To appear in *ALGORITHMS AND THEORY OF COMPUTATION HANDBOOK (Second Edition), Mikhail Atallah and Marina Blanton (Eds.)*. CRC-Press.

DRAFT BOOK CHAPTER

by Berry Schoenmakers, TU Eindhoven, July 9th, 2008

Any comments appreciated, please email to `berry@win.tue.nl`

Chapter Contents

Chapter X

Voting Schemes

X.1 Introduction

Electronic voting is probably the single most controversial application in the field of information security. Almost any area in information security, from computer security and cryptographic issues to human psychology and legal issues, is brought together when designing secure electronic voting systems. Not surprisingly, a wide variety of approaches have been used and a multitude of voting systems have been proposed.

This chapter is about *cryptographic* schemes for electronic voting, or *voting schemes* for short, whose main task is to facilitate a *secure and private* way of casting and counting votes.[1] The first wave of interest in voting schemes started in the early 1980s with the publication of Chaum's paper on anonymous communication [8]. Subsequently, approaches to voting were discovered, where the emphasis was on concepts and feasibility, much like other work in cryptography in that decade. A second wave started in the 1990s with the emergence of the World Wide Web, which created a huge interest in remote voting. Also, the emphasis shifted to efficiency concerns, resulting in quite practical schemes. And, currently, we are experiencing a third wave because of the 'Florida 2000' US election fiasco, which renewed interest in voting from polling stations, leading to voting schemes which combine cryptographic and physical aspects.

Voting schemes are intended to form the cryptographic core of electronic voting systems. The general goal of these schemes is to eliminate as many security problems as possible, thereby limiting the number and the extent of the residual assumptions needed to ensure the security of the overall voting system.

For example, proper *encryption* and *authentication* of votes ensures that no illegal modifications of existing votes and insertions of extra votes is possible, whereas appropriate *redundancy* techniques from distributed computing (realizing Byzantine fault tolerance; see also Chapter **??**)

[1] The game-theoretic aspect of what choices to offer voters to ensure as much as possible that the most favored candidates actually win is outside the scope of this chapter. Social choice theory, with results such as Arrow's impossibility theorem in the 1950s (generalizing voting paradoxes such as Condorcet's paradox from the late 18th century), deals with these issues. Ideally, voting schemes are sufficiently versatile to support any electoral system such as plurality voting, approval voting, single transfer voting, etc., but often voting schemes are biased towards a particular electoral system.

ensure that no votes can be deleted from the system. However, these basic techniques which are commonly used to implement the main goals in information security, namely *Confidentiality*, *Integrity*, and *Availability* (CIA) are not sufficient for the special combinations of security properties found in electronic voting. E.g., a strict level of ballot secrecy *and* full verifiability of the election result can only be achieved by using special forms of encryption and authentication rather than conventional ones. The strongest voting schemes thus employ advanced cryptographic techniques such as threshold cryptography, homomorphic encryption, blind signatures, and zero-knowledge proofs, thereby eliminating the need for trusting individual (or small groups of) entities. This goes beyond what is common in the voting machine industry, where the design, implementation, and operation of voting machines is essentially split between a very limited number of key players (often, involving just one company, at best two or three companies). The challenge is to find the sweet spot with the best trade-off between cryptography and other security measures.

X.2 Problem Description

The main reason why voting schemes pose a challenge is the desire to achieve some form of ballot secrecy, i.e., hiding people's votes. Secret ballot elections are important to facilitate free choice, such that voters need not be concerned or afraid of the consequences of making a particular choice. Even stronger, voters should not be able to willingly exchange their votes for money or other favors. Of course, by choosing a particular candidate voters also expect some favorable results in the long run, but rich candidates should not be able to buy poor people's votes by alleviating their immediate needs.

Ballot secrecy must be maintained while ensuring at the same time that only eligible voters are able to cast votes ("one (wo)man, one vote"). The difficulty is to authenticate the voters, or rather to authenticate the secret ballots, while not creating any links between voters and their votes. In traditional elections, voters are present in person and use some physical token to cast their vote, e.g., by using a small ball of a particular color (a 'ballotta', a colored pea or bean, first Italian word) or marking a paper ballot form. Before counting the votes, these tokens are shuffled to remove any link between voters and votes (advances in forensic analysis may also necessitate cleaning of the tokens, or even require voters to wear gloves).

For remote voting, physical presence of the voter is not required. Still, voters need to be able to cast their votes in an authenticated but secret manner. Postal voting is a traditional way to facilitate remote voting. The voter sends its voted ballot sealed inside two envelopes, where the outer envelope is signed (and linked to the voter) and the inner envelope is anonymous. Each outer envelope will be checked and if okay the inner envelope is put in a ballot box. After shuffling the ballot box, the inner envelopes are opened and the votes are counted.

In general, the essence of the voting problem can be captured by formulating it as a problem in secure multiparty computation. Each voter V_i provides its vote v_i, $1 \leq i \leq n$, as *private* input, and the object is to compute the election result $f(v_1, \ldots, v_n)$ *correctly*. A simple example is a referendum, where $v_i \in \{0,1\}$ either represents a yes-vote ($v_i = 1$) or a no-vote ($v_i = 0$), and the election result is the sum of the votes $v_1 + v_2 + \cdots + v_n$. Here, it is understood that the list of eligible voters $V_1, \ldots, V_n$ is known (and has been compiled as intended—voter registration is beyond the scope of voting

schemes).

Voting schemes solving the voting problem should thus enable correct computation of $f(v_1, \ldots, v_n)$, while hiding any further information on the votes $v_1, \ldots, v_n$, and should do *nothing else*. Another way to state this is that we want to achieve the effect of an *ideal* election, which could be run using a single (completely) trusted party as follows: each voter identifies itself to the trusted party, and then tells the trusted party its vote in private; once all votes have been cast, the trusted party announces the election result. A voting scheme must emulate such an ideal election, though without relying on a single trusted party. Instead, multiple parties (or entities) will be involved in the protocols constituting a voting scheme, thereby reducing the level of trust put into each party individually.

This succinct formulation encompasses many security properties. For example, by stating that only the election result should be revealed, it means that no intermediate results or results per county or precinct should be revealed. Similarly, as only the trusted party learns the votes, voters cannot convince anyone else of how they voted. Also, there is the premise that the trusted party can actually identify voters upon accepting their votes. As identification of voters cannot be solved by cryptographic means only, we have to assume some external mechanism. For instance, a common assumption is that each (eligible) voter holds a key pair for authentication purposes, where the public key is registered as part of a Public Key Infrastructure (PKI). Voter authentication may also involve biometrics to ensure that someone is taking part in person in the voting protocol.

Voting schemes consist of several protocols involving entities, such as election officials, voters, talliers, and observers (or scrutinizers). Protocols for voting and tallying cover the main tasks of a voting scheme, while additional protocols for setting up various part of the system and possibly for various verification tasks need to be provided as well. As explained above, the goal for a voting scheme is to emulate the trusted party as close as possible, under reasonable assumptions. We will use a basic measure of resilience to indicate how well a voting scheme emulates the trusted party. A scheme will be called **k-resilient**, if at least k parties need to be corrupted in order to compromise the scheme. Similarly, a scheme is called **k-resilient w.r.t. a security property**, if at least k parties need to be corrupted to compromise the security property.

The ideal scheme outlined above is thus 1-resilient, but this is often unacceptable. Generally, a $(k+1)$-resilient system is better than a k-resilient system. In some cases, ∞-resilience is possible, for security properties that hold unconditionally. However, k-resilience (w.r.t. a security property) is not an absolute measure. Another important factor is the total number of parties running the system: if more parties are involved it is generally easier to find a collusion of corrupted parties. For example, a 1-resilient system in which any of 10 involved parties can be corrupted is worse than a 1-resilient system in which any of 5 involved parties can be corrupted. Thus, from a security point of view, introducing additional parties to a voting system must be done with care.

In this chapter we focus on the voting protocol and the closely connected tally protocol. We will discuss a variety of ways to design voting schemes of sufficient resilience. Important aspects are the assumptions needed and the building blocks used (the mere use of which may imply further assumptions). A full security analysis of a voting scheme is often elusive, because determining the security properties and their resilience, including a complete list of assumptions, is a complex task.

X.3 Building Blocks

In this section we recapitulate several (cryptographic) primitives that are typically used in the design of advanced voting schemes. Our description of each primitive may be deceptively simple, in no way reflecting the potential complexity or subtlety of implementations for the primitive. However, application of these primitives may incur considerable costs, and similar primitives need not be interchangeable.

