Tel Aviv University
Raymond and Beverly Sackler
Faculty of Exact Sciences
School of Computer Sciences

Bare-Handed Electronic Voting with Pre-processing

Submitted as a partial fulfillment of the
requirements towards the Master of Science degree by

Ben Riva

The research work has been conducted
under the supervision of

Dr. Amnon Ta-Shma

January 2008

Abstract

Many electronic voting protocols assume the voter votes with some computing device. This raises the question whether a voter can trust the device he is using. Three years ago, Chaum, and independently Neff, proposed what we call *bare-handed* electronic voting, where voters do not need any computational power in the voting booth. Their protocols have a strong unforgeability guarantee. The price for that, however, is that they require the voter to tell his vote to the voting booth.

In this work we propose a protocol where the voter votes bare-handedly, and still maintains his privacy even with respect to the voting booth. We do this by allowing the voter the use of a computer device but only at a pre-processing stage - the voting itself is done bare-handedly. This has many advantages. A voter who has to verify calculations at the booth has to trust the software he is using, while a voter who verifies pre-processed calculations can do that at his own time, getting help from whatever parties he trusts.

Achieving private, coercion-resistant, bare-handed voting with pre-processing is a non-trivial task and we achieve that only for elections with a *bounded* number of candidates. Our solution works by proposing an extension to known voting protocols. We show that such extended protocols enjoy the same unforgeability guarantee as that of Chaum and Neff. In addition, our extended protocol is private, and the voter does not reveal his vote to the booth.

Acknowledgments

..

Contents

Chapter 1

Introduction

Two centuries ago, Thomas Alva Edison received his first patent on an Electric Vote Recorder.[1] There are several basic properties required from such a system. Above all, a voting protocol has to be *unforgeable*, i.e., even a coalition of (computationally unbounded) adversaries can not forge the voting results. Also, a voting protocol has to be *private*, meaning that an adversary can not learn how a specific voter voted. Another more subtle property is that of *coercion-resistance* which basically means that a voter can deny his vote.[2] Finally, we would like the system to be *auditable* (also called *verifiable*), meaning that all actions taken during the elections are written down on a public board open for inspection and verification by *everyone*.

In the last years, there are two growing directions for achieving such a system: using cryptographic systems and using non-cryptographic systems. The first, is the subject of this work and we will return to it soon. The second, the non-cryptographic systems, are used worldwide in many large scale elections [Wik07b], usually using the Direct Recording Electronic (DRE) voting machines. Those machines basically consist of a generic computer and a display. Such a machine is placed in an isolated voting booth and when a voter enters the booth he can choose his vote using a touch-screen or a keyboard. Then, the machine records the vote (in a memory card for instance) and after the elections are over all the votes are summed and the results are

[1]He, however, failed selling this patent [Tho].
[2]We explain the different variants of this property later.

published. On the good side, those machines improve the vote counting speed and accuracy, they are more voter-friendly and even more accessible (they can provide large touch-screen and headphones for handicapped people, or, they can support many languages) and they spare the troubles with paper ballots (print them, save them in a safe place and keep them unspoiled). On the bad side, they are poorly auditable and suspect to many attacks. Even though, those machines have already served more then a quarter of the registered voters in the United States. Another system worth to mention is the recent ThreeBallot voting system [RS07] which uses paper ballots and a scanner, but gives better auditability and privacy then the simple paper ballot elections. It does so without cryptography.

Lets return to the other direction, cryptographic based voting systems. At first, we note that such a system is a special case of the much more general problem of *Incoercible Secure Multiparty Computation* [CG96] where a set of players jointly compute an *arbitrary function* of their inputs, without revealing information about their inputs. Furthermore, they can deny their inputs afterwards (or more correctly, they can lie to a coercer about their inputs). Even though this computation has a polynomial solution, this solution is still unsatisfying against a coercer which can coerce a voter *before* the elections.

There are many proposals for cryptographic electronic voting protocols and we describe a few of them in Section 4.[3] Many of those protocols require that the voter uses computational power in the booth, and the underlying assumption is that honest voters can control the algorithm they run. However, we should question this assumption. Viruses and mal-wares are common reality today. How can one be sure that the algorithm one runs is indeed the intended one? Unforgeability is guaranteed only if honest voters properly carry out their computations, and so, viruses and mal-wares could largely affect the results of elections.

This leads to a series of questions. Can we check the device we are running? Can we trust the algorithm that checks our device? Can we check our vote after the fact to see whether it was maliciously altered, and if so can we prove it was altered and

[3]From here we use the shorter term *electronic voting* for referring to *cryptographic electronic voting*.

how do we fix the situation?

This led David Chaum [Cha04], and independently Andrew Neff [Nef04], to suggest the notion of what we term as *bare-handed voting*. The idea is that the voter comes to the voting booth without any computational power and manually verifies that his vote is properly processed (e.g., using his eyes and visual cryptography in Chaum's scheme). A very appealing aspect of this approach is that the system is auditable. The auditors (and anyone can be an auditor) can verify the validity of the votes and the elections in real time. Instead of verifying the election machines back-to-front (reviewing the source code, checking the hardware) only the machine's computation results are being audited. As a result the system is truly unforgeable.

One way to view Chaum's and Neff's algorithms is that the voter delegates his computations to the voting booth, and his only role is to check his vote is correctly registered. The price of this approach is that the voting booth knows what each voter voted. Thus, in terms of privacy, the system is unsatisfactory. For example, the government can easily find out what each citizen voted.

Recent protocols (e.g., [CRS05, LTR$^+$06b, AR06, Cha07]) use paper based voting where the paper ballots can be prepared in advance by one or more authorities. For example, in [Cha07] one authority prepares the ballots, in [CRS05] one authority prepares the ballots, but the ballots are encrypted with a cascade mixing using the public keys of several parties, and in [LTR$^+$06b] the encryption is distributed.

It is important to understand that there is no privacy towards the party (parties) that prepare the ballots. The above protocols basically transfer the point of failure. In [Cha04] we trust the booth and in the above protocols we have to trust the party that prepares the ballots. For example, if the government prepares the ballot then there is no privacy towards the government.

Moreover, even if we trust the parties who prepare (and encrypt) the ballots, there is still a severe privacy problem with many protocols (e.g., [CRS05, AR06, Cha07]). Suppose some party A can watch the encrypted ballots before they are being used. Then, that party knows the matching between candidates and encryptions (that appears on the ballots). After a ballot is used, the published information on the public board contains the voter's name and an encrypted value, and therefore party

A knows exactly what the voter voted. In other words, the ballots should be kept guarded until they are used.

