In other words, a mixnet input can be outputted at any index with equal probability.

4.4.2 Neff's mixnet

Neff's mixnet [42, 44] works as follows: the original inputs for the mixnet are a list of ElGamal ciphertexts and each ciphertext is accompanied with a zero-knowledge proof to prove the knowledge of its plaintext. Before the shuffle starts, any input with an invalid proof will be removed and this process can be publicly verified. After that, the first mix server downloads the remaining ciphertexts from the WBB (ignores their proofs), re-encrypts each of the ciphertexts and outputs the results to the WBB in a random order. Moreover, the mix server also generates an efficient proof to prove that the re-encryption is correctly performed without revealing the shuffle. Such a proof is also published on the WBB. Then the following mix servers will implement exactly the same processes in sequence. Finally, the mixnet outputs are published onto the WBB by the last mix server.

The key contribution of Neff's mixnet is demonstrating how to construct an efficient proof. Here, we review the basic ideas. For more technical details, the readers are referred to [42, 44].

Iterated Logarithmic Multiplication Proof Protocol (ILMPP): suppose the vectors $\{X_i\}_{i=1}^k$ and $\{Y_i\}_{i=1}^k$ are publicly known, where $x_i = log_g X_i$ and $y_i = log_g Y_i$ are only known to the prover $\mathcal{P}$. Then $\mathcal{P}$ can use ILMPP to prove to the verifier $\mathcal{V}$ that $\prod_{i=1}^k x_i = \prod_{i=1}^k y_i$ without revealing any of the x_i and y_i.

The Simple k-Shuffle: suppose the vectors $\{X_i\}_{i=1}^k$ and $\{Y_i\}_{i=1}^k$ are publicly known, where $x_i = log_g X_i$ and $y_i = log_g Y_i$ are only known to the prover $\mathcal{P}$. In addition, constants $c \in Z_q$ and $d \in Z_q$ are only known to $\mathcal{P}$, and their commitments $C = g^c$ and $D = g^d$ are made public. Then $\mathcal{P}$ can prove to the verifier $\mathcal{V}$ that $Y_i^d = X_{\pi(i)}^c$ for some permutation π, without revealing any of the value x_i, y_i, c, d and π. Note that $\mathcal{P}$ actually proves that $y_i/c = x_{\pi(i)}/d$ for $i = 1, 2, \ldots, k$. The protocol works as follows:

- $\mathcal{V}$ generates a random challenge $t \in Z_q$ and sends it to $\mathcal{P}$

The proof prevents the adversary from submitting a ciphertext which is related to another ciphertext by an honest party. Otherwise, the adversary may use this attack to find out the honest party's plaintext.

- $\mathcal{P}$ and $\mathcal{V}$ publicly compute $U = D^t = g^{dt}$ and $W = C^t = g^{ct}$
- Then, for the public inputs

$$(X_1/U, X_2/U, \ldots, X_k/U, \overbrace{C, C, \ldots, C}^{k})$$

and

$$(Y_1/W, Y_2/W, \ldots, Y_k/W, \overbrace{D, D, \ldots, D}^{k})$$

$\mathcal{P}$ can use the ILMPP as a subprotocol to prove to $\mathcal{V}$ that $c^k \times \prod_{i=1}^{k}(x_i - dt) = d^k \times \prod_{i=1}^{k}(y_i - ct)$.

Note that if we divide $(cd)^k$ at both sides of the above equation, the equation can be re-written as:

$$\prod_{i=1}^{k}(x_i/d - t) = \prod_{i=1}^{k}(y_i/c - t)$$

Therefore, because t is a random value chosen by $\mathcal{V}$, $\mathcal{P}$ actually proves that for some permutation π and $i = 1, 2, \ldots, k$, we have $y_i/c = x_{\pi(i)}/d$.

ElGamal Shuffle: in a mixnet, if the mix server $\mathcal{M}$ is honest, for any output j and some permutation π, we should always have

$$(\alpha'_j, \beta'_j) = (g^{r_{\pi(j)}}\alpha_{\pi(j)}, y^{r_{\pi(j)}}\beta_{\pi(j)})$$

To prove the shuffle is correctly performed, $\mathcal{M}$ first publishes a commitment C and a random vector $\{T_j\}_{i=1}^{k}$, where $c = log_g C$ and $t_i = log_g T_j$. Then $\mathcal{M}$ can generate $U_j = T_{\pi(j)}^{c}$ and prove that this is correctly performed using the simple k-shuffle as a subprotocol. Finally, $\mathcal{M}$ just demonstrates the knowledge of $\Delta = \sum_{j=1}^{k} r_j t_j$ such that

$$log_g \frac{\prod_{j=1}^{k}(\alpha'_j)^{u_i}}{\prod_{j=1}^{k}\alpha_j^{t_i c}} = log_y \frac{\prod_{j=1}^{k}(\beta'_j)^{u_i}}{\prod_{j=1}^{k}\beta_j^{t_i c}} = \Delta c$$

If the above equation holds, it proves two facts: firstly, the same randomisation value has been used to re-encrypt both α_j and β_j. And secondly, because

$$\sum_{j=1}^{k} r_{\pi(j)} t_{\pi(j)} c = \Delta c = \sum_{j=1}^{k} r_j t_j c$$

it proves that the same permutation π has been used both in the simple k-shuffle and the ElGamal shuffle.

Note that $d = 1$ in this case.

4.5 Other useful techniques

Some other techniques are also sometimes used in designing verifiable voting schemes. Here we review four of them. The blind signature allows the signer to sign a message without learning the message content. The designated verifier proof is used to prove some fact to a designated verifier, but the verifier cannot transfer the proof to others. Plaintext equivalent test can test whether two ciphertexts are containing the same plaintext without revealing the plaintext. Proxy re-encryption is used to transfer a ciphertext encrypted under one public key to a ciphertext encrypted under another public key, where the two ciphertexts are containing the same plaintext.

4.5.1 Blind signature

Blind signature [13] is a kind of digital signature in which the message is blinded before it is signed. Therefore, the signer will not learn the message content. Then the signed message will be unblinded. At this moment, it is similar to a normal digital signature and it can be publicly checked against the original message. Blind signature can be implemented using a number of public key encryption schemes. Here, we only introduce the simplest one, which is based on RSA encryption (see Section 4.1.1). The signer has a public key is (n, e) and secret key d. Suppose a party $\mathcal{A}$ wants to have a message m signed using the blind signature. She should execute the protocol with the signer $\mathcal{S}$ as follows:

1. $\mathcal{A}$ first randomly chooses a value k which satisfies $0 \leq k \leq n-1$ and $gcd(n, k) = 1$.
2. For the message m, $\mathcal{A}$ computes $m^* = mk^e \pmod{n}$ and sends m^* to $\mathcal{S}$.
3. When receives m^*, $\mathcal{S}$ computes $s^* = (m^*)^d \pmod{n}$ and sends s^* back to $\mathcal{A}$.
4. $\mathcal{A}$ computes $s = s^*/k \pmod{n}$. Now s is $\mathcal{S}$'s signature on the message m.

4.5.2 Designated verifier proof

Designated verifier proof (DVP) [33] can be used to prove some fact, e.g., an ElGamal re-encryption is performed correctly, to a designated verifier in a way that the proof cannot be transferred to others.