X.3.1 Communication Primitives

Voting schemes are formulated in terms of a communication model that describes what type of channels are available between which pairs or subsets of the entities involved. The communication model hides the implementation details of these channels, which are often far from trivial and may also come with a substantial cost. Voting schemes sharing the same communication model can be compared meaningfully, but when different types of channels are used the implementation details of the channels may be required for a useful comparison.

A **bulletin board**, or **public authenticated broadcast channel**, lets a sender broadcast a message such that everybody sees the same message. Each sender has a designated section on the bulletin board, implying that senders are authenticated. In principle, a bulletin board requires a complex protocol, involving techniques such as Byzantine agreement (see Chapter ...).

The basic goal of an **anonymous channel** [8] is to hide the identity of the sender, whereas the receiver is not necessarily anonymous. A sender is able to transmit messages to an intended receiver, and possibly allows for an acknowledgement by the receiver as well. In particular, an anonymous channel must withstand traffic analysis, where an adversary monitors the activity throughout an entire network, to see who is communicating with whom.

A **private channel** allows for information-theoretically private communication between two nodes, protecting against eavesdroppers, and possibly providing end-to-end authentication as well. Private channels are typically used as building blocks in protocols for *unconditionally* secure multiparty computation (e.g., see [3]). Private channels are often not realistic to assume, as keys for one-time pad encryption (see Chapter **??**) must have been set up previously, requiring an untappable channel (see below).

A **secure channel** is similar to a private channel, but usually provides *computational* security only (for encryption and/or authentication). Secure channels provide a means for abstracting away everything required to ensure a protected link between two nodes. Commonly, secure channels are implemented by some combining some form of key exchange, resulting in session keys used for symmetric encryption and authentication. SSL is a practical example.

Finally, **untappable channels** have been introduced [5] to accommodate totally unobservable communication, preventing the adversary from capturing any of the information transmitted. The analogy here is that one may observe two persons whispering to each other, but one has no way of listening in. An untappable channel is meant to be implemented physically, by some form of out-of-band communication.

Combinations of these channels are possible as well, but care must be applied to see what is sensible, and to see what it takes for a good implementation. E.g., an anonymous channel is already

hard to implement, let alone an "untappable anonymous channel", say. Furthermore, one can refine these notions by considering *directed* links, which results in notions like "one-way untappable channels".

X.3.2 Authentication Primitives

Voter authentication is a basic issue which must be addressed by any voting scheme. An important distinction is whether the authentication mechanism is external or internal to the voting scheme. An *external* authentication mechanism provides the link with the list of eligible (or registered) voters, hence forms an integral part of any voting scheme. Depending on the voting scheme, further *internal* authentication mechanisms can be used, typically providing some form of anonymous signatures as described below.

A wide range of voter authentication mechanisms is used, generally using a mixture of physical and digital techniques. External authentication of voters may be as simple as matching voters' names against a list (electoral roll) by an election official at a polling station. Similarly, internal authentication may be done implicitly by having successfully authenticated voters line up for a voting machine, or by handing out a ticket to them, which must be handed in to the person at the voting machine. Some basic internet voting schemes use sealed envelopes containing unique authentication codes, which are sent by postal mail to the addresses of all registered voters. Here, external authentication relies on the sealed envelopes indeed reaching the intended voters, whereas the codes are used for internal authentication, possibly without the printer keeping track of who gets which code—to provide some level of anonymity.

Cryptographic techniques are used to increase the resilience of voting schemes. External authentication of voters can be based on any conventional symmetric technique (incl. password-based authentication) or asymmetric technique (digital signatures, incl. one-time signatures or PKI-based signatures), where the main security concern is protection against impersonation attacks; see Chapter **??** for more information.

For internal authentication, more advanced authentication primitives may be used, which we will collectively call **anonymous signatures**. Ordinary digital signatures are verified against the public key of the signer, revealing the identity of the signer. In many applications, however, there is a need for digital signatures providing authentication of messages without revealing the identity of the signer. Anonymous signature schemes aim at various levels of privacy for the signer, examples of which with relevancy to electronic voting, are *blind signatures* [9] (see Chapter **??**), *group signatures* [11], *restrictive blind signatures* [6], *ring signatures* [38], and *list signatures* [7]. In these schemes, all signatures are verified with respect to the public key of a designated party, called the issuer, which uses the corresponding private key to assist in the generation of signatures. The key property is that that the issuer does not learn the contents of the messages thus signed, nor is able to recognize the actual signatures thus produced.

Proper use of anonymous signatures usually requires the availability of anonymous channels. Moreover, special care must be taken, e.g., to prevent that additional blind signatures can be issued, which are then used to cast additional votes or to replace votes cast by legitimate voters (which may go unnoticed as long as the total number of votes does not exceed the total number of voters). Use of threshold cryptography to enforce that (blind) signatures are issued jointly by a number of

parties (rather than a single issuer) helps to achieve an acceptable level of resilience; see Chapter **??** for some more details.

X.3.3 Encryption Primitives

Basic symmetric and asymmetric cryptosystems as well as some more advanced and related techniques have been covered extensively in previous chapters. Voting schemes use a large variety of such techniques to let voters encrypt their votes in some secure way. Here, it should be stressed that 'vote encryption' is meant in a broad sense. Apart from encryption using a cryptosystem, we also use it to cover any application of secret sharing and/or commitment schemes. For instance, vote encryption may also be done by having a voter distributing shares of its vote to a number of parties: this way, the voter performs the role of the dealer in a secret sharing scheme, and 'decryption' is done by reconstructing the secret (vote) from the shares. Similarly, vote encryption may actually be done by publishing a commitment to the vote, in which case 'decryption' is done by later opening the commitment.

Throughout this chapter, we will use $[\![v]\!]$ as an abstract notation for any such encryption of a vote v. A basic feature of a vote encryption $[\![v]\!]$, which is suppressed from the notation, is the fact that $[\![v]\!]$ is obtained in a *probabilistic* way. This ensures semantic security, which means that given a vote encryption $[\![v]\!]$ one cannot get any information on vote v. In particular, if $[\![v]\!]$ and $[\![v']\!]$ are produced by two different voters, one cannot even decide if $v = v'$ holds, or not.

Two particular features for encryption schemes used in voting are homomorphic encryption and threshold decryption. Informally, **homomorphic encryption** satisfies the property that the *product* of two encryptions is an encryption of the *sum* of the corresponding messages:

$$[\![x]\!][\![y]\!] = [\![x+y]\!],$$

for messages x and y. For a basic example, let $\langle g\rangle = \{1, g, g^2, \ldots, g^{q-1}\}$ be a cyclic group, written multiplicatively, of large prime order q. Hence, g is a generator of order q, $g^q = 1$. Let $h \in \langle g\rangle$ be a public key. Then, an *additively* homomorphic ElGamal encryption [16] of a message $x \in \mathbb{Z}_q$ is computed as:

$$[\![x]\!] = (A, B) = (g^r, h^r g^x),$$

where r is chosen uniformly at random in $\mathbb{Z}_q$. The homomorphic property holds if we define the product of two encryptions (A, B) and (C, D) in the obvious way as (AC, BD), and we define $+$ as addition modulo q. Given the private key $\alpha = \log_g h$, encryption (A, B) is decrypted as

$$x = \log_g(b/a^\alpha).$$

Compared to standard ElGamal encryption, recovering message x from an *additively* homomorphic ElGamal encryption $[\![x]\!]$ thus involves the computation of a discrete logarithm in $\langle g\rangle$, which is in general hard. However, if x is known to be from a limited set of values, one may still recover x efficiently (e.g., if $0 \le x < u$, one may find x by a straightforward $O(u)$ linear search, or by Pollard's $O(\sqrt{u})$ lambda method [36]).

A strongly related example is Pedersen's commitment scheme [35], in which commitments are computed as:

$$[\![x]\!] = h^r g^x,$$

where r is chosen uniformly at random in $\mathbb{Z}_q$. Hence, Pedersen commitments are just elements of $\langle g \rangle$, and the homomorphic property clearly holds. Strictly speaking, $[\![x]\!]$ is not an encryption in the sense that it completely determines x (knowing g and h). The idea is that by publishing the value of $[\![x]\!]$, one is 'committed' to the value of x meaning that one can at a later stage 'open' $[\![x]\!]$ in only one way by revealing the pair (r,x) (assuming that it is infeasible for the committer to compute the discrete log $\log_g h$): indeed, the knowledge of any $(r,x) \neq (r',x')$ with $h^r g^x = h^{r'} g^{x'}$ implies that one knows $\log_g h = (x'-x)/(r-r')$ as well. Also, for a given value of $[\![x]\!] = h^r g^x$ and an hypothesized value x' for the committed message there exists exactly one $r' \in \mathbb{Z}_q$ such that $[\![x]\!] = h^{r'} g^{x'}$, which implies that even with unlimited computational power one is not able to extract any information on x from $[\![x]\!]$. The value of x is thus said to be unconditionally (or information-theoretically) hidden.