Thus, current protocols we are aware of, either require some computational power from the voter at the booth, and then in return give the voter full privacy, or do not require computational power from the voter at the booth, but as a result the voter loses his privacy against the party that prepared the ballot. In this work we show how to maintain privacy (even against the government) without requiring the voter to have computational power at the booth. We do that by letting the voter prepare the ballot himself. This raises several problems which we will discuss next.

1.1 Contributions of this work

In this work we consider bare-handed voting *with pre-processing*. In our model, voters need computational power but only at a pre-processing stage. They later on come to the voting booth (with the pre-processed paper ballots) and vote bare-handedly. The pre-processing stage in our approach resembles preparing paper ballots in current manual elections. Any voter can prepare any number of pre-processed ballots in the pre-processing stage. He can also choose to test the ballots or any (random) subset of them. Subsequently, the voter comes equipped with the pre-prepared ballots to the voting booth and manually votes. In the booth we require only simple human abilities such as: reading and the ability to compare strings.

In our protocol the voter prepares the ballots at home. This has a privacy advantage, but potentially makes the protocol coercible. Nevertheless, our protocol supplies a strong guarantee against coercion. We assume a powerful coercer that can give coerced ballots to the voter, and make sure the voter has no other ballots with him. We show that if a coercer can coerce a voter, then the coercion is detected with a good probability. This, in particular, implies that a coercer can not coerce many people to vote without being detected. We describe how this is done in Section 5.1.

One might ask why the use of a computer outside the booth is safer then the use of a computer device inside the booth (as in the protocol of [BFP$^+$01]). One reason is that the voter has no way to check how his device functions inside the booth. Moreover, he can be coerced to use a malicious device. In contrast, a voter has a choice how to prepare his pre-processed ballots: he can download an open-source software, program such a software by himself or use a public web-site for that. Furthermore, he can create as many ballots as he wishes, and therefore he can choose a subset of the created ballots and check their validity. Furthermore, as our protocol is coercion-resistant, the voter can get his ballots from a coercer or from a public machine, as long as he believes that the booth does not collude with that machine.

1.2 Our technique

At this stage one should wonder how difficult it is to transform a bare-handed protocol where the computation is delegated to the booth, to a bare-handed protocol with pre-processing where no computation is delegated to the booth. Indeed, this is the subject of our work. It turns out that this transformation is not easy, and we can achieve it only for the case where the number of candidates is bounded.

Let us first describe in a schematic level how previous, bare-handed voting protocols work. The voter tells the booth his vote. The booth publishes an encryption of the vote. The encryption is necessary because we want to maintain privacy towards the rest of the world. As the booth publishes an encrypted message that others can not read, the booth also has to prove the validity of the vote, and so the booth also prepares a non-interactive, zero-knowledge proof of validity and publishes it. We want the system to be auditable, and so, the encrypted vote and the proof are publicly published. We now have an immediate concern: how does the voter know that the booth properly encrypted his vote without changing it. Thus, a cut-and-choose technique is used in which one copy is used for voting, and the other for testing the booth. The voter chooses randomly which copy to test. We now stand before the dilemma who actually does this testing? The voter can not do that at the booth because he is bare-handed. We could hope to delegate it to the auditors, but this seems to imply that the auditors know both the original vote and its encryption and we are back to where we have started.

Indeed, Chaum deals with this problem by using visual cryptography, thus letting the voter do a complicated task visually without a computer. The later protocols [CRS05, AR06] use a different approach. One way of looking at what they do is that essentially they delegate the testing to the auditors, but instead of testing the booth just on the candidate the voter picked, they test the booth on all candidates. As a result, they simplify Chaum's early ideas and get rid of the visual cryptography component. On the other hand, this has the cost of losing the ability of dealing with an unbounded number of candidates (or write-in ballots).

We now want to see whether these ideas can be extended to allow privacy even towards the booth. First, as we have to preserve privacy towards the booth, we let the voter encrypt his vote, and because we do not allow the voter computational power at the booth, the voter does that before he comes to the booth. The voter gives the booth an encrypted message that the booth can not read, and so the voter also prepares a non-interactive, zero-knowledge proof of validity before he comes to vote. Before we go on we record one problematic issue here, which is that since the voter comes to the booth with a pre-prepared ballot, we have to make sure he is not coerced. In particular, we need to keep in mind the scenario where the ballot the voter uses was prepared by a coercer.

Nevertheless, we go on with our attempt. We want the system to be auditable, and so the vote and the proof have to be publicly published. This raises the threat of coercion-resistance: the voter might be able to prove his vote by opening the vote or the validity proof. We deal with that by asking the booth to re-encrypt the vote randomly. Now, we have to worry about the booth changing the vote, and so we again use the cut-and-choose technique in which one copy is used for voting and the other for testing the booth, and we stand again before the dilemma of who is going to check the testing copy.

Our solution to those problems is surprisingly simple. It follows the spirit of Adida and Rivest protocol [AR06]. We also deal only with the case of a bounded number of candidates. We ask the voter to come to the booth with a pre-prepared ballot (both an encryption of a vote and a proof of validity) for *every possible candidate*. The booth then checks that the voter knows how to associate a candidate with the encrypted vote, and he does so using a cut-and-choose technique. At the end of this process we know that the voter knows a vote for each candidate, and at least in theory can choose the candidate he actually wants. In a way, this is similar to how privacy is ensured in paper-ballot elections when the voter enters a booth with a ballot for each candidate.

We then use ElGamal homomorphic re-encryptions for coercion-resistance. We use a cut-and-choose technique for testing the booth without revealing the way re-encryptions are done. The testing is done with respect to all candidates and the actual verification of the test is done by the auditors. Finally, we use existing voting protocols (which are not bare-handed), like Sako and Kilian [SK95] and Cramer, Gennaro and Schoenmakers [CGS97] for the tallying phase.

We also mention that in previous work these techniques could either solve active attacks, or give coercion-resistance, but could not simultaneously give both properties (we explain what active attacks are at the end of Section 4.2.2 and our solution in Section 5.3). Our protocol provides both.

1.3 Thesis outline

In Chapter 2 we describe the problem (the participants and the attack model) with somewhat more detail. In Chapter 3 we review some cryptographic tools that we will later use and in Chapter 4 we sketch previous work, both for protocols which require a computer assistance and for protocols which do not. In Chapter 5 we describe our bare-handed extension, implementing the above general ideas. Last, in Chapter 6 we conclude our work.