Let (p, q, g) be the ElGamal parameters. Suppose s_v is the secret key of the verifier $\mathcal{V}$, and the corresponding public key is $y_v = g^{s_v}$. Denote $(G, M) = (my^\alpha, g^\alpha)$ be the original message, and $(G', M') = (Gy^\beta, Mg^\beta)$ be a re-encrypted message generated by the prover $\mathcal{P}$. $\mathcal{P}$ can prove to $\mathcal{V}$ using DVP that the re-encryption is executed properly, but $\mathcal{V}$ cannot use the same proof to convince others about this fact. The key point of the proof is to prove that G'/G and M'/M have the same discrete logarithm β under the bases g and y, respectively. A non-interactive proof of the DVP is as follows:

1. $\mathcal{P}$ chooses $k, r, t \in_R Z_q$.
2. $\mathcal{P}$ computes $(a, b) = (g^k, y^k)$ and $d = g^r {y_v}^t$.
3. $\mathcal{P}$ computes $c = H(a, b, d, G', M')$ and $u = k - \beta(c + r) \pmod q$.
4. $\mathcal{P}$ sends (c, r, t, u) to $\mathcal{V}$.
5. $\mathcal{V}$ verifies $c = H(g^u(M'/M)^{c+r}, y^u(G'/G)^{c+r}, g^r {y_v}^t, G', M')$.

If $\mathcal{P}$ has re-encrypted correctly, the honest $\mathcal{V}$ will always accept the proof because:

$$\begin{aligned} a &= g^k = g^{u+\beta(c+r)} = g^u g^{\beta(c+r)} = g^u(M'/M)^{c+r} \\ b &= y^k = y^{u+\beta(c+r)} = y^u y^{\beta(c+r)} = y^u(G'/G)^{c+r} \end{aligned}$$

Therefore

$$c = H(a, b, d, G', M') = H(g^u(M'/M)^{c+r}, y^u(G'/G)^{c+r}, g^r {y_v}^t, G', M')$$

In this protocol, $d = g^r {y_v}^t$ is a trapdoor commitment. If $\mathcal{P}$ does not know the secret key s_v, then t and r have been properly committed and $\mathcal{P}$ has to calculate u to ensure the proof will be accepted by $\mathcal{V}$. Since $\mathcal{P}$ knows β, she can find out such u. But because $\mathcal{V}$ knows the secret key s_v, she can reform $d = g^r {y_v}^t$ as $d = g^r \cdot g^{s_v t} = g^{r+s_v t}$, $\mathcal{V}$ is able to generate a fake proof for any $(\bar{G}, \bar{M}) = (m'y^\theta, g^\theta)$ that $(G', M') = (my^{\alpha+\beta}, g^{\alpha+\beta})$ is the re-encryption of $(\bar{G}, \bar{M})$. This is because that $\mathcal{V}$ can generate any pair $(\bar{r}, \bar{t})$, where $r + s_v t \equiv \bar{r} + s_v \bar{t} \pmod q$. In this case, $\mathcal{V}$ can work as the prover to fabricate a proof. She first selects $(\bar{\gamma}, \bar{\delta}, \bar{u})$ and computes

$$\bar{c} = H(g^{\bar{u}}(M'/\bar{M})^{\bar{\gamma}}, y^{\bar{u}}(G'/\bar{G})^{\bar{\gamma}}, g^{\bar{\delta}}, G', M')$$

Then $\mathcal{V}$ computes $\bar{r}$ as $\bar{r} = \bar{\gamma} - \bar{c} \pmod q$, and $\bar{t}$ to satisfy $\bar{\delta} = s_v\bar{t} + \bar{r} \pmod q$. As a result, the verifier will accept $(\bar{c}, \bar{r}, \bar{t}, \bar{u})$ as the proof because

$$\begin{aligned}
\bar{a} &= g^{\bar{u}}(M'/\bar{M})^{\bar{c}+\bar{r}} = g^{\bar{u}}(M'/\bar{M})^{\bar{\gamma}} \\
\bar{b} &= y^{\bar{u}}(G'/\bar{G})^{\bar{c}+\bar{r}} = y^{\bar{u}}(G'/\bar{G})^{\bar{\gamma}} \\
\bar{d} &= g^{\bar{r}}{y_v}^{\bar{t}} = g^{\bar{r}+s_v\bar{t}} = g^{\bar{\delta}}
\end{aligned}$$

Therefore

$$\bar{c} = H(\bar{a}, \bar{b}, \bar{d}, G', M') = H(g^{\bar{u}}(M'/\bar{M})^{\bar{\gamma}}, y^{\bar{u}}(G'/\bar{G})^{\bar{\gamma}}, g^{\bar{\delta}}, G', M')$$

4.5.3 Plaintext equivalent test

Suppose (G_1, M_1) and (G_2, M_2) are two ElGamal ciphertexts encrypted under the same public key, where the private key is threshold shared among a set of parties. The plaintext equivalent test (PET) [31] is a function to check whether the two ciphertexts are containing the same plaintext, without revealing it. Denote $(\varepsilon, \zeta) = (G_1/G_2, M_1/M_2)$, therefore if and only if the two ciphertexts contain the same plaintext, (ε, ζ) will represent an encryption of the plaintext integer 1. Each party P_j randomly selects $z_j \in Z_q$ and commits it using the Pedersen commitment [49]. Then P_j published $(\varepsilon_j, \zeta_j) = (\varepsilon^{z_j}, \zeta^{z_j})$ with the Chaum-Pedersen proof that (ε_j, ζ_j) is well formed. As follows, all parties jointly decrypt $(\gamma, \delta) = (\prod_{j=1}^{n} \varepsilon_j, \prod_{j=1}^{n} \zeta_j)$. If and only if the result plaintext is 1, the two ciphertexts (G_1, M_1) and (G_2, M_2) will contain the same plaintext.

4.5.4 Proxy re-encryption

A proxy re-encryption [30] is a function to transfer an ElGamal encryption from one encryption key to another encryption key. Let $(G_1, M_1) = (m \cdot y_1^r, g^r)$ be an ElGamal encryption of a plaintext m using public key y_1, and let x_1 be the corresponding secret key, which is shared among a number of parties using a threshold scheme. A quorum Q of these parties can transfer (G_1, M_1) to an ElGamal encryption (G_2, M_2), which contains the same plaintext with respect to the public key y_2, without revealing m. Firstly, P_j selects a value δ_j uniformly at random from Z_q, and computes $(\alpha_j, \beta_j) = (M_1^{-x_{1j}L_j}{y_2}^{\delta_j}, g^{\delta_j})$. Here x_{1j} is P_j's share of the secret key and $L_j = \prod_{i \in Q} \frac{i}{i-j}$. Then (G_2, M_2) can be computed as $(G_2, M_2) = (G_1 \prod_{j \in Q} \alpha_j, \prod_{j \in Q} \beta_j)$.

5 Survey of noteworthy schemes

In this section, we review a number of noteworthy verifiable voting schemes. Our purpose is not to cover every scheme in the literature, but we try to divide the existing schemes into several categories and we briefly describe one or two typical schemes in each category. Hopefully, this will give the readers an overview of various research works in developing verifiable voting schemes.

5.1 Schemes based on blind signature

Schemes based on blind signature were first introduced by Fujioka, Okamoto and Ohta in [28], which is normally called the FOO scheme. Although several later papers [46, 47, 45, 11] have introduced various further improvements to the FOO scheme, their election procedures are similar and the FOO scheme is still widely regarded as the milestone in this category.

The FOO scheme works as follows: the involved parties are the voters, the administrator, the counter and the WBB. At first, a certain voter selects her choice v, encrypting it by bit-commitment $\{v\}_k$ and then by blind signature $\{\{v\}_k\}_{blind}$. After that, she sends it to an administrator. The administrator will only sign the ballot if this voter is eligible and has not applied the signature before. When the voter receives the signed ballot $\{\{\{v\}_k\}_{blind}\}_{sign}$ from the administrator, she will unblind it $\{\{v\}_k\}_{sign}$ and send it to the counter through an anonymous channel. Normally, the anonymous channel is implemented by mixnets. As follows, the counter checks whether the ballot contains the administrator's signature. If yes, the counter will put it onto the WBB. Otherwise, she will reject this ballot. Now, the voter can verify whether her vote $\{v\}_k$ is correctly displayed on the WBB. If not, she can complain to a trusted party. Otherwise, she will send her de-commitment key k to the counter anonymously after some designated time T. Finally, the counter decrypts each ballot v and publishes them on the WBB.