Threshold decryption is another feature which is particularly useful when constructing voting schemes. Briefly, for **threshold decryption** the private key of a public key cryptosystem is generated in such a way that each of ℓ parties obtains a share of the private key such that any t parties can cooperate to jointly decrypt a given encryption. The critical security property is that the private key is never revealed to anyone as part of the decryption protocol (nor as part of the key generation protocol). See Chapter **??** for more details on threshold cryptography, where it is shown how threshold decryption can be realized, starting from Shamir's threshold secret sharing scheme.

Voting schemes rely on threshold decryption to prevent that a single entity or single master key suffices to decrypt votes. As shown in [34], there is also an efficient way to generate the key pair for discrete log based cryptosystems such as ElGamal.

X.3.4 Verification Primitives

The category of 'verification' primitives provide some further functionality that is not covered by the primitives discussed so far. A **zero-knowledge proof** is the main example of a verification primitive, which allows a prover to convince a skeptical verifier of the validity of a given statement without revealing any information beyond the truth of the statement. The theory of zero-knowledge proofs tells us that any so-called NP-statement can actually be proved in zero-knowledge (see, e.g., [21]). We will see some concrete zero-knowledge proofs when discussing particular voting schemes later on; these proofs are generally *non-interactive*, which means that the proof is produced by the prover on its own, and may then be verified independently by multiple verifiers (much like a digitally signed message can be verified multiple times, by different verifiers).

A **designated verifier proof** is a special type of non-interactive zero-knowledge proof which is convincing only to a single verifier, specified by the proof [26]. The idea is to refer to a particular verifier V by using its public key pk_V, say, and to provide a non-interactive zero-knowledge proof for a statement of the form "I know the private key sk_V for public key pk_V or statement Φ holds". This proof will not be convincing to anyone else than verifier V, because only V is sure that the prover didn't know sk_V, hence statement Φ must actually hold. Note that this approach works only

if the verifier properly protects its private key sk_V. Designated verifier proofs are typically used to achieve some level of receipt-freeness in voting schemes (see Section X.6).

X.4 Classification of Voting Schemes

The design and analysis of electronic voting systems is a truly complex endeavor, which is approached in many different ways, revolving around many types of voting schemes. The literature on voting schemes is vast, and consists mostly of rather informal conference papers. Finding a useful, pragmatic way to group the known and yet unknown voting schemes is therefore a challenging task.

Below, we will identify a few characteristics, which are all formulated in terms of building blocks used and other properties, but without reference to which security properties hold (hence classification can be done by inspecting the description of the voting system, rather than performing a security analysis). Based on these characteristics we present a very brief taxonomy, distinguishing two major types of voting schemes.

X.4.1 Characteristics

A basic characteristic is whether the voting protocol requires *interaction between voters*. Almost every voting protocol avoids interaction, such that voters may cast their votes independently and in parallel to each other. However, see, e.g., [28] for a voting protocol in which a cryptographic counter is updated in turn by each of the voters. Interaction may also be limited to all voters belonging to a given precinct, say. In this chapter, we focus on voting protocols without interaction between voters, which is the common case.

The main technical characteristics we use are derived from the contents and properties of the voted ballot. The *voted ballot* (ballot, for short) consists of the information recorded by the voting servers (or machines), upon completion of the voting protocol by a voter. As such, the voted ballot is the interface between the voting stage and the tallying stage: it comprises all the information, which is created during the voting protocol as a function of the voters' votes, and which is needed for successful tallying after the election closes.

A major characteristic is whether the voted ballot is *anonymous vs. identifiable*. Normally, an anonymous voted ballot is submitted via an anonymous channel, preventing any link with the voter's identity. For identifiable voted ballots, there is no need for an anonymous channel. A subtle point is that even if voted ballots are anonymous, there may be further information available from the voting stage (possibly, in combination with information from other stages) that helps identifying voters. For instance, the IP address from which a vote is cast may help identifying a voter, but an IP address is in general not part of the voted ballot

Another important characteristic is whether the voted ballot contains the *vote in the clear* (derivable from public information only), *or in encrypted form*. If the vote is in the clear, then intermediate election results will be available, which is usually not allowed. Encryption of votes in some form is always needed for the voted ballots to prevent that intermediate election results can be computed. Thus, to count the votes based on the voted ballots at least one secret key should be required,

and preferably shared between several parties.

As a final characteristic, we look at *how votes (or, voted ballots) are authenticated*, and in particular if authentication is done in an identifiable way or an anonymous way. Proper authentication should ensure that each voter casts at most one vote.

X.4.2 A Brief Taxonomy

For the purpose of this chapter, we will just consider two characteristics: namely whether voted ballots are anonymous or not, and how voted ballots are authenticated. As for the other characteristics we will simply assume that the voting protocol requires no interaction between voters, and further that voted ballots are not in the clear, hence encrypted (or committed) in some way.

The main question is how authentication and encryption of voted ballots is accomplished. Given that a voted ballot contains the vote in encrypted form, we consider three basic possibilities: (i) outer authentication (a signed encrypted vote), (ii) inner authentication (an encrypted signed vote), or (iii) both outer and inner authentication (a signed encrypted signed vote). Outer authentication can be checked as part of the voting protocol (and also later), and invalid voted ballots can be discarded immediately. Inner authentication can only be checked during the tallying stage (and also later), after decryption of the votes.

The main distinction is anonymous vs. identifiable voted ballots.

Type I: Anonymous Voted Ballots

An anonymous voted ballot, as stored upon completion of the voting protocol, contains no information that helps identifying the voter. Clearly, an anonymous channel should be used for the delivery of the such voted ballots to prevent that the voting server or anyone monitoring communications may correlate voted ballots with voters or their computers.

The voted ballots should be authenticated to ensure that each eligible voter is able to cast at most one valid vote. Anonymous signatures are used for this purpose, both for outer authentication and inner authentication.

Type II: Identifiable Voted Ballots

For identifiable voted ballots, the voted ballot may be delivered in a non-anonymous way, e.g., posted to a bulletin board. For outer authentication one can use ordinary digital signatures (possible relying on a PKI). For inner authentication, if present, anonymous signatures need to be used, however, as this signature will typically be revealed in conjunction with the vote (thus the case of inner authentication only does not occur, as this corresponds to an anonymous voted ballot).

X.4.3 Examples

Well-known examples of Type I schemes are [17] and follow-up papers such as [25, 33, 32]. In these examples, voters need to get a blind signature (preferably, a threshold blind signature, where the power of the issuer is shared among a number of parties) prior to the election, which is then

used to authenticate say an encrypted vote (preferably, a threshold encryption, such that the power to decrypt is shared among a number of parties). An anonymous channel is used to sent such an anonymously authenticated encrypted votes to the voting server, which is supposed to publish these data objects on a bulletin board. Once the election closes, the votes may be counted by threshold decrypting each of the voted ballots with valid authentication.

Well-known examples of Type II schemes are the 'homomorphic' schemes of [13] and follow-up work such as [4, 39, 15, 16, 41, 2], and the 'mix-based' schemes such as [40, 31, 19]. In these schemes the voter may directly post a digitally signed encrypted vote to a bulletin board. We will consider these schemes in more detail in the next sections, emphasizing the universal (public) verifiability property of these schemes.

However, type II schemes are not necessarily universally verifiable. This depends on the details of the vote encryptions and the tally protocol. A limited level of verifiability may for instance be achieved by letting each voter encrypt its vote v together with a random bit string R of sufficient length, resulting in an encrypted vote of the form $[\![v,R]\!]$. (This idea was already used in an early voting protocol by Merritt [30].) The encrypted votes are mixed before decryption takes place, however without providing a zero-knowledge proof of validity (unlike in verifiable mixing, see below). The tallied votes are published together with the random strings, so that voters can check their votes. A major drawback of this approach is that all voters are required to actively check their votes, which may be unrealistic. Moreover, it is not clear how to resolve disputes and in particular how to maintain ballot secrecy when resolving disputes: e.g., how should a voter efficiently prove (in zero-knowledge) that its pair (v,R) is *not* present among the millions of tallied votes?