Chapter 2

Participants, Required Properties and Attack Model

We have voters, voting booths, trustees and auditors. As with many other protocols we have a *public board* which is a reliable database accessible by *everyone*. The auditors have access only to this public board and constantly check its integrity (data is only added to the database, old data does not change, everyone gets to see the same picture) and its contents (proofs are correct etc.). Everyone can be an auditor. One may think of this public board as an Internet site where all data is accumulated, and where its reliability stems from the fact that it is under constant public inspection. The assumption that such a public board can be maintained is made in many previous works (e.g., [HS00, Cha04]).

Some very basic requirements from an electronic voting protocol (stated in a very informal way) are:

Unforgeability - No one can falsify the result of the voting.

Eligibility, Unreusability - Respectively requires that only eligible voters vote and no voter can vote twice.

Auditability, Universal auditability - The first describes the ability of any individual voter to determine whether or not his vote has been correctly placed.[1] The second corresponds to the ability of any auditor to determine that the whole

[1]Also called Voter-verifiability.

protocol was followed correctly, given that votes had been correctly placed.

Robustness - Dishonest participants can not disrupt the voting. In particular cheating players should be detected and it should be possible to prove their malicious behavior and finish the voting process and the counting without their help.

Privacy - No one can link a voter with his vote.

Receipt-freeness, Coercion-resistance - The notion of receipt-freeness was introduced by Benaloh and Tuinstra [BT94], and it means that the voter can not prove to which candidate he voted. This notion can be generalized in several ways. The strongest one, usually called *coercion-resistance*, avoids even scenarios where the voter cooperates with the coercer, and they both try to find a strategy where the voter can prove that he followed the coercer instructions (e.g., they can choose specific private keys and a strategy such that the voter can prove that he voted a specific value or a random value). A formal definition was given in Juels, Catalano and Jakobsson [JCJ05].

For unforgeability, auditability and universal auditability, we assume the malicious party includes any subset of malicious voters, the voting booth and all of the trustees. We assume the malicious party is computationally unbounded. The requirement is that if the malicious party changed the votes of t honest voters then it would be caught cheating with probability at least $1 - 2^{-\Omega(t)}$. If this property holds only for computationally bounded adversary we say we have *computational* unforgeability/auditability/universal auditability. We believe unforgeability is the most sensitive property of a voting system. Losing privacy is bad, but losing vote credibility is disastrous. Hence, we believe having a stringent unforgeability definition requiring defence against an all-powerful, extensive coalition of malicious players is appropriate.

There are many ways to define privacy. The most appropriate one is probably saying that the information the adversary holds is computationally close to a distribution that has very low mutual information with the actual mapping between voters and votes, and this should hold even if there is some a-priori knowledge on voting patterns. Such a definition protects not only individuals but also groups of persons (e.g., it will not leak information about the way a certain minority group voted). In any case, we inherit the privacy guarantee that we get from the underlying protocol that we use. For privacy, we restrict ourselves to computationally bounded adversaries. We allow the adversary to consist of a coalition of the voters, the booth and some of the trustees (the exact number of trustees depends on the underlying protocol).

Finally, for coercion-resistance, we restrict the adversary to be computationally bounded. We allow a coalition of malicious voters, the coercer and some trustees (again, depending on the underlying protocol). Here we make the essential assumption that the booth does not cooperate with this attack (in manual elections there are voting booths that physically isolate the voter for coercion-resistance. The same is true for electronic elections as well. All the protocols that we are aware of guarantee privacy and coercion-resistance assuming some trust in the system and usually assuming the coercer does not collude with the booth). We also need to use what we call a *recordable, private channel* between the voter and the booth. A recordable, private channel between two parties A and B is an untappable channel between A and B that has the following two properties:

- At the request of one of the players, the channel can be examined by an auditor (this is the reason we call the channel *recordable*).
- At the end of the conversation, if the two parties agree, the recording is erased and lost.

The first property is important for robustness, and the second for coercion-resistance. This assumption calls into some physical device implementing these properties, e.g., a printer printing the transcript between the two parties, where later on the printout is shredded. Similar channels appear in previous works in the area. In Sako and Kilian [SK95] and in Hirt and Sako [HS00] the channel is defined to have the second property only (and indeed no robustness is supplied). In Chaum's visual scheme proposal [Cha04], and in the following protocols of [CRS05, AR06] parts of the protocol transcript can be shredded. We discuss this in more detail in Section 4.4.

Chapter 3

Cryptographic Tools

We briefly review the cryptographic tools we will refer to in the next chapters.

3.1 ElGamal cryptosystem

ElGamal [Gam85] is a frequently used probabilistic public key cryptosystem which we will use later. We work with ElGamal over a multiplicative group of prime order, as suggested by [Pfi94, SK95]. At first, we publicly choose two large primes q' and q such that $q|q'-1$, i.e., $q' = qk+1$ for some integer k. We also fix a generator g' of $\mathbb{F}^*_{q'}$. The cyclic group G we work with is the one generated by $g = (g')^k$ and has order $\frac{q'-1}{k} = q$.

Now, the three cryptosystem algorithms are:

Key generation algorithm - Randomly select $x \in_R \mathbb{Z}^*_q$ and make it the private key. The corresponding public key is -

$$h = g^x \tag{3.1}$$

Encryption algorithm - To encrypt a message $m \in G$ (given the public key h), we uniformly choose $r \in_R \mathbb{Z}^*_q$ and output -

$$E(q', q, g, m, h; r) = (g^r, m \cdot h^r) \tag{3.2}$$

Decryption algorithm - To decrypt a message (α, β) we compute -

$$m \;=\; \beta \cdot \alpha^{-x} \tag{3.3}$$

As we will work with global values q', q, g and h which are public and shared by all participants, we abbreviate $E(q', q, g, m, h; r)$ to $E(m; r)$.

ElGamal is homomorphic, i.e.,

$$E(m_1; r_1) \cdot E(m_2; r_2) \;=\; E(m_1 \cdot m_2; r_1 + r_2) \tag{3.4}$$

where the product of the ciphertext pairs is element-wise.

A re-encryption of an encrypted message $(\alpha, \beta) = E(m; r)$ is the value $E(1; r') \cdot (\alpha, \beta)$ for some $r' \in_R \mathbb{Z}_q^*$ which is $E(m; r + r')$, i.e, another encryption of m.