Schemes based on blind signature ensure voter privacy and allow voters to verify that their votes are received by the election system. Furthermore, the fairness property is guaranteed so that no early result can be revealed before the designated time T. However, they also suffer several drawbacks. One issue is that messages must be sent to the election authorities twice, which means that voters have to be involved during the

whole election procedures. Another issue is that voter privacy will be violated if a voter makes discovers an incorrectly recorded receipt and complains to the authority. Moreover, if there exists some mixnet for the anonymous channel, then the blind signature technique is no longer needed to design verifiable voting schemes. Because of these issues, blind signature schemes have not attracted much recent interest.

5.2 Schemes based on mixnets

Note that the schemes based on mixnets discussed here are early schemes which assume that voters are able to generate their encrypted votes as well as the necessary proofs. So they only focus on the tally phase. Many later voter-verifiable schemes also employ mixnets as building blocks, but they concentrate on a different problem: how to allow ordinary voters to cast their encrypted votes without special knowledge.

Schemes based on mixnets have been developed along with the mixnets. In the first mixnet protocol [12], Chaum suggested that the mixnet can be used in voting schemes to provide voter privacy. And later, many mixnet protocols have used voting as an example of their application. Here, we describe the scheme introduced by Sako and Kilian in [61], and many other schemes share similar ideas.

At first, each voter generates an encrypted vote $(\alpha_i, \beta_i) = (m_i \cdot y^r, g^r)$ which contains her choice m_i. The voter also generates a Σ-proof (e.g., using the Schnorr Identification Algorithm) that she knows m_i without revealing it. Then she publishes (α_i, β_i) as well as the Σ-proof onto the WBB. After receiving all the encrypted votes from every voter, a set of mix servers will re-encrypt and shuffle these votes in sequence. Finally, the mixnet outputs will be decrypted in a threshold fashion.

For a particular mix server, suppose the list $L_{in} = \{(\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n)\}$ is her inputs and the list $L_{out} = \{(\alpha'_{\pi(1)}, \beta'_{\pi(1)}), \ldots, (\alpha'_{\pi(n)}, \beta'_{\pi(n)})\}$ is her outputs. To audit this mix server, she is required to generate another list $L_{mid} = \{(\alpha''_{\sigma(1)}, \beta''_{\sigma(1)}), \ldots, (\alpha''_{\sigma(n)}, \beta''_{\sigma(n)})\}$ which is also the re-encryption and shuffle of the list L_{in}. Then the verifier can flip a coin. If heads, the mix server needs to reveal σ and all the necessary randomisation values to prove that L_{mid} is a shuffle of L_{in}. Otherwise, if tails, the mix server will reveal $\pi \circ \sigma^{-1}$ and all the necessary randomisation values to prove that L_{out} is a shuffle of L_{mid}. It is clear that such an audit will not reveal the permutation π, and it gives 50% probability of detecting cheating if one vote has been altered during the shuffle. Moreover, the audit

can be repeated for several rounds (each round with an independently generated L_{mid}) to increase the probability of detecting cheating. Hence it can be verified that no vote has been added, altered or removed within the mixnet.

To design a verifiable voting scheme based on mixnet, another challenge is how to verify that all the mixnet outputs have been correctly decrypted. Normally, the secret key x is shared among a number of tellers in a threshold fashion as described in Section 4.2, and ciphertexts are threshold decrypted by a quorum of tellers.

5.3 Schemes based on homomorphic encryption

Schemes based on homomorphic encryption were first introduced by Josh Benaloh in [19, 7, 5]. Later, several improved schemes (e.g., [6, 21, 22, 36, 3]) were developed. These schemes follow similar election procedures, but they introduce new security properties, such as the receipt-freeness, and they use more efficient building blocks to replace those in Benaloh's schemes. Here, we review a recent scheme introduced by Baudron et al. in [3].

Suppose the maximum number of voters is M and there are k candidates. Those candidates will be assigned the values $\{M^0, M^1, \ldots, M^{k-1}\}$ respectively. Suppose also that a voter wants to vote for the i-th candidate. She first generates a Paillier ciphertext which encodes M^{i-1} as well as a proof that her ciphertext is valid. The proof is generated using the witness hiding protocol (the CDS protocol), and it proves that her plaintext is within the set $\{M^0, M^1, \ldots, M^{k-1}\}$ without revealing which one it is. Then she submits both the encrypted vote and the proof to the WBB. When the election closes, any vote with invalid proof will be removed from the tally. As follows, the remaining encrypted votes will be multiplied together into a single ciphertext. Thanks to the additive homomorphic property, this single ciphertext will encode a value R which is the sum of each individual plaintext. Moreover, the R value can be considered to contain a set of counters $\{M^0, M^1, \ldots, M^{k-1}\}$ and each counter records how many votes have been received for the corresponding candidate. For example, if a voter votes for the i-th candidate, when her encrypted vote is aggregated into the single ciphertext, the counter M^{i-1} will add one. Note that the ciphertext aggregation does not require any secret information, so anyone can check whether it is done correctly by performing the calculation again. Finally, the single ciphertext is decrypted in a threshold fashion and the value R is revealed. At this

moment, if R is divided by M^{k-1}, the result is the number of votes received by the k-th candidate. If the remainder of the previous calculation is divided by M^{k-2}, we get the number of votes received by the $(k-1)$-th candidate, and so on.

Compared with schemes based on mixnets, schemes based on homomorphic encryption are much simpler in the tallying phase. However, voters' tasks are more substantial because the witness hiding proof is more complex than the Schnorr Identification proof. Moreover, they are not as versatile as the schemes based on mixnets since they lack the ability to handle information rich elections such as Single Transferable Vote (STV) elections. Schemes based on mixnets and homomorphic encryption have received much recent interest in the literature.

5.4 Specific voter-verifiable schemes

In voter-verifiable schemes, voters are not assumed to have special knowledge to generate their votes as well as to do any necessary proofs themselves. Instead, some novel techniques can help them to generate their verifiable votes, and they can verify that their votes correctly encode their intent. We review three noteworthy schemes in this category: the MarkPledge scheme by Neff [43], a scheme using visual cryptography by Chaum [14], and Scantegrity II [15]

MarkPledge: Suppose there are n candidates $\{C_1, C_2, \ldots, C_n\}$ and κ is a security parameter. The MarkPledge scheme works as follows:

1. An authenticated voter in the voting booth will be allowed to use the voting machine. This voter tells her choice C_i to the voting machine, and meanwhile, she gives $n-1$ challenges $\{c_j\}_{j\neq i}$ to the voting machine, where each challenge is a κ bits binary. These challenges are supposed to be generated uniformly random, but if a voter is coerced to vote for the k-th candidate C_k, he can use the value given by the coercer to replace c_k.