X.5 Verifiable Elections

We will now focus on two main approaches for achieving *universally* verifiable elections. We will show how electronic elections can be made universally (or, publicly) verifiable in much the same way as digital signatures are. Namely, to verify a digital signature all one needs are (a) the public key of the signer, (b) a message, and (c) a purported signature. By applying a given predicate, represented as a formula or algorithm, to these three data values, it is decided unequivocally whether the signature is valid or not (w.r.t. the given message and public key). Moreover, the verification can be repeated by anyone who is interested and as many times as deemed necessary, always giving the same result.

Similarly, to verify an election one needs (a) the joint public key of the talliers, (b) the encrypted votes, and (c) the election result. By applying a given predicate to these data values, it can now be decided unequivocally whether the election result is valid or not. Admittedly, the predicate for electronic elections is a bit more involved than for digital signatures, but the underlying principle is the same: the predicate evaluates to true (or, 'accept') if and only if the election result corresponds exactly to the votes, which are given in encrypted form only.

The digital signature analogy also works when considering the use of the private keys. To produce the election result, the talliers need to use their (shared) private key, similar to a signer using its private key to produce a digital signature. Namely, once tallying is completed, resulting in a valid election result, there is no need to repeat this process ever. As such, the purpose of a recount

is lost! However, as with digital signatures, the verification of the election result is universal, meaning that it can be performed by anyone and repeated as many times as desired.

The difficulty with verifiable elections is of course that ballot secrecy must be maintained as well—at all times. Since we are considering a Type II election, it is known which voter produced which encrypted vote (e.g., because the encrypted votes are digitally signed). The hard cryptographic problem that needs to be solved is thus formulated as follows. Suppose a threshold homomorphic cryptosystem $[\![\cdot]\!]$ (cf. Section X.3) has been set up between a plurality of parties (talliers) $P_1, \ldots, P_\ell$. We wish to design a protocol TALLY satisfying a specification of the following form:

Protocol TALLY
Input: $[\![v_1]\!], \ldots, [\![v_n]\!]$ + Validity Proofs
Output: $f(v_1, \ldots, v_n)$ + Validity Proof

Here, each $[\![v_i]\!]$ denotes a probabilistically encrypted vote v_i. The election result is defined as some function f of these votes. A common example is $f(v_1, \ldots, v_n) = \sum_{i=1}^{n} v_i$ for yes/no votes, where $v_i = 1$ means 'yes' and $v_i = 0$ means 'no'. But, function f may potentially hide even more information on the votes, e.g., $f(v_1, \ldots, v_n) \equiv \sum_{i=1}^{n} v_i > n/2$, indicating whether there is a strict majority or not (but revealing nothing else, like how small or large the majority is). A possibility at the other extreme is $f(v_1, \ldots, v_n) = (v_1, \ldots, v_n)$, where all votes are simply revealed in the same order as in the input.

Each encrypted vote $[\![v_i]\!]$ is required to be accompanied by a validity proof, which is commonly used to show that v_i is within a given range. In addition, the validity proof must provide a cryptographic link with the voter's identity, which we will denote by VID_i. This way one prevents encrypted votes are duplicated by other voters (without actually knowing the contents of the duplicated votes).

We will consider two types of tallying in more detail, called homomorphic tallying and mix-based tallying. Both approaches are built using a threshold homomorphic cryptosystem, but in different ways. The challenge in designing the protocols is to find efficient ones, and in particular, to find efficient ways of rendering universally verifiable validity proofs.

X.5.1 Homomorphic Tallying

The basic premise of homomorphic tallying is that individual votes can somehow be added in a meaningful way. Hence, the election result takes the following form:

$$f(v_1, \ldots, v_n) = \sum_{i=1}^{n} v_i.$$

The addition of votes may be defined in various ways. The case of *yes/no votes* corresponds to integer addition of votes $v_i \in \{0, 1\}$. But, for example, elections based on *approval voting* in a race with m candidates also fit this pattern: votes $v_i \in \{0, 1\}^m$ are binary m-tuples which are added componentwise. This allows each voter to indicate whether they approve of each candidate or not, and candidates with the highest numbers of vote (approvals) win. As a further example, consider *voting on a scale*, where voters express how strong they (dis)favor a given proposition by using

votes $v_i \in \{-2,-1,0,1,2\} \simeq \{--,-,\circ,+,++\}$. Using integer addition to compute $\sum_{i=1}^{n} v_i$, one can determine how much, on average, the electorate (dis)favors the proposition.

A protocol for homomorphic tallying can in general be described as follows: one simply multiplies together all encrypted votes v_i which—by the homomorphic property of the cryptosystem—results in an encryption of the sum of the votes. Protocol TALLY thus consists of the following steps:

1. Check the validity proof of every $[\![v_i]\!]$, and discard all invalid ones.
2. Multiply all the valid encryptions: $C = \prod_i [\![v_i]\!] = [\![\sum_{i=1}^{n} v_i]\!]$.
3. Jointly decrypt C to obtain $T = \sum_i v_i$ plus a validity proof.

The validity proof in the final step shows that T is indeed the plaintext contained in C, implying that $T = \sum_i v_i$. The validity proofs for the given encryptions $[\![v_i]\!]$ prevent malicious voters from entering multiple votes. For instance, in a binary (yes/no) election, voters are supposed to enter an encryption $[\![0]\!]$ or $[\![1]\!]$, but not encryptions like $[\![2]\!]$ (twice 'yes') or $[\![-7]\!]$ (seven times 'no'). As long as $0 \leq T \leq n$ the presence of such illegal encryptions may remain unnoticed. A validity proof for $[\![v_i]\!]$ shows that $v_i \in \{0,1\}$, without revealing any further information on v_i.

To verify the election result T, anyone interested executes the following steps:

1. Check that encryption C is equal to the product of all valid encryptions $[\![v_i]\!]$.
2. Check the validity proof showing that T is the threshold-decryption of C.

The validity proofs are checked against public key of the threshold homomorphic cryptosystem $[\![\cdot]\!]$.

As a concrete example we consider the simplest case of the voting scheme of [16], namely the case of yes/no votes in combination with additively homomorphic ElGamal encryption. Knowledge of the public key h, say, of the ElGamal threshold homomorphic cryptosystem, as set up by the talliers, suffices for encryption as well as validation of the votes. As explained in Section X.3, encryption of a vote $v \in \{0,1\}$ takes the following form:

$$[\![v]\!] = (A,B) = (g^r, h^r g^v), \tag{X.1}$$

for a uniformly random value $r \in \mathbb{Z}_q$.

Since such an encryption $[\![v]\!]$ can easily be formed for any value $v \in \mathbb{Z}_q$, a validity proof is required to show that either $v = 0$ or $v = 1$, without revealing which of the two cases holds. In other words, the proof must be *zero-knowledge*. Technically, the validity proof for $[\![v]\!]$ amounts to showing the existence of a $v \in \{0,1\}$ and an $r \in \mathbb{Z}_q$ such that $A = g^r$ and $B = h^r g^v$, which can be reformulated by stating that either $\log_g A = \log_h B$ holds (if $v = 0$) or $\log_g A = \log_h (B/g)$ holds (if $v = 1$).

Let $\mathscr{H}$ denote a suitable cryptographic hash function. Applying the theory of [14] to the Chaum-Pedersen protocol for proving equality of discrete logarithms [12], the non-interactive zero-knowledge validity proof is a 4-tuple (d_0, r_0, d_1, r_1), computed as follows:

$$\begin{aligned}
&d_{1-v} \in_R \mathbb{Z}_q \\
&r_{1-v} \in_R \mathbb{Z}_q, \\
&d_v = \mathscr{H}(a_0, b_0, a_1, b_1, A, B, \mathrm{VID}_i) - d_{1-v} \pmod q, \\
&r_v = w - r d_v \pmod q,
\end{aligned}$$

where

$$a_{1-v} = g^{r_{1-v}} A^{d_{1-v}}, \quad b_{1-v} = h^{r_{1-v}} (B/g^{1-v})^{d_{1-v}},$$
$$a_v = g^w, \quad b_v = h^w, \quad \text{for } w \in_R \mathbb{Z}_q.$$

A proof (d_0, r_0, d_1, r_1) is valid for encryption (A, B) and voter identity VID_i w.r.t. public key h if and only if

$$d_0 + d_1 = \mathscr{H}(g^{r_0} A^{d_0}, h^{r_0} B^{d_0}, g^{r_1} A^{d_1}, h^{r_1} (B/g)^{d_1}, A, B, \text{VID}_i) \pmod{q}. \tag{X.2}$$

Note that the order in which a_0, b_0, a_1, b_1 are computed when generating the proof is determined by the value of v, but verification of the proof is done independently of v.