Last, let us denote by $E(m; U)$ the distribution obtained by picking r uniformly at random from U and evaluating $E(m; r)$. We have the following assumption about ElGamal:

Assumption 1 *(ciphertext indistinguishability) For every $m \neq m' \in G$, for almost all secret keys, no computationally bounded process can distinguish between the distribution $E(m; U)$ and the distribution $E(m'; U)$.*

We remark that assumption 1 does not hold if we choose a random q' and take q to be $q' - 1$, and in particular a non-prime. This was pointed out in [Pfi94], and we repeat the argument here as we think it clarifies things. As q' is prime, $q = q' - 1$ is divisible by 2. In particular, the cyclic group G has a subgroup A of order $q/2$. Also, it is easy to test membership in A by checking that the element is a $q/2$ root of unity. Now, if g^r happens to be in A (which happens with probability half) then so does h^r, and in particular $m \cdot h^r$ belongs to A iff m does. Thus, one can discover one bit of information about m, and in particular distinguish between $m \in A$ and $m' \notin A$. We therefore use as already suggested in [Pfi94] a prime subgroup G of $\mathbb{Z}_{q'}^*$.

We note that the famous RSA [RSA83] is also a public key cryptosystem. RSA is based on the hardness of factorization and has the useful property that $E(D(m)) = m$ which can be used for digital signatures.

3.2 Secret sharing

A *(t,N) - threshold secret sharing scheme* allows a dealer to share a secret between N players such that any group of t $(\leq N)$ or more players can reconstruct the secret, but any group of less then t players learn nothing about it.

One classic solution to this problem is the *Shamir's Secret Sharing Scheme* [Sha79]. In this scheme, the secret (S) is an integer in $\mathbb{F}_p$ (where p is a public prime integer):[1]

- The dealer selects $t-1$ secret random integers $a_i \in_R \mathbb{Z}_p^*$.
- The dealer calculates N shares of the secret -

$$s_i = S + \sum_{j=1}^{t-1} a_j \cdot i^j \tag{3.5}$$

- The dealer sends s_i to player i for $i = 1 \ldots N$.

Now, any group of t players can calculate (using Lagrange interpolation over the field $\mathbb{F}_p$) the value of S using their shares $s_{k_1}, \ldots, s_{k_t}$ -

$$S = \sum_{j=1}^{t} s_{k_j} \lambda_j \text{ where } \lambda_j = \prod_{i=1..t, i \neq j} \frac{k_i}{k_i - k_j} \tag{3.6}$$

Because we started with $t-1$ degree polynomial, any group of less then t shares has no information about S.

One modification of this scheme is the *Publicly Verifiable Secret Sharing scheme* presented in Appendix A. This a scheme has two improvements over the above scheme:

- Anyone can verify that the dealer sent valid shares.
- When a player reveals his share s_i, anyone can verify if it is a real share.

3.3 Threshold cryptosystem

A *(t,N) - threshold cryptosystem* allows decryption of a ciphertext only when a group of t $(\leq N)$ or more players cooperate, but any group of less then t players can not

[1]Where all computations are in field $\mathbb{F}_p$.

gain any information about the plaintext. There is no single dealer who knows the secret as in the *(t,N) - threshold secret sharing scheme.*

One solution to this problem is the *Threshold ElGamal Cryptosystem* [Ped91, Ped92] where the solution is based on N parallel executions of publicly verifiable secret sharing. Informally, the scheme consists of three stages: a *key generation* stage where the threshold public key is computed, a *keys distribution* stage where each player distributes his (private) share of the threshold public key and computes his share of the threshold private key, and, a *decryption* stage.

We will use the same notation from Section 3.1 for ElGamal cryptosystem, meaning we have a group G with a generator g and order q. As we mentioned, the scheme starts with a key generation stage:

- Each player P_i chooses a private key $x_i \in_R \mathbb{Z}_q^*$ and computes a share $h_i = g^{x_i}$. He publishes a commitment $C_i = Commitment(h_i, r_i)$ using a random r_i.
- When all participants published their commitments, each player opens his commitment C_i and publishes h_i, r_i.
- The threshold public key is computed $h = \prod_{j=1}^{N} h_j$.[2]

Next, each P_i distributes his private key x_i using a (t,N)-verifiable threshold secret share:

- P_i chooses a random polynomial $f^{(i)}$ with coefficients over $\mathbb{Z}_q$ of degree $t-1$ where $f_0^{(i)} = x_i$ -
$$f^{(i)}(z) = \sum_{j=0}^{t-1} f_j^{(i)} \cdot x^j \qquad (3.7)$$
- P_i publishes $F_{ij} = g^{f_j^{(i)}}$ for $j = 0 \ldots t-1$.
- After all participants published their F_{ij} values, P_i *privately* sends a signed share $s_{ij} = f^{(i)}(j)$ to each P_j.
- When P_i receives a share s_{ji} from P_j he verifies his consistency by checking

[2]The corresponding private key is as in ElGamal, $x = \sum_{j=1}^{N} x_j$, but it is not known to anyone.

that-

$$g^{s_{ji}} = \prod_{l=0}^{t-1} F_{jl}^{i^l} \tag{3.8}$$

In case of inconsistency, P_i can publish the signed value of s_{ji}.

- P_i signs on h (and after all participants have signed, the public key is legitimate).
- P_i calculates his share of the threshold private key x- [3]

$$s_i = \sum_{j=1}^{N} s_{ji} \tag{3.9}$$

and publishes a commitment $z_i = g^{s_i}$. Any player P_j can verify z_i by -

$$g^{s_i} = \prod_{l=0}^{N} g^{s_{il}} = \prod_{l=0}^{N}(\prod_{k=0}^{t-1} F_{lk}^{i^k}) \tag{3.10}$$

Last, a decryption of (α, β) can be computed by any group of t players $P_{k_1}, \ldots, P_{k_t}$ using the following:

- Each player P_{k_i} publishes $w_i = \alpha^{s_{k_i}}$ and proves that $Log_g z_{k_i} = Log_\alpha w_i$ using a non-interactive zero-knowledge proof (see Section 3.5).
- Using a Lagrange interpolation the plaintext can be recovered by $m = \frac{\beta}{\prod_{j=1}^{t} w_i^{\lambda_{k_i}}}$ (where λ_{k_i} are lagrange coefficients).

3.4 Anonymous channels and Mix-nets

David Chaum introduced the idea of anonymous channels [Cha81] in order to anonymize e-mail communication. His idea is based on a set of authorities who route e-mails in a way they can not be traced without some help from the authorities. This set of authorities is called a Mix-net (and each authority is called a Mix). A simple mix-net can use any public key encryption in a protocol as followed (also known as *Chaumian mixes*):

[3]Let f be a polynomial at degree N over $\mathbb{Z}_q$ and let $s_i = f(i)$. We observe that $f(0) = x$ and thus s_i is a share of x.