2. The voting machine generates this voter's ballot which can be illustrated as follows:

	1	2	3		κ
C_1	$\boxed{0}\boxed{1}$	$\boxed{1}\boxed{0}$	$\boxed{0}\boxed{1}$	$\cdots$	$\boxed{1}\boxed{0}$
C_2	$\boxed{0}\boxed{1}$	$\boxed{0}\boxed{1}$	$\boxed{1}\boxed{0}$	$\cdots$	$\boxed{1}\boxed{0}$
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
C_i	$\boxed{1}\boxed{1}$	$\boxed{1}\boxed{1}$	$\boxed{0}\boxed{0}$	$\cdots$	$\boxed{1}\boxed{1}$
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
C_n	$\boxed{1}\boxed{0}$	$\boxed{0}\boxed{1}$	$\boxed{0}\boxed{1}$	$\cdots$	$\boxed{0}\boxed{1}$

Denote $\boxed{0}$ and $\boxed{1}$ as ElGamal ciphertexts with plaintexts 0 and 1 respectively. If the voting machine is honest, for the i-th candidate C_i, the voting machine generates κ pairs of ElGamal ciphertexts, where the plaintext is the same in each pair. But for all other candidates, it generates κ pairs of ElGamal ciphertexts, where the plaintext is different in each pair.

3. For the ballot generated in the previous step, the voting machine commits all pledges how each ElGamal ciphertext pair will be opened. Because for all candidates except the i-th one, the voting machine has already known how their ElGamal ciphertext pairs will be challenged, it can announce their pledges properly.

4. The voter then sends the challenge c_i for the i-th candidate to the voting machine. c_i is also a κ bits binary.

5. For all candidates, the voting machine reveals the ElGamal ciphertext pairs according to the challenge values. For example, for any candidate, if the t-th bit of the challenge is 0, the voting machine opens the left part in the t-th ElGamal ciphertext pair. Otherwise, it opens the right part.

6. This voter, as well as any party who interested, can verify that whether all opened plaintexts match what the voting machine has committed (the pledges) in the third step.

Later, all the received votes will be tallied using mixnets, which is very similar as in section 5.2. An attractive property of this scheme is that the voter does not need knowledge of cryptography to follow the election procedures. Later, whether the encrypted vote is correctly generated can be publicly checked via a cryptographic proof, and this

will not reveal how the voter has voted. Moreover, adversaries are unable to coerce the voter to vote for a particular candidate. This is because the voter provides a real proof for her preferred candidate, and decoy proofs are provided for the other candidates; anyone who checks the proofs will not know which one is real.

If the voter follows the correct election procedures in the MarkPledge scheme, the encrypted vote not only can be cast but also can be used to verify that the voting machine is honest. For a dishonest voting machine, its cheating behaviour can only go without being detected with probability $2^{-\kappa}$. However, if the voter does not understand the correct election procedures and reveals the challenge for her preferred candidate before the encrypted vote is constructed and how to open the ElGamal ciphertext pairs is pledged, the voting machine can cheat the voter by generating an encrypted vote for a different candidate. Moreover, the MarkPledge scheme lacks the ability to handle ranked elections and the size of its encrypted votes is much larger compared with many other schemes.

Chaum's visual cryptography scheme: To understand this scheme, some basic knowledge of *Visual Cryptography* [41] is necessary. There are two pixel symbols as shown in Figure 2. If we randomly choose one pixel symbol as the top layer and one pixel symbol as the bottom layer, and superimpose the two layers, the image can be illustrated in Figure 3. Thus, if the same pixel symbol occupies the same position in both layers, the image will be part-transparent. Otherwise, it will be opaque.

Figure 2: Two pixel symbols

As shown in Figure 4, Visual Cryptography can be used to convey information if both layers are superimposed, but given either the top layer or the bottom layer, it contains no useful information.

Chaum's visual cryptography scheme works as follows:

1. In the voting booth, an authenticated voter will be allowed to use the voting machine. She first reveals her choice to the voting machine.

2. The machine then prints a ballot image, similar as shown in Figure 4. In both layers,

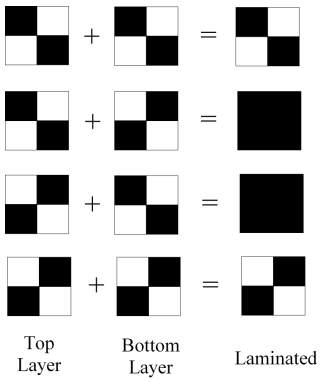

Figure 3: Two pixel symbols are laminated

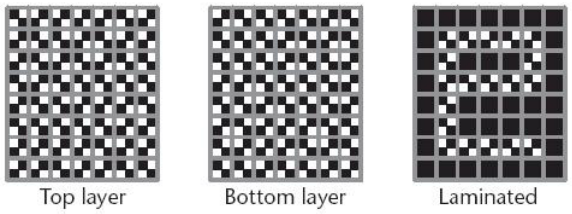

Figure 4: An application of visual cryptography

the information (θ_t, θ_b) is printed as well. If θ_t and θ_b are properly decrypted, they can be used to constructed the pixel symbols in the top layer and in the bottom layer respectively.

3. The voter checks whether the image contains her choice. If yes, she randomly chooses one layer to retain as her receipt, and the other layer needs to be destroyed.

4. Suppose this voter chooses to keep the top layer as her receipt. A copy of this layer will be published onto the WBB as her encrypted vote. Later, the voter can check whether the vote in her receipt is displayed on the WBB. If not, she can make a complaint to a trusted party using her receipt.

To tally this vote, the pixel symbols in the top layer and θ_b will enable the election authorities to recover the vote choice. The tallying phase is done using Chaum's mix [12] and incorporating Randomised Partial Checking [32] to ensure that this phase is done properly.

Furthermore, the voter can use her receipt to check whether the ballot has been

Note that she should check that the pixel symbols, θ_t and θ_b are all matching.

correctly generated by the voting machine. The voter first sends θ_t to the election authorities. Then they decode it and generate its corresponding pixel symbols. Finally, the voter compares the pixel symbols given by the election authorities and the ones printed on her receipt. As the retained layer is randomly chosen by the voter, this check gives the voter at least 50% chance to detect the cheating if the voting machine is dishonest.

In Chaum's scheme, the voting procedures are straightforward and it has the potential to handle various election methods (e.g., ranked elections). We will show later that its user interface can be further improved using the Prêt à Voter style ballot forms.

Scantegrity II. This scheme [15] augments existing optical scan voting systems with voter verifiability. Optical scan systems are already in common use in the U.S.: voters mark 'bubbles' on a paper ballot against their choices, and the ballot is read by an optical scanner, and later tallied. The ballot forms are retained by the system.

Scantegrity II introduces a random secret code (e.g. a 3 character code) with each bubble. The codes are fixed in advance and pre-committed cryptographically so they cannot be changed during or after the election. The code is revealed only when that bubble is marked. This is achieved using invisible ink for the code, and special pens to mark the bubbles, such that the code is revealed. The voter makes a private note of the code and ballot serial number, and the ballot form is scanned and retained as before. The codes received are published against the ballot serial numbers, and the voter can verify if the published code matches her record. If it does not then she can raise a challenge using her record of the code—a genuine code from the ballot form is considered as evidence of casting that vote, since it is extremely unlikely for a voter to guess a genuine code.

Voters can also audit ballot forms using cut-and-choose to check that the codes printed on them match the pre-committed codes, by revealing all of the codes. Hence voters can check that ballot forms are correctly formed, and thus verify that their vote has been captured and recorded correctly.

5.5 Non-crypto schemes

Verifiable voting schemes also can be designed without using cryptography. Here we describe two interesting examples: one was introduced by Randell and Ryan in [51], and the other one is the ThreeBallot scheme by Rivest [52].

Randell & Ryan's scheme: It is a variant of the Prêt à Voter protocol (we will describe Prêt à Voter in more detail in the next section.). The ballot form, as shown in Figure 5, has a perforation down the middle. The left hand side (LHS) lists the candidate names in a random order, and the candidate ordering varies in different ballots. At the bottom of the LHS, there is a unique *vote identification number* (VIN). The voter will use the right hand side RHS to mark her choice. At the bottom of the RHS, there is a number to record the *order of the candidate names* (OCN), but it is overprinted with a scratch strip, and the same VIN number is printed on top of the scratch strip.