In the random oracle model (which means that $\mathscr{H}$ is viewed as an ideal hash function, selected uniformly at random from all functions of the appropriate type), it can be proved that these validity proofs release no information on the vote v. The voter's identity VID_i (a bit string unique to the voter) is included as one of the inputs to $\mathscr{H}$ in order to bind the proof to a particular voter. This prevents voters from duplicating other voters' votes (copying both the encryption and the validity proof); even though voters wouldn't know the value of the votes they're duplicating, the basic requirement of independence would be violated. For the same reason, no partial election results should be available during an election (and it is usually not even allowed to announce results of exit polls during election day).

So, in principle, the talliers need to perform a single joint decryption only. For more advanced elections, such as approval voting and voting on a scale (see above), vote encryptions and the validity proofs get a bit more complicated; e.g., approval voting for a race with m candidates can be done by running m instances of a yes/no election (one per candidate) in parallel.

X.5.2 Mix-based Tallying

In *mix-based tallying* the basic idea is to mimic the use of a ballot box in traditional paper-based elections. Once the election is closed, or each time a ballot is inserted into the ballot box, the ballot box is shaken so as to remove any dependency on the order in which the ballots were actually inserted during the election.

Translated to our setting, the election result takes the following form:

$$f(v_1, \ldots, v_n) = (v_{\pi(1)}, \ldots, v_{\pi(n)}),$$

for a permutation π. To ensure ballot secrecy, permutation π will be generated uniformly at random between the talliers, by means of the following TALLY protocol:

1. Check the validity proof of every $[\![v_i]\!]$, and discard all invalid ones.

2. The talliers take turns in *verifiably mixing* the list of encrypted votes, resulting in a final list of permuted votes $[\![v'_1]\!], \ldots, [\![v'_n]\!]$.

3. Jointly decrypt every $[\![v'_i]\!]$ to obtain v'_i plus a validity proof.

Clearly, if at least one of the talliers is honest, the final list will indeed be randomly permuted. The validity proof in the first step is simply a "proof of plaintext knowledge", proving that the contents of the encryption was known to the voter upon casting the vote. This prevents vote duplication, which may be particularly harmful when mix-based tallying is used: by duplicating the encrypted vote of voter X, say, once or several times, and checking for duplicates in the output of the TALLY protocol, one may learn what X voted.

The final step is the same as for homomorphic tallying, except that decryption must be done for each encrypted vote separately. To verify the election result one performs these steps:

1. Check that the list of valid encryptions $[\![v_i]\!]$ is computed correctly.
2. Check the validity proof for each mix performed by a tallier.
3. Check the validity proof for each decrypted permuted vote.

As a concrete example we consider verifiable mixing of homomorphic ElGamal encryptions. On input of a list of encryptions $C = (c_1, \ldots, c_n) = ([\![m_1]\!], \ldots, [\![m_n]\!])$, one sets

$$d_i = [\![0]\!]\, c_{\pi(i)}, \tag{X.3}$$

$i = 1, \ldots, n$, for a random permutation π. The output is the list of encryptions $D = (d_1, \ldots, d_n)$. Here, each occurrence of $[\![0]\!] = (g^{r_i}, h^{r_i})$ denotes a probabilistic encryption of 0, using its own random value $r_i \in \mathbb{Z}_q$, and due to the homomorphic property, $[\![0]\!]c_{\pi(i)}$ and $c_{\pi(i)}$ both contain plaintext $m_{\pi(i)}$. Therefore, $[\![0]\!]c_{\pi(i)}$ is called a **random re-encryption** of $c_{\pi(i)}$. Combined with randomly permuting the encryptions it is ensured that one cannot tell which entry of the output list corresponds to which entry of the input list—of course, relying on the semantic security of the ElGamal cryptosystem. Since encryptions are never decrypted (only re-encrypted), access to the private decryption key is not required for mixing.

To make mixing verifiable, it must be proved (in zero-knowledge) that the lists C and D represent the same multiset of plaintexts. Efficient zero-knowledge proofs for this task are quite advanced. To give some intuition we first describe a simple but inefficient way (based on similarity with zero-knowledge proofs for graph isomorphism; due to Josh Benaloh). The prover generates an additional lists E computed in the same way as D was computed from C, using a fresh permutation and randomness. Then a challenge bit is given which indicates whether the prover should show how E corresponds to C *or* to D (by revealing which permutation and which randomness transforms list E into the requested one). It is not difficult to see that this way no information is leaked on how D exactly corresponds to C. However, if D is not valid mix of C, then E can correspond to C or to D but not to both at the same time! So, the probability of being caught is at least $1/2$. By repeating this protocol k times, cheating will remain undetected with probability at most 2^{-k}.

The computational complexity of this protocol is rather high, namely $O(kn)$ modular exponentiations, for n voters and security parameter k. With more advanced techniques, however, it is possible to achieve a computational complexity of $O(n)$ modular exponentiations.

Neff [31] starts by stating that the polynomial $c(x) = \prod_{i=1}^{n}(x - m_i)$ corresponding to the input list C should be identical to the polynomial $d(x) = \prod_{i=1}^{n}(x - m_{\pi(i)})$ corresponding to the output list

D. A key observation is that these polynomials are defined over the message space Z_q, where q is a very large prime (e.g., $q \approx 2^{256}$), so to prove that $c(x) = d(x)$ it suffices to evaluate the polynomial $c(x) - d(x)$ at a random value $x' \in_R \mathbb{Z}_q$: if $c(x) \neq d(x)$, the probability that $c(x') - d(x') = 0$ is bounded by n/q, which is negligibly small (as the number of roots of $c(x) - d(x)$ is at most its degree n). Neff uses this property in the design of an efficient verifiable mix, requiring $O(n)$ modular exponentiations only.

Furukawa and Sako [19] approach the problem in terms of permutation matrices. The relation (X.3) between the input and output encryptions can be written as

$$d_i = [\![0]\!] \prod_{j=1}^{n} c_j^{A_{ij}}$$

for an $n \times n$ permutation matrix A. The key property for their zero-knowledge proof is that A is a permutation matrix if and only if for all $1 \leq i, j, k \leq n$:

$$\sum_{h=1}^{n} A_{hi} A_{hj} = \delta_{ij}, \qquad \sum_{h=1}^{n} A_{hi} A_{hj} A_{h,k} = \delta_{ij} \delta_{jk},$$

where δ_{ij} denotes the Kronecker delta ($\delta_{ij} = 1$ if $i = j$, and $\delta_{ij} = 0$ otherwise). The entries of A are numbers modulo q, for a prime q, hence A is defined over a finite field. Furukawa and Sako show how to prove that A satisfies these conditions, without leaking any further information on A, and without requiring more than $O(n)$ time (counting modular exponentiations).

Follow-up papers such as by Groth [22] and by Furukawa [18] improve upon Neff's protocol and Furukawa-Sako's protocol, respectively. For instance, [18] uses the observation that if $q \not\equiv 1 \pmod 3$, then A is a permutation matrix if and only if $\sum_{h=1}^{n} A_{hi} A_{hj} A_{h,k} = \delta_{ij} \delta_{jk}$, which leads to some improvements under a mild restriction on q. All these verifiable mixes use $O(n)$ modular exponentiations and $O(nk)$ bits communication, where the hidden constant is reasonably small (say, around 5, not counting the modular exponentiations for the random re-encryptions).

The mixes described so far are known as *re-encryption mixes*. So-called *decryption* mixes have also been considered in the literature. In a decryption mix (e.g., see [18]), the mixing stage and the decryption stage are combined such that each tallier needs to be active only once: each tallier uses its share of the private key to *partially* decrypt the elements on the input list and at the same time mixes the list as before. The zero-knowledge proofs for decryption mixes are slightly more complicated than the proofs for re-encryption mixes, but decryption mixes may lead to an overall performance gain. Re-encryption mixes, however, are more generally applicable, and in the context of universally verifiable elections more flexible, as the roles of mixers and talliers can be decoupled.