- Each mix M_i (for $i = 1 \dots n$) has a secret key x_i and a public key h_i.
- A player who wishes to communicate m, sends a message $E_{h_1}(E_{h_2}(\dots E_{h_n}(m)))$ to the first mix M_1.
- M_1 gets messages from several players, decrypts its input messages using its secret key, randomly permutes them and then sends the permutated messages to M_2.
- M_2 decrypts the messages from M_1, randomly permutes them and sends the permutated messages to M_3 and so on...
- M_k sends the plaintexts to their destination (we assume the destination is part of the plaintext).

This type of mix-net is called *decryption* mix-net. If one of the mixes is honest, messages can not be linked to senders.

Other variants of mix-nets are *re-encryption mix-nets* and *decrypt and re-encrypt mix-nets*.[4] When the underlying encryption scheme is homomorphic (e.g., ElGamal or Pailier), this can be somewhat simpler. E.g., with ElGamal:

- Each mix M_i (for $i = 1 \dots n$) has a secret ElGamal key x_i and a public key $h_i = g^{x_i}$. Lets denote $H_{>=i} = \prod_{j=i}^{n} h_j$.
- A player who wishes to communicate m selects a random number r and sends a message $E(m, H_{>=1}; r)$ to the first mix M_1.
- For each of its inputs (a, b), M_i decrypts using its secret key, selects a random number r' and re-encrypts using r' and H_{i+1}. Then, it outputs all those re-encryptions in a random permutation and sends it to the next mix.
  Formally, it decrypts each input $(a, b) = E(m, H_i; r)$ and gets $(\overline{a}, \overline{b}) = E(m, H_{i+1}; r)$. Then it re-encrypts and gets $(a_{new}, b_{new}) = (\overline{a}, \overline{b}) \cdot E(1, H_{i+1}; r') = E(m, H_{i+1}; r + r')$.

Such re-encryption mixes will be used later in Chapter 4.

Mix-nets are very useful. It can be used for anonymous communication of any kind, including emails, peer-to-peer networks and anonymous vote casting.

[4]Decrypt and re-encrypt mix-nets are often also refereed to as re-encryption mix-nets.

3.4.1 Verification of mixes

Here we are concerned with verifying that a mix functions properly. I.e., we want to make sure that it did not alter, delete or insert messages.

There are many types of mixes and many methods for verifying their behavior. The two common methods are: A verifiable secret shuffle [Nef01, Gro03] and Randomized partial checking (RPC) [JJR02]. The first is based on a zero-knowledge proof of the validity of the mix shuffle. It is very efficient but works only with a re-encryption mix-net. The second, RPC, is less efficient but can be used for any type of mix-net.

We present a simple implementation of RPC as suggested by Chaum [Cha04]. We use a mix-net where each authority i is responsible for the two consecutive mixes M_{2i-1} and M_{2i}. Let A denote the set of inputs to M_{2i-1} and B the set of inputs to M_{2i}. We select a random subset $A_i \subset A$ of inputs. Let $B_i \subset B$ be the set of outputs of M_{2i-1} (inputs of M_{2i}) that are connected to A_i. We ask mix M_{2i-1} to reveal the edges leaving inputs in A_i, and we ask mix M_{2i} to reveal the edges leaving inputs *not* in B_i (See Figure 3.1).

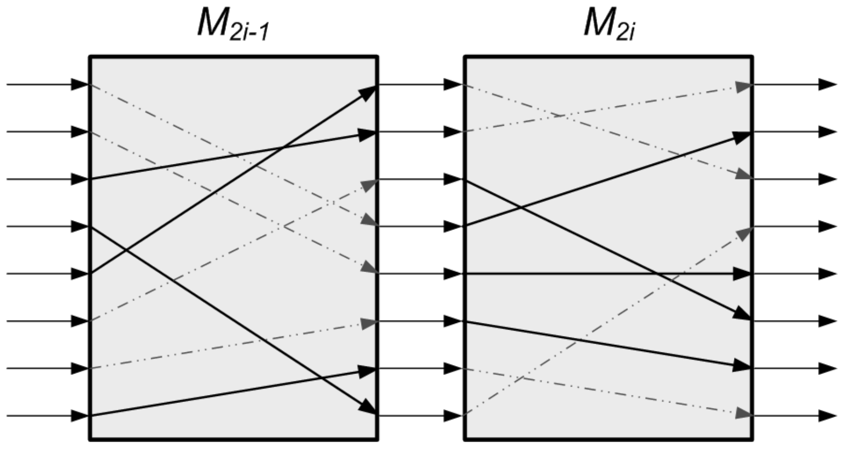

Figure 3.1: Achieving universal-verifiability using dual mixes. A party is responsible for two consecutive mixes. The party has to reveal a random subset of the first layer, and the complement subset for the other layer.

Revealing an edge means revealing its two ends and proving the transformation over that edge (e.g., decryption and re-encryption) was done properly. For a re-encryption ElGamal mix-net as we described above, revealing an edge (a, b) means publishing on the public board the following:

- The corresponding output ElGamal tuple (a_{new}, b_{new}).
- The intermediate values $\bar{b}$ and r' used in the computation of (a_{new}, b_{new}).
- A non-interactive zero-knowledge proof that the decryption was properly done (proving that $\log_g h_i = \log_a(b(\bar{b})^{-1})$).

Using this implementation, each mix reveals only one bit of information for each message, but also, if a mix tries to cheat about t messages it will be caught with probability $1 - 2^{-t}$.

3.5 Zero-knowledge proofs

Zero-knowledge proofs are interactive protocols between a prover and a verifier, in which the prover proves to the verifier, with high probability, that some statement is true, while not leaking any information besides the validity of the statement. I.e, the proof should satisfy the following:

Completeness - If the statement is true, an honest verifier always accepts the proof.

Soundness - If the statement is false, the verifier will reject the proof with a high probability.

Zero-Knowledge - If the statement is true, a cheating verifier learns nothing besides the validity of the statement (formally, this means that there exists a simulator that can simulate the protocol transcripts).

If the zero-knowledge guarantee is only for an honest verifier, we say the protocol is an *honest verifier zero-knowledge (HVZK) proof.* If all the verifier does during the interaction is picking random coins and sending them as his messages, we say the protocol is a *public-coin* protocol. An HVZK and public-coin protocol can be converted into a non-interactive zero-knowledge proof using the Fiat-Shamir heuristic [FS86], by setting the verifier's messages to be the hash of the preceding transcript.

We now present three zero-knowledge proofs (ZKP) which will be used later on.

3.5.1 Zero-knowledge proof of equality of discrete logarithms

Let G be a multiplicative group of order q, and let g_1, g_2 be two generators of G. The inputs are $v, w \in G$. The prover knows the discrete logarithms of v and w, i.e., x_1 and x_2 such that $v = {g_1}^{x_1}, w = {g_2}^{x_2}$, and claims they are the same, i.e., $\log_{g_1} v = \log_{g_2} w$. Lets denote $x = x_1 = x_2$.