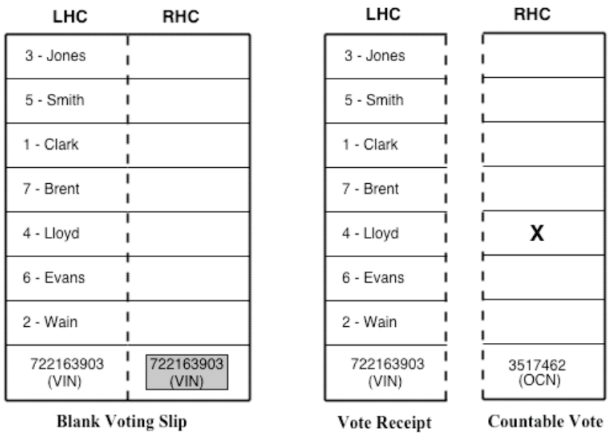

Figure 5: A ballot form example in the Randell & Ryan scheme

In the polling station, an eligible voter will be given a random ballot. To prevent others from seeing the candidate ordering, each ballot can be distributed within an envelope. The voter takes the ballot into the voting booth, marks the choice against her preferred candidate and separates the ballot along the perforation. Then she keeps the LHS as her receipt and submits the RHS without removing the scratch surface. Note that the election officials will only accept a ballot if its scratch surface is intact. Later, the VIN values for all the received ballots are published on the WBB. The voter can use her receipt to check whether her VIN number has been correctly recorded. To tally the votes, the election officials first remove the scratch surface of all received ballots. Then for each vote, its selected candidate can be retrieved using the position of the mark and the OCN value.

Apart from casting a ballot, the voter can also audit the ballot. The audit checks

that the OCN value correctly represents the candidate ordering, and this will prevent a mark against one candidate being counted as a vote for another candidate. The voter can audit as many ballots as she likes, but she can only use an unaudited ballot to cast her vote. The reason is that once a ballot has been audited, its scratch surface will be removed.

The ThreeBallot scheme: A ballot form, as shown in Figure 6, consists of three parts which can be separated along the perforations between them. In each part, the candidate names are listed in the canonical order. But a unique value at the bottom of each part is different. To cast a vote, the voter should proceed row by row through the ballot form. Each row corresponds to one candidate, and there are three bubbles in each row, one on each part. To vote for a candidate, the voter must fill in exactly two of the bubbles on that candidate row and not to vote for a candidate, she must fill in exactly one of the bubbles. Then the voter inserts her ballot into some trusted machine which checks whether the ballot is correctly filled in. If yes, the machine prints some marks (e.g., a red line) on the ballot to prove it is valid. Otherwise, it will reject the ballot. Suppose the voter has filled in a valid ballot. Then she separates the ballot along the perforations, and submits all the three parts to the election officials. At this moment, the voter is allowed to randomly choose one of three parts, make a photocopy of it and take the photocopy home as her receipt.

BALLOT		BALLOT		BALLOT	
President		President		President	
Alex Jones	○	Alex Jones	○	Alex Jones	○
Bob Smith	○	Bob Smith	○	Bob Smith	○
Carol Wu	○	Carol Wu	○	Carol Wu	○
Senator		Senator		Senator	
Dave Yip	○	Dave Yip	○	Dave Yip	○
Ed Zinn	○	Ed Zinn	○	Ed Zinn	○
3147524		7523416		5530219	

Figure 6: A ballot form example in the ThreeBallot scheme

Later, all the received ballot parts will be published on the WBB. The voter can then

use her receipt to check whether it is correctly displayed. Otherwise, the receipt can be used as the proof to make a complaint. Note that if any part of the voter's ballot has been altered or removed from the WBB, she will have at least 33% probability to detect the cheating. Finally, the election result is calculated using the recorded ballot parts on the WBB, and this process can be publicly verified. Suppose there are n voters, and the candidates $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ have received a, b and c marked bubbles respectively. Then the actual votes received by each voter is calculated by subtracting n from a, b, c respectively.

Because no complicated crypto technique is used in these non-crypto voting schemes, they are easy to understand and can be used to explain the basic ideas of verifiable voting schemes to ordinary people without special knowledge. However, they normally require much stricter assumptions than their crypto counterparts. For example, in Randell & Ryan's scheme, for any received ballot, it is important that its scratch strip is not removed by dishonest election officials before the tally. Otherwise, this ballot will be treated as invalid and it will not be included in the tally. In the ThreeBallot scheme, election integrity relies on the fact that the machine to check the ballot validity is honest. Moreover, the election officials should not know which part of the ballot has been photocopied as the receipt. Otherwise, they can replace the other parts without being detected. Therefore, the non-crypto schemes are normally viewed as academic proposals rather than practical proposals.

5.6 Remote voting schemes

All verifiable voting schemes introduced above require voters to cast their votes in the voting booth. However, this is not convenient for voters who are not able to attend the voting booth on the election day. To solve this issue, several verifiable remote voting schemes have been introduced. However, because voters are not protected from adversaries when they cast their votes, many of these schemes do not ensure the same security level as the supervised voting schemes. Here, we describe a novel remote voting scheme that not only achieves end-to-end verifiability, but also provides very high level coercion protection to the voters. Because this scheme was introduced by Juels, Catalano

Apart from the privacy and receipt-freeness properties offered by many supervised voting schemes, it also provides protection against three other attacks: the randomisation attack, the forced abstention

and Jakobsson in [35], it is normally referred as the JCJ scheme. This scheme works as follows:

1. *Registration phase:* Before the election, every voter needs to register herself in some controlled environment. Once being authenticated, the voter V_i will be provided with a credential σ_i, which is generated by some trusted party. After that, the election authorities encrypt this credential using the ElGamal encryption as $E_{pk}(\sigma_i)$, where pk is a public key and its corresponding secret key is threshold shared among a number of tellers. This ciphertext will be published onto the WBB in a list L. Then the authorities prove to V_i that this process is executed properly using the DVP protocol. Note that the voter needs to remember her credential σ_i, and it could be used in many different elections.

2. *Voting phase:* suppose V_i wants to vote for the candidate m_i, she calculates two encrypted values as

$$c_i^{(1)} = E_{pk}(m_i), \;\; c_i^{(2)} = E_{pk}(\sigma_i)$$

   Then V_i submits her vote $v_i = (c_i^{(1)}, c_i^{(2)}, \delta_i)$ to the WBB through an anonymous channel, where δ_i is a zero-knowledge proof that V_i knows the plaintexts of both ciphertexts.

3. *Tallying phase:* after the election day, election authorities collect all received votes on the WBB. They first check the proof δ_i of each vote, and any vote with an invalid proof will be eliminated immediately. Then for the remaining votes which form a list L_1, the authorities perform the Plaintext Equivalent Test (PET) between every two votes in L_1. This process ensures that if several received votes use the same credential (either valid or invalid), only the last submitted one will be retained and the others are removed from the list L_1. At this moment, the remaining votes form a list L_2. Then the authorities shuffles the list L_2 using mixnets, resulting a list L_2'. As follows, these authorities perform pairwise PET checks between L_2' and L. If any vote in L_2' cannot be matched with an encrypted credential in L, it means that this vote contains an invalid credential and it will be removed from L_2'. Finally, the remaining votes form a list L_3 and it will be threshold decrypted by a quorum of tellers to reveal the election result.

attack and the simulation attack.