For a basic decryption mix one assumes that each mix will actually contribute to the decryption. In other words, one uses (t, ℓ)-threshold decryption with $t = \ell$. To tolerate faulty or corrupt mixes, one can use (t, ℓ)-threshold decryption instead. However, the set of t mixes going to perform the decryption must be decided upon before the mixing starts; and one needs to "backup" and let all mixes do some extra work if any mix drops out.

X.5.3 Other Tallying Methods

If applicable, homomorphic tallying is very simple and efficient. However, the complexity of the validity proof for each vote $[\![v_i]\!]$ may become the bottleneck for increasingly complex votes (e.g., single transfer voting (STV) where each vote v_i is an ordered list of candidates may lead to prohibitively expensive validity proofs). For mixed-based tallying, the complexity is hardly sensitive to the range of possible values for votes v_i; here, the bottleneck is that each of the talliers must operate in turn, handling $O(n)$ encryptions.

By using techniques from secure multiparty computation, however, other methods for tallying are conceivable, which are potentially more efficient or flexible. For example, an alternative to mixed-based tallying is *sorting-based* tallying:

$$f(v_1,\ldots,v_n) = (w_1,\ldots,w_n),$$

where $w_1 \leq w_2 \leq \ldots \leq w_n$ and $\{v_1,\ldots,v_n\} = \{w_1,\ldots,w_n\}$ (as a multiset). Hence, the votes are output in sorted order, destroying any link with how the votes were entered. A protocol along these lines uses secure comparators which put two encrypted input values in sorted order (e.g., on input $[\![a]\!], [\![b]\!]$, output $[\![0]\!][\![\min(a,b)]\!], [\![0]\!][\![\max(a,b)]\!]$). Each secure comparator may be implemented as a joint protocol between the talliers, involving one or more joint decryptions (see, e.g., [20]). Sorting-based tallying is then done by arranging these secure comparators in a sorting network. Practical sorting networks such as odd-even mergesort and bitonic sort are of depth $O(\log^2 n)$, with each layer consisting of $O(n)$ comparators, where the hidden constants are small (see, e.g., [29, Section 5.3.4]).

X.6 Receipt-Freeness and Incoercibility

So far, we have considered the problem of maintaining ballot secrecy during the tallying stage, where it must be prevented that anyone is able to find out who voted what. This protects against large scale fraud, where a small number of people directly involved in running the election (insiders such as election officials, talliers, auditors, equipment manufacturers) compromise ballot secrecy of many, potentially targeted individual voters. The severity of such a large scale, centralized violation of ballot secrecy is exacerbated by the fact that it can remain completely unnoticed, without leaving any traces in the operational systems. For example, if vote encryption depends on a single (non-shared) master key, a *copy* of the master key suffices to read the contents of any encrypted vote. The existence of such copies of the master key cannot be excluded—in a verifiable way—and in fact some copies may be required as backups.

So, no one should be able to find out what voters voted, except the voters themselves! A tautology, but nevertheless a serious issue, because many remote voting schemes actually allow voters to tell others what they voted for *in a verifiable way*. This leads to problems known as **voter coercion** and **vote trading**, where voters either unwillingly or willingly give up the rights to their vote.

Whereas centralized violations of ballot secrecy by insiders, as mentioned above, may remain completely unnoticed by the voters, it is clear that voters will be completely aware of any (attempts

to) voter coercion or vote trading. This means that large scale coercion is risky as the chances of detection are high: any of the affected voters involved may leak to the press what happened. And, e.g., internet sites facilitating vote trading will be hard to hide as well.

A technical aspect is whether a voter is able to convince the coercer or vote buyer of having voted in a particular way, i.e., whether a voter is able to produce a *receipt* proving how it voted. A voting scheme is said to be **receipt-free** if voters cannot produce such receipts. Note that the ideal voting scheme mentioned in Section X.2 was indeed receipt-free as the voters tell their vote to the trusted party in private. Below we will see to what extent receipt-freeness can be achieved for cryptographic voting schemes. The property of **incoercibility** is even stronger than receipt-freeness, and is mostly beyond the scope of cryptographic techniques: e.g., a coercer may force a voter not to vote at all.

X.6.1 Coercion in paper-based elections

For postal voting it is hard to rule out that others watch someone casting a vote, willingly or unwillingly. This leads to problems such as 'family voting' where the ballots are marked by the dominating family members. However, even when people vote in person from a voting booth at a polling station, coercion is still possible as voters may produce receipts in subtle ways—of course, these days people should be asked to hand in their mobile phones, spy cams, etc., before entering the (Faraday cage like) voting booth, and this should be checked using similar equipment as for airport security.

Two well-known attacks illustrating the problem of coercion in paper-based elections are as follows. *Chain voting* is a simple attack, which makes essential use of the fact that spare ballot forms are (and should be) not readily available. To mount a chain voting attack, the coercer must first get hold of an ballot form. Next, the coercer marks this ballot form in the desired way, and hands the *pre-marked* ballot form to a voter, upon entering the polling station, and the voter is kindly asked to return with an *empty* ballot form. The empty ballot form serves a receipt, convincing the coercer that the voter put the pre-marked ballot form in the ballot box, and may be used to repeat the attack.

A so-called *Italian attack* is applicable when ballot forms allow voters to cast their votes each in a unique way. Sometimes this may be done by marking the paper ballot in a particular way, using recognizable marks. A more subtle way, which also works for electronic voting, applies when voters are supposed to rank the candidates (by listing them in the preferred order on the ballot form), such as in STV ('Single Transfer Voting') elections. In that case, the coercer will demand the intended winner to be put at the top followed by the remaining candidates in an order unique for the targeted voter. Later, the coercer will check if the requested ballot appears on the list of voted ballots.

X.6.2 Receipt-Freeness for Encrypted Votes

The secrecy of a probabilistic public key encryption actually depends on two factors: the secrecy of the private decryption key held by the receiver *and* the secrecy of the randomness used by the

sender. Hence the necessity of secure (pseudo)random generators both during key generation *and* during encryption. But what if the sender willingly discloses the randomness?

For instance, if ElGamal encryption is used to encrypt a vote v as $[\![v]\!] = (A,B) = (g^r, h^r g^v)$, then revealing the random value r to a coercer proves that the vote contained in $[\![v]\!]$ is equal to v. Indeed, the coercer first checks that $A = g^r$ holds, and if so, checks that $b = h^r g^v$ holds as well. In other words, the value of r serves as a receipt. Clearly, this is true for probabilistic public key cryptosystems in general: the randomness used to compute an encryption can be used as a receipt.

Obviously, for revealing r to the coercer the voter needs to have access to this value. If a software voting client is used, running on a platform controlled by the voter, such as a PC or a smart phone, it will be quite easy to extract the value of r. In fact, since the voting protocol is known (and preferably based on an openly available specification), the voter (or the coercer, for that matter) may program its own voting client with the additional feature of copying the value of r to the output. Hence, a way to make the voting protocol receipt-free is to execute it on a platform beyond the control of the voter, such as a smart card or a direct-recording electronic (DRE) voting machine, with the immediate drawback that the voter is not ensured anymore that its vote is recorded as intended and the additional problem that the voter must trust the smart card or the voting machine with its vote.

A more refined approach is to use what it is known as a **randomizer**, an idea reminiscent of the use of the *wallet-with-observer* paradigm due to Chaum (see Chapter **??**). A randomizer, which may be implemented as a smart card or as a part attached to a voting machine, runs a protocol with a voter to jointly produce an encrypted vote together with a validity proof. To this end, the voter and the randomizer communicate with each other over an untappable channel.

The randomizer will generate part of the randomness used to encrypt the vote and to generate the validity proof. The basic idea is to let the randomizer produce a random re-encryption of the voter's encryption, and modify the validity proof (as generated by the voter) accordingly. So, the voter starts out by sending a homomorphic encryption $[\![v]\!]$ of its vote v to the randomizer, over the untappable channel. The randomizer will then reply with a re-encryption of the from $[\![0]\!][\![v]\!]$, accompanied by a designated verifier proof for the validity of the re-encryption. This proof is only meaningful to the voter, as it is bound to the public key of the voter. The tricky part is how to divert the validity proof of the voter for $[\![v]\!]$ to a validity proof for $[\![0]\!][\![v]\!]$. This is done by first jointly generating a proof for $[\![v]\!]$ and then adjusting it to a proof for $[\![0]\!][\![v]\!]$.