The following protocol is from [CP92]:

- The prover chooses a random $z \in \mathbb{Z}_q^*$ and sends $a = g_1^z$, $b = g_2^z$ to the verifier.
- The verifier chooses a random challenge $c \in \mathbb{Z}_q^*$ and sends it to the prover.
- The prover sends $r = (z + cx)\ (mod\ q)$ to the verifier.
- The verifier checks that $g_1^r = av^c$ and $g_2^r = bw^c$.

The protocol is honest verifier, perfect, statistical zero knowledge, with perfect completeness and $1/q$ soundness error. It is not known to be zero-knowledge against dishonest verifiers.

It is also a three round public coin protocol, so the proof can be turned into non-interactive by the Fiat-Shamir heuristic (changing the challenge c to a hash of a, b, v, w).

3.5.2 A Zero knowledge proof for 1-out-of-ℓ re-encryption

We use the same notation as before. Let G be the multiplicative group as before and let $g \in G$ be the generator and $h \in G$ the ElGamal public key. Now, the prover wants to prove that one of the ℓ pairs $(x_1, y_1), \ldots, (x_\ell, y_\ell)$ is an ElGamal re-encryption of the pair (x, y). Say, the re-encrypted pair is $(x_t, y_t) = (x, y) \cdot E(1; r) = (xg^r, yh^r)$, where r is known only to the prover. The protocol is described in Figure 3.2 and is taken from [CGS97].

Using Fiat-Shamir heuristic, the protocol can be made non-interactive using the challenge $c = H(x, y, a_1, \ldots, a_\ell, b_1, \ldots, b_\ell, x_1, \ldots, x_\ell, y_1, \ldots, y_\ell)$. The prover publishes $c, d_1, \ldots, d_\ell, r_1, \ldots, r_\ell$ and verifying is the same as before.

Re-encryption is a symmetric property (if (a', b') is a re-encryption of (a, b), then (a, b) is a re-encryption of (a', b')).[5] In particular, the above is also a ZKP for the

[5]This is because $E(1; r)^{-1} = E(1; -r)$.

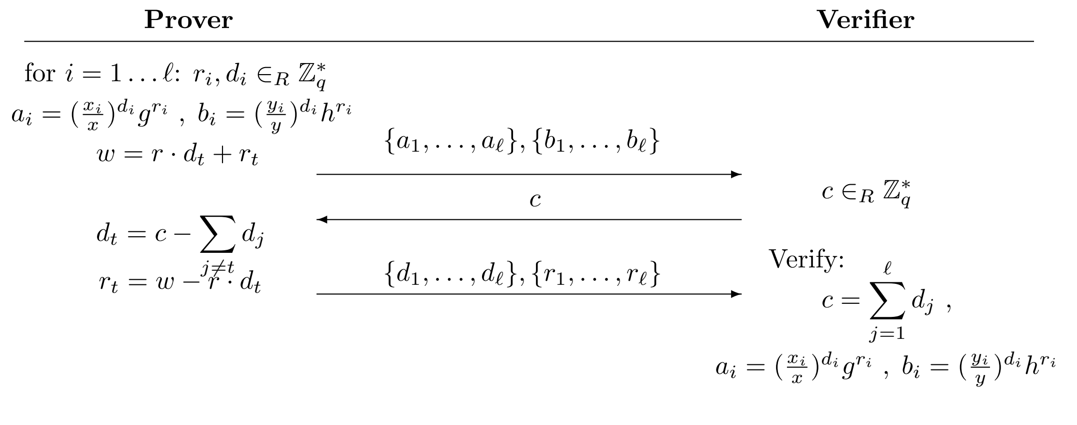
Prover
Verifier
for $i = 1 \ldots \ell$: $r_i, d_i \in_R \mathbb{Z}_q^*$
$a_i = (\frac{x_i}{x})^{d_i} g^{r_i}$, $b_i = (\frac{y_i}{y})^{d_i} h^{r_i}$
$w = r \cdot d_t + r_t$
$\{a_1, \ldots, a_\ell\}, \{b_1, \ldots, b_\ell\}$
$c \in_R \mathbb{Z}_q^*$
c
$d_t = c - \sum_{j \neq t} d_j$
$r_t = w - r \cdot d_t$
$\{d_1, \ldots, d_\ell\}, \{r_1, \ldots, r_\ell\}$
Verify:
$c = \sum_{j=1}^{\ell} d_j$,
$a_i = (\frac{x_i}{x})^{d_i} g^{r_i}$, $b_i = (\frac{y_i}{y})^{d_i} h^{r_i}$

Figure 3.2: Re-encryption of 1-out-of-ℓ interactive proof.

case where we are given (x, y) and we want to prove that it is a re-encryption of one of the ℓ pairs (x_i, y_i).

3.5.3 A Zero knowledge proof for 1-out-of-ℓ message encryption

We now look at the following problem: we are given ℓ plaintext messages $m_1, \ldots, m_\ell$ and one encryption (x, y) and we want to prove that it encrypts one of the ℓ plaintext messages. The protocol for that is given in [CGS97] and is based on the 1-out-of-ℓ re-encryption protocol. We give it here for completeness.

Given $m_1, \ldots, m_\ell$ and $(x, y) = E(m_t; r)$ (for some t and r known to the prover), the prover publishes $(x_i, y_i) = (x, y m_i^{-1})$ for $i = 1 \ldots \ell$. It is easy to check that $(x_i, y_i) = E(m_t m_i^{-1}; r)$. The prover now proves that one of (x_i, y_i) is a re-encryption of $E(1; 1)$ using the ZKP from the previous section.

3.6 Coercion in zero-knowledge protocols

We mention that in both the interactive and the non-interactive protocols in Section 3.5 the prover is coercible if the transcripts are public. For example, during the interactive protocol of *Zero-knowledge proof of equality of discrete logarithms* (Section 3.5.1) the prover commits to z (using g_1^z). If the transcripts are public, a coercer can coerce the prover to reveal z (which can be done only in one way) and using this he can calculate $x = (r - z)/c$. In the non-interactive protocol this coercion is done using the hash function and z.

Chapter 4

Existing Voting Protocols

In the last 30 years, many electronic voting protocols have been proposed. We review here the ideas we think are the most related to our research and have a big influence.

We can describe a generic process of electronic voting by:

Voter identification - almost all electronic voting protocols use the voter's ID to identify a voter, and then check that he or she is an eligible voter (who votes only once).

Casting a vote - The voter casts a vote using some kind of a voting booth or a public board. Most electronic voting protocols use isolated voting booth in order to achieve coercion-resistance, and assume the booth does not cooperate with coercers.