Obelix	
Asterix	
Idefix	
Panoramix	
	$7rJ94K$

Figure 7: A Prêt à Voter ballot form

To see why the JCJ scheme has achieved verifiability, the voter can verify that her encrypted credential $E_{pk}(\sigma_i)$ has been included in the list L and her encrypted vote v_i has reached the WBB. Also, it can be verified by the public that the tallying phase is correctly performed. If adversaries coerce the voter to reveal her credential, she can simply reveal a fake one. This is because adversaries cannot distinguish a fake credential from a real one. Moreover, if adversaries use this fake credential to cast a vote, it will be removed in the tallying phase but they are unable to find out whether their cast vote has been removed or not.

6 Prêt à Voter

Prêt à Voter [17] was inspired by Chaum's voting scheme [14], replacing the conceptually and technologically rather complex visual cryptography with a simpler device of permuting the candidate order on each ballot. Each voter is given a ballot form with an independent permutation of candidates printed down the left hand side, and an encryption of the permutation is printed on the right hand side.

The voting ceremony is as follows. The voter picks a random ballot form at the polling station. The ballot forms will be sealed in envelopes for privacy. A typical ballot form is shown in Figure 7. In the privacy of a booth, the voter marks the chosen candidate on the right hand side of the form. She then separates the two sides and destroys the LHS. Exiting the booth, she takes the RHS to be scanned and recorded by the system. The RHS is validated as cast, for example by digital signing and franking by the officials. This is retained as a receipt which she can later check against a WBB to verify that her vote has been correctly registered.

Note that due to the permutation of candidates, the vote cannot be inferred from

the RHS alone without decrypting the onion, so ensuring ballot privacy. Receipt-freeness relies on destruction of the LHS, hence elimination of the link between the LHS and RHS of the ballot.

After voting has ceased, the receipts posted to the WBB pass through a series of anonymising mixes (see Section 4.4). Decrypted votes are then published on the WBB for public verification. Ballot generation and tabulation are discussed in the following.

The way in which the vote is encoded in the receipt has a number of important consequences that distinguish it from previous schemes. Firstly, the voter is not required to communicate her choice to an encryption device. This sidesteps threats arising from the possibility of a corrupt device leaking information about votes via side-channels or subliminal channels. Secondly, ballot auditing is very clean: correct encoding of the vote follows from correct construction of the ballot form. Thus, ballot auditing is performed on the ballot forms rather than on the receipts, as would be the case for other verifiable schemes. Whether a ballot form is correctly constructed is a simple binary decision and is independent of the vote or indeed the voter. Contrast this with earlier schemes: the voter inputs a choice and the device produces one or more encryptions of this. Suppose that the voter chooses to audit this: the encryption is opened and the plaintext is compared with the claimed input. In the event that the voter claims that the decryption does not match her input, the difficulty is how to distinguish the two situations: the device is corrupt and has encrypted the wrong vote; or the voter is mistaken or lying about her input. A further difficulty is that auditing can undermine ballot privacy. Of course, a voter who intends to audit can input a dummy vote, but this requires a certain level of understanding that not all voters may possess.

Another way of phrasing this is that in Prêt à Voter, what is encrypted is not the vote itself but rather the (randomised) frame of reference in which the vote is encoded. This can be performed before the ballot form is associated with a vote or voter.

6.1 Evolution of Prêt à Voter

Since its inception, Prêt à Voter has undergone a number of changes and a number of design options have been identified. We focus here on some of the more important changes and options. This highlights some of the associated issues and challenges. In

many cases the driving force behind changes to the system has been to address particular scenarios or contexts, and the additional perceived threats that may become relevant in those circumstances. Sometimes enhancements are for practical reasons, for example to improve efficiency of the scheme or the availability of new cryptographic primitives.

6.1.1 Tabulation issues

The original versions of Prêt à Voter used RSA decryption mixes. This had the advantage of allowing the receipts to be transformed as they go through the mixes so that they emerge in the canonical frame of reference. More precisely, the final candidate permutation on the ballot is formed as the product of several permutations, each defined in a layer of the onion. As a receipt moves through the mix the server reveals the seed value encrypted at the layer in question, computes the inverse permutation and transforms the index or vector accordingly. The index/rank vector emerges in the canonical frame or reference, i.e., in the canonical candidate order. This conveniently removes all information about the permuted candidate order during the mixing.

The downside of decryption mixes is that they lack robustness and flexibility. Mixing and tabulation are intertwined: each server must hold a decryption key and the order of in which votes are decrypted must be pre-determined. Another problem is that the onions grow in size with the number of mixes and the size of the seed space for each layer. These considerations led to the introduction of re-encryption mixes and using ElGamal in place of RSA [58].

With re-encryption mixes, the mixing and decryption phases can be separated. Mix servers do not require secret keys and their ordering does not need to be pre-determined, so they can be easily replaced if any one fails. Full mixing of terms in the group is possible and the onion size is independent of the number of mixes. Perhaps most importantly, many re-encryption mixes can be run in parallel or sequence. Further, if a mix is found to be flawed it can be re-run and re-audited fairly easily. As decryption mixes are deterministic, they cannot be audited or rerun without compromising privacy.

6.1.2 Permutations of the candidate order

A re-encryption mix does not involve (partial) decryption at each stage, so there is no obvious way to mimic the construction above, i.e., to transform the index/vector while

preserving the meaning. One possibility is to leave the index/vectors invariant. An attacker could partition the mix according to the index values with possible privacy issues, but this is not necessarily a problem if the number of voters greatly outnumbers the number of voting options. In [58], this is dealt with by restricting to cyclic shifts of the candidates and exploiting the homomorphic property of ElGamal to absorb the index into the onion, thereby allowing conventional re-encryption mixes to be used. From a secrecy point of view this is enough to conceal the choice of a single candidate. It is not enough to conceal the choice in more elaborate voting methods such as STV. It is also arguably rather fragile from an integrity point of view: if an adversary has a way to alter ballots in an undetectable way and he wants to shift votes from one candidate to another he applies the appropriate shift to the index value.

There have been a number of attempts to go beyond cyclic shifts while employing re-encryption mixes in Prêt à Voter. Ryan and Teague [59] propose use of affine permutations of the candidate order whilst retaining receipt-freeness. The set of possible permutations is defined by a shift and a scaling. This (largely) eliminates the shift attack described above while only using one pair of onions.

An obvious approach to handling full permutations is to introduce n onions per ballot (where n is the number of candidates) [66, 60]. However, this becomes computationally intensive as the number of candidates grows. Handling full permutations in a re-encryption mix in a more elegant way is an ongoing research problem.

6.1.3 Leakage of Ballot Information

In the case of Prêt à Voter although no device actually learns the voter's choice, there is still the possibility of a subliminal channel attack: the entity creating the ballot forms secretly encodes information about the permutation in the ciphertext by, for example, selecting the randomisation of the encryption in such a way that a secret keyed hash applied to the onion leaks information about the candidate order. These are known as kleptographic attacks [67, 37].

A possible counter to such an attack is the use of pseudo-random, rather than pure random values. Where ballots are created on demand by a device in the booth, the randomisation factor of the encryption could for example be derived from a signature applied to a serial number printed on the ballot, or using Verifiable Random Functions

[39]. Alternatively, we can use a distributed construction of the ballots in such a way that no single entity knows sensitive information: the final candidate order, the randomisation factors etc [10, 9].

6.1.4 Coercion

"Classic" Prêt à Voter is vulnerable to certain types of coercion such as randomisation and "Italian" attacks, particularly with ranked voting methods.

In the randomisation attack, the coercer demands that the voter produces a receipt with a mark at a pre-specified position, irrespective of which candidate this represents. This is tricky to counter in any simple way, but if Benaloh challenges are available, the voter can obtain further receipts until she finds one that allows her to vote as she intends while satisfying the coercer's demands. This works quite well for simple elections with a small number of options on the ballot, but is not feasible for more complex voting methods such as STV, etc.