As an example we apply Hirt's technique [23, Chapter 5] to the yes/no voting protocol described in the previous section. The *diverted* proof looks as follows. Let $[\![v]\!] = (A,B) = (g^r, h^r g^v)$ denote the voter's encryption of vote v. The randomizer re-encrypts this to $(A'',B'') = (g^{r'}, h^{r'})(A,B)$, where $r' \in_R Z_q$. As before, the voter will contribute a proof (d_0, r_0, d_1, r_1) for (A,B) satisfying (X.2), except that this time the challenge hash is applied to appropriately 'blinded' versions of (a_0, b_0, a_1, b_1). This results in a proof $(d_0'', r_0'', d_1'', r_1'')$ for the validity of encryption (A'',B''), satisfying (X.2). The contribution of the randomizer to the proof consists of $d_0', d_1', r_0', r_1' \in \mathbb{Z}_q$ subject to $d_0' + d_1' = 0$, such that

$$\begin{aligned} d_0'' &= d_0 + d_0', \qquad & d_1'' &= d_1 + d_1' \\ r_0'' &= r_0 + r_0' - r' d_0'', \qquad & r_1'' &= r_1 + r_1' - r' d_1'' \end{aligned}$$

Note that the randomizer does not learn what vote v is. And, the voter does not know how to open (A'', B''), as the value of r' remains hidden to the voter. And, even if the coercer forces the voter to compute (A, B) and the proof (d_0, r_0, d_1, r_1) in a particular way, it's still possible to simulate the protocol transcript where the voter actually uses a different vote.

A randomizer may be implemented as a tamper-resistant smart card. The voter's software client will communicate with the smart card privately. Voters are required to use the randomizer, which can be ensured by a digital signature of the randomizer on the re-encrypted vote. The smart cards should be beyond the reach of the coercers, and one needs to assume that the smart cards use good randomness. If the voter has a choice between several randomizers, all of these must be non-coerced, otherwise the coercer will force the voter to use a coerced one.

The use of randomizers can be seen as an add-on solution. One may decide per election, and also per category of voters, whether and how many randomizers are required. As large scale vote trading or coercion may be hard to hide anyway, one needs to weigh the use of cryptographic measures such as randomizers against other measures such as severe punishments.

X.7 Untrusted Voting Clients

A basic assumption in the voting schemes described above is that each voter has a PC, a mobile phone, or any other computer device at its disposal, which can be used as a voting client. The voting protocol is assumed to be implemented on such a device, and, given an open specification of the voting protocol, voters may in principle build their own implementations (or, pick one from many vendors who comply with the open specification).

In many elections, such as traditional elections from polling stations, there is no possibility for voters to use their own computer devices and software. Instead, one needs to trust equipment provided at a polling station, running software which cannot be monitored. This raises the question whether, e.g., verifiable elections are still possible in such a setting.

Similarly, for paper-based elections verifiability is limited: election observers must continuously monitor all proceedings and trace exactly what happens to each of the paper ballots. There is no reliable way for checking whether one's voted ballot is actually included in the final tally. Also, using numbered ballot forms and giving each voter a copy of the marked ballot form may violate ballot secrecy and is clearly not receipt-free. More advanced techniques are needed to ensure in a verifiable way that each and every legitimately cast paper ballot—and no other ones—will be tabulated.

Much recent work has thus been focused on addressing the issue of ensuring that votes are "cast as intended", complementing the work on verifiable tally protocols which ensures that votes are "counted as cast". Work in this direction was initiated by Chaum (see, e.g., [10] and references therein) and by Neff (see, e.g., [1] and references therein, incl. "Practical High Certainty Intent Verification for Encrypted Votes" by Neff, 2004).

To illustrate the ideas behind these new approaches we briefly describe the common idea behind a range of proposals by Ryan *et al.* (see, e.g., [37] and references therein). Consider an election in which voters must pick one candidate from a list of candidates. The idea is that each paper ballot is perforated vertically with the left-hand side containing the names of the candidates in a

random order and the right-hand side containing an empty box for each candidate. To cast a vote, the voter will (i) mark the box of the candidate of its choice, (ii) tear the ballot form into two parts, (iii) destroy the left-hand half, and (iv) drop the right-hand half into a ballot box—possibly after receiving a copy of this part. Since the candidates were listed in a random order on the left-hand half, the marked right-hand half does not reveal which candidate was chosen.

The ballot forms are numbered, though, in such a way that each right-hand half can be linked to an encryption of information indicating how the marked right-hand half should be interpreted. Cryptographically, the technique is similar to the use of "masked" ballots as introduced in [39], and used in [16] as well for yes/no votes. Similar to (X.1), one computes an encryption of the form:

$$[\![b]\!] = (g^r, h^r g^b), \tag{X.4}$$

for a uniformly random value $r \in \mathbb{Z}_q$, and a uniformly random bit $b \in_R \{0,1\}$. Depending on the value of b, the form will list the options in the order "no-yes" or "yes-no". Together with the position of the mark on the marked ballot, encryption $[\![b]\!]$ is then transformed into an encryption $[\![v]\!]$ of the intended vote v. The encrypted votes $[\![v]\!]$ may then be tallied in a verifiable way, as described above.

The production of the ballot forms and the associated encryptions forms a critical part of these voting systems. Encryptions such as (X.4) must be produced in a private yet secure way such that they correspond to the printed ballot forms, but without revealing the contents of these encryptions. Thus, this problem is best viewed as a (non-trivial) problem in secure multiparty computation as well. The challenge is achieve efficient solutions for these problems.

These new approaches are still under scrutiny and several issues need to be addressed. For instance, even though a receipt (e.g., a copy of the marked right-half) does not reveal the vote, such a receipt nevertheless opens a possibility for coercion because of the so-called *randomization attacks* due to Schoenmakers who noted this issue first for the voting scheme of [24] (see also [27]). In a randomization attack, a coercer will simply specify for each voter to mark a particular box on the right-hand half—regardless of which candidate this box corresponds to. The coercer can easily check whether a voter complied to the specified rule (e.g., "mark the box at the bottom of the form"). Since the candidates are listed in a random order, different on each ballot form, the coercer thus forces a voter to effectively cast a random vote, which may have a deciding influence on an election.

X.8 Defining Terms

Anonymous channel: A communication channel which hides the identity of the sender, and possibly allows for an acknowledgement by the receiver as well.

Blind signature: A signature scheme where the signer cannot recognize the message-signature pairs resulting from the signing protocol.

Bulletin Board: A publicly readable broadcast channel, possibly with authenticated write operations.

Computational security: If a security property holds subject to a computational intractability assumption.

Designated verifier proof: A noninteractive zero-knowledge proof bound to a particular public key which only convinces the holder of the corresponding private key.

Homomorphic encryption: A public key cryptosystem in which the 'product' of two encryptions is equal to an encryption of the 'sum' of the corresponding plaintexts.

Information theoretic security: If a security property holds regardless of the attacker's computational power.

Private channel: A communication channel protected against eavesdropping, and possibly providing end-to-end authentication as well, achieving information-theoretic security.

Secure channel: A communication channel protected against eavesdropping, and possibly providing end-to-end authentication as well, achieving computational security.

Threshold decryption: A public key cryptosystem in which decryption requires the participation of a certain number of parties, each holding a share of the private key.

Untappable channel: A totally unobservable (out-of-band) communication channel.

Validity proof: A zero-knowledge proof showing the validity of an input, or step performed in a protocol.

Zero-knowledge proof: A proof for a statement without giving away why it is true.

Bibliography

[1] Ben Adida and C. Andrew Neff. Ballot casting assurance. In *Proceedings of the USENIX/Accurate Electronic Voting Technology (EVT) Workshop 2006*, 2006.

[2] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard. Practical multi-candidate election system. In *Proc. 20th ACM Symposium on Principles of Distributed Computing (PODC '01)*, pages 274–283, New York, 2001. A.C.M.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th Symposium on Theory of Computing (STOC '88)*, pages 1–10, New York, 1988. A.C.M.

[4] J. Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, Yale University, Department of Computer Science Department, New Haven, CT, September 1987.

[5] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *Proc. 26th Symposium on Theory of Computing (STOC '94)*, pages 544–553, New York, 1994. A.C.M.

[6] S. Brands. Untraceable off-line cash in wallet with observers. In *Advances in Cryptology—CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318, Berlin, 1994. Springer-Verlag.

[7] S. Canard, B. Schoenmakers, M. Stam, and J. Traoré. List signature schemes. *Discrete Applied Mathematics*, 154:189–201, 2006. Special issue on Coding and Cryptology.