An important issue here is whether the voter can use a computer in the booth or not (what we call *bare-handed voting* [1]).

Verifying that a vote was casted as intended - In most electronic voting protocols, the voter or some other organizations verifies that the voter's vote is recorded as intended. We mention that this step is usually skipped in DRE machines. Indeed, this is the main reason why forgery is a sensitive issue with those machines and why many forgery allegations were raised (see [Har03, KSRW04]) and actually happened (see [Wik07a] and the Emmy award nominated film

[1]We do not use here the notation of Voter-verifiable which means that the voter can only verify that his vote was casted correctly. We want to emphasize that the voter is only a human and when he is voting, he has only human abilities and no computational device.

Hacking Democracy).

Tallying the votes - The registered (encrypted) votes are tallied.

Verifying the tally - Auditors (ideally everyone can be an auditor) verify that the tally is correct.

For many years, the goal of electronic voting protocols was to present a secure protocol assuming the voter can carry any efficient computing task. Such an assumption is both unpractical (it assumes a voter has to come to the booth with a computing device) and risky (it make correctness depend on the voter's computer honesty). Nevertheless, it gives the base ground for many more recent protocols. Roughly speaking, one can divide such protocols (where a computing device is used by the voter) into the following three categories: using blind signatures (e.g., [FOO92, Oka98]), using mix-networks (e.g., [SK95, HS00, Nef01]), and, using homomorphic encryption (e.g., [BT94, CGS97, Sch99, BFP$^+$01]).

These basic protocols guarantee privacy against passive adversaries, i.e., in a scenario where dishonest votes are independent of honest votes. If we allow active adversaries, i.e., if dishonest players can vote based on what they see so far on the public board, then privacy is often not guaranteed (we present this problem in detail in Section 4.2.2).

Benaloh and Tuinstra proposed a receipt-free protocol which was later broken [Hir01]. Sako and Kilian [SK95] proposed a receipt-free protocol using mix-networks and Chameleon blobs and a physical assumption that is similar to the recordable, private channels we defined, but their protocol requires the voter to know at least one mix which is honest (rather than just knowing that one such mix exists). [HS00] proposed a similar but more efficient solution using threshold encryptions, but it has the same drawback. Moreover, both protocols can be coerced.[2]

Some other related receipt-free protocols use another participant or device to add randomness (that is not known to the voter) to the protocol: [MBC01] proposed a solution which uses a tamper resistant smart-card that produces a random value hidden from the voter, and [BFP$^+$01] proposed a solution which requires the use of an authority for randomness, similar to the role of the booth in our protocol.

[2]A coercer can force the voter to vote randomly and verify his behavior.

Bare-handed protocols started with the ground-breaking works of Chaum and Neff [Cha04, Nef04]. Many other protocols followed (e.g., [CRS05, Rey05, LTR$^+$06b], and the more recent [Cha07, AR06, MN06]). In many of these protocols there is no privacy towards the booth (and the voter simply tells his vote to the booth), and in many of these protocols privacy towards a malicious ballot creator is lost.

4.1 Chapter outline

We start by reviewing protocols which assume that the voter has computational power in the booth. One of the most influencing idea in this area is Chaum's mix-nets [Cha81]. Anonymous communication channels underly in many electronic voting protocol. We review [FOO92] protocol which uses anonymous channels and blind signatures. Next, we sketch [SK95] which is simple and yet strong protocol that is based on mix-nets (Section 4.2.2). We also review a novel protocol based on Threshold encryption (Section 4.2.3) and its improvement using mix-net (Section 4.2.4).

As we mentioned earlier, most of the early electronic voting protocols assume the voter has computational power at the booth. On the second part we review two protocols that drop this assumption. We briefly explain Chaum's visual scheme, and review a variant of Chaum's scheme named Prêt *à* voter (PaV) [CRS05] which is as strong as [Cha04] but easier to understand and to implement. We also present a short section about the main problems with current bare-handed protocols.

4.2 Protocols which assume the voter has a computational power in the booth

4.2.1 [FOO92]

The protocol is based on two ideas proposed by Chaum: anonymous channels (Section 3.4) and blind signatures [Cha82]. A blind signature allows party A to get a signature of party B on a message, without revealing any information about the message to party B. The method consists of three steps (described very informally):

Blinding - Party A *blinds* the message by adding a pseudo-random data to the message.

Signing - Party B sings the blinded message.

Un-blinding - Party A removes the blinding by removing the pseudo-random data.

Now, a sketch of [FOO92] protocol is as follows:

Initialization - The elections authority and the mixes publish their public keys.

Encrypt a vote - The voter selects his vote b and encrypts it $E(b; r)$.

Get a blind signature - The voter identifies to the elections authority and gets a blind signature of the authority on the encrypted vote $Sig_{authority}(E(b; r))$.

Publish the encrypted vote - The voter sends $Sig_{authority}(E(b; r))$ to the public board using the anonymous channel.

Open the votes - After the elections are over, the voter sends b, r and $Sig_{authority}(E(b; r))$ to the public board using the anonymous channel. The public board publishes it and anyone can compute the final tally and audit the elections.

One might ask why the voter sends two messages instead of sending a signed open vote in the first message. In such a scenario the votes will be published during the elections, and therefore, we will lose fairness.

Protocol properties:

The protocol is private (as long the anonymous channel is secret) and achieves robustness, unforgability, eligibility and auditability (only by the voters). The protocol does not achieve coercion resistance because a voter can prove what his vote is, simply by showing b, r and $Sig_{authority}(E(b; r))$ to the coercer before the last stage.

4.2.2 [SK95]

In their paper [SK95], Sako and Kilian presented the universally verifiable mixes (over ElGamal cryptosystem) and one application of such mixes, a receipt-free voting protocol.

If we simplify their ideas, we find two novel voting protocols. The first protocol is rather simple: a voter encrypts his vote and publishes it on a public board, and from now on, all the mixes' computations are universally verified. This is a straightforward use of the universal verifiability property of the mixes. The second protocol, which is a receipt-free, is a little more complex and similar to [HS00] which we will describe in Section 4.2.4.

Protocol properties (of the first protocol):

The protocol is private against passive adversaries. As long as there is at least one honest mix, finding a voter's vote is reducible to breaking ElGamal. Active adversaries can learn (with high probability) what any specific voter has voted. This was observed by Pfitzmann [Pfi94], and we describe this attack soon. The protocol enjoys unforgeability (due to the verification steps), eligibility, unreusability and universal auditability. As for robustness: if a mix stops working the mixing stage is stuck. Such a mix can be replaced if we use publicly verifiable secret sharing [Sch99], but then we need at least half of the mixes to be honest. As before, the protocol does not achieve coercion-resistance or even receipt-freeness because the randomness used for the encrypted vote can be used as a receipt.