6.1.5 Chain-voting

This attack, which is also effective against conventional voting systems, is particularly virulent in verifiable schemes. An adversary obtains an unused ballot form, marks this with his chosen candidate and passes it to a voter, who is required to obtain a fresh ballot at the polling station but to submit the pre-marked form. The coerced voter smuggles out her unused ballot form, hands it over to the adversary, and so the chain continues.

A possible countermeasure is to use serial numbers on the ballot forms, and for officials to note the serial number when the ballot form is issued, and to check it before the vote is cast, similar to the mechanism suggested by Jones [34].

7 Threats to Verifiable Voting Systems

Failure of voters to verify their receipts against the WBB can impact on the integrity of a system. A possible countermeasure is a Verifiable Encrypted Paper Audit Trail (VEPAT) mechanism [55] in which hard copies of receipts are stored securely and used to perform independent checks against the receipts posted to the WBB. Theoretically a VEPAT may be useful in the event of a dispute but in practice, tracing individual receipts in a large

election could prove difficult.

In the "Italian" attack, an adversary may demand that the voter fill in the ballot in a very specific way that serves, with high probability as a unique identifier for that ballot. This is potentially very effective where there is a large number of options on the ballot, e.g., STV with a significant number of candidates. Such attacks are particularly problematic in verifiable schemes in which decrypted ballots are publicly posted. countermeasures such as lazy decryption [29], can mitigate this by avoiding revealing full ballots at any stage of the tabulation process, but they are computationally intensive and do not eliminate all leakage.

7.1 Authentication of receipts

With receipted schemes, anti-faking mechanisms are important both to prevent dishonest voters from discrediting an election and from a dishonest system cheating voters. Digital signatures or franking applied to receipts are possibilities but, especially with the former, can be difficult for voters to verify without easily accessible technology.

Ryan discusses more easily human-verifiable methods such as special printing or paper [56]. In practice both digital signatures and anti-counterfeiting may be necessary: digital signatures to verify ballot construction, anti-counterfeiting to protect the system against fraudulent ballot receipts. Possible mechanisms, practical implementation and associated issues are an important research questions.

7.2 Use of Cryptography

Modern cryptography appears to be perfectly suited to solving the apparent conflict between verifiability and privacy in voting systems but there are obstacles to its deployment.

Establishing understanding and trust in the mechanisms and guarantees provided by cryptographic systems is not straightforward. In addition, proper implementation of cryptography can be complicated and problematic. As the privacy afforded by cryptographic means is usually computational, there may be concerns about the long-term privacy of votes. Schemes have been devised however, to provide everlasting privacy [65, 40].

An encryption-free, paper-based voting system, conceptually similar to [51] has been described in Section 5.5. The relative simplicity of the system, together with its similarity to lottery card games may be helpful in gaining voter confidence and trust.

8 Conclusion

Conducting elections in a way that ensures that the outcome is demonstrably correct while at the same time ensuring that all ballots remain secret has been a challenge at the very foundations of democracy from the outset. The history of democracy is a constant battle between those who seek to guarantee the integrity of elections and those who seek to to undermine and corrupt the outcome. Many technologies have been applied to address this challenge, especially in the US, but none has been wholly successful. More recently, as described in this chapter, cryptographers and security experts have turned their attention to the problem. In many ways this presents a unique and especially demanding challenge: there is no "god's eye" view to tell us what the correct outcome of an election should be, and consequently a voting system can fail in a non-manifest fashion. This is in contrast to most other critical systems, for example Internet banking, avionics etc. to which voting is sometimes compared. Such comparisons are misleading however, precisely because in these applications failures are manifest and, in the case of banking at least, usually correctable.

A number of cryptographically based schemes have emerged in the last few years which hold out the promise of fully verifiable elections: where the outcome can be proved correct with minimal trust assumptions. In this chapter we have outlined some of the most notable and promising of these schemes, along with the cryptographic primitives required in their construction. Several of these schemes have been implemented and even trialled. For example the Scantegrity II scheme has been used in municipal elections in Takoma Park in the US and Prêt à Voter is currently being adapted for use in the State of Victoria in Australia.

Despite the significant advances in verifiable voting, we have yet to see significant deployment of such schemes. An interesting question then is: why has there been so little uptake to date? It appears that the main obstacle is the use of cryptography, which many stake-holders regard with suspicion. Thus, the major challenge now is to present these schemes in a way that convince the stake-holders of the security properties they afford. It is true that the concepts underlying "modern cryptography" are subtle and the arguments showing that verifiable schemes indeed achieve the claimed security properties are quite sophisticated, so it is unreasonable to expect the average voter to follow all the details. But then, people routinely use cryptography for Internet shopping

etc without understanding all the intricacies. It is to be hoped therefore that, properly presented and after a period of informed debate, verifiable schemes will find their place in supporting democracy. It would seem sensible to deploy such schemes initially for less critical elections: officials of student bodies, professional societies etc., before deployment in real, binding political elections.

Verifiable voting systems remain an active area of research and doubtless there are further breakthroughs to be made. Various challenges and open questions remain, aside from the previously mentioned challenge of overcoming the natural aversion to cryptography. A prime example is how to perform systematic analysis of a voting system as a socio-technical system, i.e., a system comprising not only technical components such as the cryptographic algorithms and protocols, but also humans and procedures etc.

In this chapter we have focussed on polling station/supervised elections. There is considerable interest in remote, in particular Internet voting. Here the challenges are even more daunting than for supervised voting and, in particular, there is no way to ensure that a coercer does not interact with the voter. Some elegant theoretical approaches to countering coercion in the remote context have been proposed, but it seems fair to say that none are sufficiently simple and understandable to be effective in practice.

References

[1] Ben Adida. Helios: web-based open-audit voting. *Proceedings of the 17th conference on Security Symposium (SS'08)*, pages 335–348, 2008. Berkeley, CA.

[2] Roberto Araújo, Ricardo F. Custódio, and Jeroen van de Graaf. A verifiable voting protocol based on Farnel. *Proceedings of IAVoSS Workshop on Trustworthy Elections (WOTE'2007)*, pages 57–64, 2007. Ottawa, Canada.

[3] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC'01)*, pages 274–283, 2001. New York, NY, USA.

[4] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. *Proceedings of the 1st ACM Conference on Computing and Communications Society (CCS'93)*, pages 62–73, 1993. New York, NY, USA.

[5] Josh Benaloh. Secret sharing homomorphisms: keeping shares of a secret secret. *Advances in CRYPTO'86*, pages 251–260, 1986. LNCS 263.

[6] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC'94)*, pages 544–553, 1994. New York, NY, USA.

[7] Josh Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. *Proceedings of the 5th ACM Symposium on Principles of Distributed Computing (PODC'86)*, pages 52–62, 1986. New York, NY, USA.

[8] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. *Advances in CRYPTO'97*, pages 425–439, 1997. LNCS 1294.

[9] Craig Burton, Chris Culnane, James Heather, Thea Peacock, Peter Y. A. Ryan, Steve Schneider, Sriramkrishnan Srinivasan, Vanessa Teague, Roland Wen, and Zhe Xia. A supervised verifiable voting protocol for the Victorian Electoral Commission. *In the 5th International Conference on Electronic Voting (EVOTE 2012)*, 2012.

[10] Craig Burton, Chris Culnane, James Heather, Thea Peacock, Peter Y. A. Ryan, Steve Schneider, Sriramkrishnan Srinivasan, Vanessa Teague, Roland Wen, and Zhe Xia. Using Prêt à Voter in the Victorian State elections. *In the 2012 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT 2012)*, 2012.

[11] Sébastien Canard, Matthieu Gaud, and Jacques Traoré. Defeating malicious servers in a blind signatures based voting system. *Proceedings of Financial Cryptography (FC'06)*, pages 148–153, 2006. LNCS 4107.