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

[9] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R.L. Rivest, and A.T. Sherman, editors, *Advances in Cryptology—CRYPTO '82*, pages 199–203, New York, 1983. Plenum Press.

[10] D. Chaum. Secret ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, Jan.-Feb. 2004.

[11] D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology—EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265, Berlin, 1991. Springer-Verlag.

[12] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology—CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105, Berlin, 1993. Springer-Verlag.

[13] J. Cohen and M. Fischer. A robust and verifiable cryptographically secure election scheme. In *Proc. 26th IEEE Symposium on Foundations of Computer Science (FOCS '85)*, pages 372–382. IEEE Computer Society, 1985.

[14] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187, Berlin, 1994. Springer-Verlag.

[15] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret ballot elections with linear work. In *Advances in Cryptology—EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83, Berlin, 1996. Springer-Verlag.

[16] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology—EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118, Berlin, 1997. Springer-Verlag.

[17] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology—AUSCRYPT '92*, volume 718 of *Lecture Notes in Computer Science*, pages 244–251, Berlin, 1992. Springer-Verlag.

[18] J. Furukawa. Efficient, verifiable shuffle decryption and its requirement of unlinkability. In *Public Key Cryptography—PKC '04*, volume 2947 of *Lecture Notes in Computer Science*, pages 319–332, Berlin, 2004. Springer-Verlag.

[19] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *Advances in Cryptology—CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387, Berlin, 2001. Springer-Verlag.

[20] J. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *Public Key Cryptography—PKC '07*, volume 4450 of *Lecture Notes in Computer Science*, pages 330–342, Berlin, 2007. Springer-Verlag.

[21] O. Goldreich. *Foundations of Cryptography - Basic Tools*. Cambridge University Press, 2001.

[22] J. Groth. A verifable secret shuffle of homomorphic encryptions. In *Public Key Cryptography—PKC '03*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160, Berlin, 2003. Springer-Verlag.

[23] M. Hirt. *Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting*. PhD thesis, ETH ZURICH, 2001.

[24] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology—EUROCRYPT '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 539–556, Berlin, 2000. Springer-Verlag.

[25] Patrick Horster, Markus Michels, and Holger Petersen. Blind multisignature schemes and their relevance for electronic voting. In *Proc. of 11th annual computer security applications conf.*, pages 149–155. IEEE Computer Society, 1995.

[26] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In *Advances in Cryptology—EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 143–154, Berlin, 1996. Springer-Verlag.

[27] A. Juels, D. Catalano, and M. Jakobsson. Coercion resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society (WPES 2005)*, pages 61–70. ACM, 2005.

[28] J. Katz, S. Myers, and R. Ostrovsky. Cryptographic counters and applications to electronic voting. In *Advances in Cryptology—EUROCRYPT '01*, volume 2045 of *Lecture Notes in Computer Science*, pages 78–92, Berlin, 2001. Springer-Verlag.

[29] D. E. Knuth. *The Art of Computer Programming (Vol. 3: Sorting and Searching)*. Addison Wesley, Reading (MA), 2nd edition, 1998.

[30] M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, February 1983.

[31] C.A. Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security 2001*, pages 116–125, New York, 2001. ACM Press.

[32] Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka, and Tatsuaki Okamoto. An improvement on a practical secret voting scheme. In M. Mambo and Y. Zheng, editors, *ISW '99*, volume 1729 of *Lecture Notes in Computer Science*, pages 225–234, Berlin, 1999. Springer-Verlag.

[33] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security protocols. 5th int. workshop. proc*, pages 25–35, Berlin, 1997. Springer-Verlag.

[34] T. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology—EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526, Berlin, 1991. Springer-Verlag.

[35] T. P. Pedersen. *Distributed Provers and Verifiable Secret Sharing Based on the Discrete Logarithm Problem*. PhD thesis, Aarhus University, Computer Science Department, Aarhus, Denmark, March 1992.

[36] J.M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[37] B. Randell and P.Y.A. Ryan. Voting technologies and trust. *IEEE Security & Privacy*, 4(5):50–56, Sept.-Oct. 2006.

[38] R. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Advances in Cryptology—ASIACRYPT '01*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565, Berlin, 2001. Springer-Verlag.

[39] K. Sako and J. Kilian. Secure voting using partially compatible homomorphisms. In *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 411–424, Berlin, 1994. Springer-Verlag.

[40] K. Sako and J. Kilian. Receipt-free mix-type voting scheme—a practical solution to the implementation of a voting booth. In *Advances in Cryptology—EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403, Berlin, 1995. Springer-Verlag.

[41] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 148–164, Berlin, 1999. Springer-Verlag.

Further Information

As described in the introduction, research in voting schemes developed in three waves, starting in the 1980s, 1990s, and 2000s, respectively. The first wave started from within the emerging cryptographic community, and many research papers on electronic voting can thus be found in the cryptographic literature, which is described extensively in Chapters **??**–**??**. Information on practical deployment of electronic voting schemes can be found on the Internet since the mid 1990s when the first experiments started. Interest in both theoretical and practical aspects of electronic voting has been growing ever since, and especially since 'Florida 2000' there is also a lot of political and societal interest as well.

Research in electronic voting is shaping up, witnessed by a flurry of workshops devoted to the topic. Researchers of various backgrounds now meet on a regular basis. Examples of workshops are those under the flag of the IAVOSS (see `www.iavoss.org`) including WOTE and other workshops, the EVT workshops co-located with the USENIX Security Symposium, and the EVOTE workshops organized by `e-voting.cc`.

Traditionally, many countries and organizations work on their own voting systems and implementations, and the business has been limited to a relatively small number of companies. The market is opening up though. A major impetus to this process will be the development of open standards capturing the main voting schemes such as described in this chapter. In particular, when focusing on Type II schemes with identifiable voted ballots, a lot of flexibility regarding the network delivery of the voted ballots (e.g., by email, by https, over a cable TV network, ...) and regarding the outer authentication of the voted ballots will be possible, whereas the encryption of the votes will be prescribed by the standards. Implementations of the key generation, voting, tally, and verification protocols adhering to these standards may then be offered by a multitude of suppliers, incl. non-commercial ones.

Even though electronic voting will remain controversial for years to come—and remote voting even more so—it's the author's opinion that inevitably remote electronic voting will start to be used

for major elections in the coming decades. With the advance of communication and computation technology, instant large scale elections in which everyone votes online within a time frame of one hour (or one minute?) will be possible. And, how else would one vote in virtual communities, or in games like Second Life? Future generations may have an innate distrust of paper-based elections, once paper becomes something archaic, used only by magicians, or other tricksters.

Read chapter 1 - An Artist and a Writer of novel DRAFT for free, written by InZanity in Webnovel, total Chapters: 18.Â Chapter 1: An Artist and a Writer. It was a random afternoon when the Writer came across her flatmate, busy with her laptop and digital tablet. As she looked at what the Artist was doing, she couldn't comprehend what was being drawn. Draft book chapters. Contribute to fonnesbeck/python_and_r development by creating an account on GitHub.Â GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together. Sign up. Draft book chapters. 1 commit. 1 branch. [[EDIT: I'll be writing over this draft by 2019 and fleshing out this book as a part of a TRIAL run. You will actually see how I go about making my fanfics so you may see the words alter from time to time.]] Chapter 1: Towards a Singularity. Chapter Text. //Placeholder. //Scattered thoughts will be outlined in the following chapters. //NOTES: [to understand what I am thinking]. // - commentary and random side/mental notes - usually my thinking out loud moments. I once wrote the first draft of a book chapter that smelt so bad, I had to open up the office window while reading it. Itâ€™s a good thing my first drafts are for me alone, and yours should be too. Contents [hide]. How to Write a First Draft: Work on It Every Dayâ€¦ Until Itâ€™s Done. How Do You Start a Rough First Draft? Writing Your First Draft Tips. â€˜I Still Canâ€™t Get Started!â€™ What to Expect From Your First Draft. Some book authors like to end each chapter with a cliffhanger, whether thatâ€™s an unresolved conflict between characters, a new crucial piece of information, or an actual cliff. Anything to keep the reader engaged in what comes next. Approach each chapter with a specific goal. One chapter might be focused on a chase scene.Â Donâ€™t worry about clear chapter breaks in your first draftâ€”unless it helps you stay organized. Itâ€™s best not to get too attached to the order or definitive direction of the elements of your story at first.