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

[13] David Chaum. Blind signature for untraceable payments. *Advances in CRYPTO'82*, pages 199–203, 1982.

[14] David Chaum. Secret ballot receipts: true voter-verifiable elections. *IEEE: Security and Privacy Magazine*, 2(1):38–47, 2004.

[15] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation

codes. In *Electronic Voting Technology Workshop / Workshop on Trustworthy Elections*, 2008.

[16] David Chaum and Torben P. Pedersen. Wallet databases with observers. *Advances in CRYPTO'92*, pages 89–105, 1992. LNCS 740.

[17] David Chaum, Peter Y. A. Ryan, and Steve Schneider. A practical voter-verifiable election scheme. *Proceedings of the 10th European Symposium on Research in Computer Science (ESORICS'05)*, pages 118–139, 2005. LNCS 3679.

[18] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: toward a secure voting system. *2008 IEEE Symposium on Security and Privacy*, 2008.

[19] Josh Cohen and Michael Fisher. A robust and verifiable cryptographically secure election scheme. *Proceedings of the 26th IEEE symposium on the Foundations of Computer Science (FOCS'85)*, pages 372–382, 1985.

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

[21] Ronald Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. *Advances in EUROCRYPT'96*, pages 72–82, 1996. LNCS 1070.

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

[23] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. *Proceedings of Public Key Cryptography (PKC'01)*, 2001. LNCS 1992.

[24] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. *Advances in CRYPTO'89*, pages 307–315, 1989. LNCS 435.

[25] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on IT*, 31(4):467–472, 1985.

[26] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. *Advances in CRYPTO'86*, pages 186–199, 1986. LNCS 263.

[27] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. *Proceedings of Financial Cryptography (FC'00)*, 2000. LNCS 1962.

[28] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. *Workshop on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, LNCS 718:244–251, 1992. LNCS 817.

[29] James Heather. Implementing STV securely in Prêt à Voter. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, CSF '07, pages 157–169. IEEE Computer Society, 2007.

[30] Markus Jakobsson. On quorum controlled asymmetric proxy re-encryption. *Proceedings of Public Key Cryptography (PKC'99)*, pages 112–121, 1999. LNCS 1560.

[31] Markus Jakobsson and Ari Juels. Mix and match: secure function evaluation via ciphertexts. *Advances in ASIACRYPT'00*, pages 162–177, 2000. LNCS 1976.

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

[33] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. *Advances in EUROCRYPT'96*, pages 143–154, 1996. LNCS 1070.

[34] Douglas Jones. A brief illustrated history of voting. http://homepage.cs.uiowa.edu/ jones/voting/pictures/.

[35] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society (WPES'05)*, pages 61–70, 2005.

[36] Emmanouil Magkos, Mike Burmester, and Vassilis Chrissikopoulos. Receipt-freeness in large-scale elections without untappable channel. *The 1st IFIP Conference on E-commerce/E-business/E-government*, pages 683–693, 2001. Zurich.

[37] Marcin Gogolewski, Marek Klonowski, Marek Przemyslaw Kubiak, Miroslaw Kutylowski, Anna Lauks, Anna and Filip Zagórski. Kleptographic attacks on e-election schemes. In *International Conference on Emerging trends in Information and Communication Security*, 2006. http://www.nesc.ac.uk/talks/639/Day2/workshop-slides2.pdf.

[38] Alfred Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[39] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *In Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS*, pages 120–130. IEEE, 1999.

[40] Tal Moran and Moni Naor. Split-ballot voting: Everlasting privacy with distributed trust. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 246–255. ACM, 2007.

[41] Moni Naor and Adi Shamir. Visual cryptography. *Advances in CRYPTO'94*, pages 1–12, 1994. LNCS 950.

[42] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. *Proceedings of the 8th ACM Conference on Computer and Communications Security (CSS'01)*, pages 116–125, 2001.

[43] C. Andrew Neff. Practical high certainly intent verification for encrypted votes. *VoteHere document*, 2004.

[44] C. Andrew Neff. Verifiable mixing (shuffling) of ElGamal pairs. *VoteHere document*, 2004.

[45] Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka, and Tatsuaki Okamoto. An improvement on a practical secret voting scheme. *Information Security'99*, pages 225–234, 1999. LNCS 1729.

[46] Tatsuaki Okamoto. An electronic voting scheme. *Proceedings of IFIP'96*, pages 21–30, 1996.

[47] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. *Proceedings of the 5th International Workshop on Security Protocols*, pages 25–35, 1997. LNCS 1361.

[48] Pascal Paillier. Public-key cryptosystems based on discrete logarithms residues. *Advances in EUROCRYPT'99*, pages 223–238, 1999. LNCS 1592.

[49] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. *Advanced in CRYPTO'91*, pages 129–140, 1991. LNCS 576.

[50] Torben P. Pedersen. A threshold cryptosystem without a trusted party. *Advances in EUROCRYPT'91*, pages 522–526, 1991. LNCS 547.

[51] Brian Randell and Peter Y. A. Ryan. Voting technologies and trust. *IEEE Security and Privacy*, 4(5):50–56, 2006.

[52] Ronald L. Rivest. The threeballot voting system. *http://theory.lcs.mit.edu/ rivest/Rivest-TheThreeBallotVotingSystem.pdf*, 2006.

[53] Ronald L. Rivest. On the notion of 'software independence' in voting systems. *Philosophical Transactions of The Royal Society A*, 366(1881):3759–3767, 2008.

[54] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Proceedings of the 21st Communication of ACM*, 21(2):120–126, 1978.

[55] Peter Y. A. Ryan. Verified encrypted paper audit trails. Technical Report Newcastle Tech Report 966, June 2006, University of Newcastle upon Tyne, 2006.

[56] Peter Y. A. Ryan. Prêt à Voter with confirmation codes. In *Electronic Voting Technology Workshop / Workshop on Trustworthy Elections*, 2011.

[57] Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à Voter: a Voter-Verifiable Voting System. *In IEEE Transactions on Information Forensics and Security (Special Issue on Electronic Voting)*, 4(4):662–673, 2009.

[58] Peter Y. A. Ryan and Steve Schneider. Prêt à Voter with re-encryption mixes. *Proceedings of the 11th European Symposium on Research in Computer Science (ESORICS'06)*, pages 313–326, 2006. LNCS 4189.

[59] Peter Y. A. Ryan and Vanessa Teague. Ballot permutations in Prêt à Voter. In *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections*, EVT/WOTE'09. USENIX Association, 2009.

[60] Peter Y. A. Ryan and Vanessa Teague. Pretty good democracy. In *Electronic Voting Technology Workshop / Workshop on Trustworthy Elections*, 2010.

[61] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme. *Advances in EUROCRYPT'95*, pages 393–403, 1995. LNCS 921.

[62] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, pages 161–174, 1991.

[63] Adi Shamir. How to share a secret. *Proceedings of 22nd Communication of ACM*, pages 612–613, 1979.

[64] Victor Shoup. Practical threshold signature. *Advances in EUROCRYPT'00*, pages 207–220, 2000. LNCS 1807.

[65] Jeroen van de Graaf. Voting with unconditional privacy: CFSY for booth voting. Cryptology ePrint Archive, Report 2009/574, 2009.

[66] Zhe Xia, Chris Culnane, James Heather, Hugo Jonker, Peter Y. A. Ryan, Steve Schneider, and Sriramkrishnan Srinivasan. Versatile Prêt à Voter: Handling multiple election methods with a unified interface. In *INDOCRYPT*, pages 98–114, 2010.

[67] Adam L. Young and Moti Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In *CRYPTO*, pages 89–103, 1996. LNCS 1109.