We finish this section by describing Pfitzmann [Pfi94] attack for active adversaries. Let there be t possible votes $m_1, \ldots, m_t$. A voter A votes m, and his masked vote $z = E(m; U)$ is published on the public board. Another dishonest voter C arrives, chooses a random $w \in G$ and votes $z \cdot E(w; U)$. Let us also assume that all other voters are honest. When all the votes are processed, C's vote, with very high probability, will not be in the set of legal votes $\{m_1, \ldots, m_t\}$. In particular, C is able to identify his vote b, learn that he actually voted the value $b = mw$, and therefore deduce that $m = bw^{-1}$.

Indeed, Pfitzmann shows that this attack exists even if all values are legal votes but the number of voters V is small compared to the size of G. The attacker C takes A's vote $(a, b) = E(m; U)$ and converts it to the vote (a^c, b^c) for some random c, which corresponds to voting $E(m^c; U)$. When all the votes are revealed C looks for a pair of votes m and m^c, and with a high probability there is only one such pair, which must represent his value. Again, C learns m.

A natural direction towards this problem is using a public-key cryptosystem that resists chosen ciphertext attacks. In such a system, the active adversary can indeed detect his modified message, but can not deduce from it anything about the original message. Indeed, [ZS93, CS98] for example, showed how to turn public key encryption to one that is immune against chosen message attacks. However, any such transformation, by definition, loses the homomorphic properties of the encryption.

4.2.3 [CGS97]

In [CGS97] Threshold encryption is used to replace the mix-net. The protocol consists of n trustees $T_1, \ldots, T_n$ and a public board, and goes as follows:

Initialization - All official trustees publish their own public keys and an ElGamal threshold encryption public key H is published. Also, another generator h of the multiplicative group is chosen and published (along with all other ElGamal public values we saw: multiplicative group size and a generator g of the group).

Voting - A voter casts his vote $b \in \{h^1, h^{-1}\}$ by publishing $E(b) = E(b; r) = (g^r, H^r \cdot b)$. He also publishes a NIZKP that this vote is legal (and not h^{100} for instance). He does that using the NIZKP from Section 3.5.2.

Tallying - After the elections are over, the value of $E(h^d) = \prod_b E(b; r)$ is publicly computed using the homomorphic property of ElGamal. A group of t trustees decrypts it and publishes h^d. Using exhaustive search, anyone can calculate d which is the difference between the two candidates.

We remark that the protocol can be extended to any bounded number of candidates (described in [CGS97]).

Protocol properties:

This protocol achieves privacy (because of the use of threshold encryption, privacy is achieved as long as there are at most $t-1$ dishonest trustees) even against active adversaries, robustness (as long as there are at least t honest trustees who can decrypt the results), unforgeability (malicious behavior is caught in the NIZKP), eligibility (using physical identification), unreusability and universal auditability. The protocol does not achieve coercion-resistance or even receipt-freeness because the randomness used for the encrypted vote can be used as a receipt.

4.2.4 [HS00]

Hirt and Sako proposed a protocol [HS00] which is a combination of [SK95] and [CGS97]. It uses universally verifiable mixes to achieve coercion-resistance and verifiability (as presented in [SK95]) and threshold encryption for efficiency (as presented [CGS97]):

Initialization - All mixes publish their public keys (as shares) and compute a threshold public key H. They also publish another generator h of the multiplicative group.

Creating encrypted votes - For every voter, the first mix creates a list of *zero encryptions*[3] of all valid votes (the votes are as in [CGS97], h^{-1} and h^{1}). Then, it sends them into a re-encryption mix-net which does not decrypt, but only re-encrypts using the threshold public key H. Each mix re-encrypts the messages and permutes them, so the output of the last mix is a random permutation of encryptions of all valid votes. Each mix also commits (using Chameleon blobs and the voter's public key) its permutation on the public board.

Voting - We assume there is an untappable channel between each mix and the voter. Using those channels, each mix de-commits its permutation. Now, the voter can link the votes to their encryptions.
The voter publicly selects the encryption of his vote and the other encryptions are discarded.

Tallying - After elections end, t mixes calculates $E(h^d) = \prod_b E(b; r)$ and decrypt it to get h^d. Again, anyone can find d using linear time exhaustive search which is the difference between the two candidates.
Those calculation can also be verified using a non-interactive zero knowledge proof of equality of discrete logarithms (3.5.1).

Protocol properties:

Similar to [SK95] with few minor refinements: privacy is achieved as long as there are at most $t - 1$ dishonest mixes. Coercion-resistance is achieved as long as the voter

[3] We denote by zero encryption of a message m the tuple $(1, m)$ which is an encryption of m using randomness $r = 0$.

knows at least one mix which is not under the control of the coercer. Robustness is achieved as long as there are at least t honest mixes.

4.3 Protocols which assume the voter is bare-handed

4.3.1 Chaum's visual scheme

In [Cha04], Chaum presented the first bare-handed protocol. We describe a sketch of Chaum's visual scheme. For full details, see [Cha04].

A voter V identifies using an ID and enters a voting booth B. First, V selects his vote m on a touch-screen. Then, B prints two layers of papers L_1, L_2 each consisting of a binary image M_{L_i} and some numeric strings. V verifies that the numeric strings on both of the layers are identical. He also checks that $m = M_{L_1} \oplus M_{L_2}$ by putting the two layers one on top of the other and checking that the created image is his vote. This is the visual part of the protocol and it is based on Visual cryptography [NS94].

If one of the tests fails, B is caught as malicious (but notice that the voter has no concrete proof for that). Otherwise, V selects one of the layers and B signs and publishes the selected layer. The other layer is shredded. The booth also prints a receipt with the selected layer and gives it to V. Using the receipt, V can later on check that his vote was published correctly on the public board. The receipt tells nothing about the real value of the vote and can not be used for vote selling.

When the elections are over, the encrypted votes are being decrypted using a mix-net and all the calculations are verified using randomized partial checks (Section 3.4.1).

There are a few disadvantages to notice: The booth knows what anyone voted. The booth can show the voter one thing and print another, or publish another value on the public board and the voter can not prove the booth is cheating because L_1, L_2 are not signed. Signing L_1, L_2 does not hold either, because the voter can not really check the signature without a computer.

4.3.2 Prêt à voter

We now present the protocol from [CRS05] named *Prêt à Voter* (PaV). This protocol requires n mixes denoted by $M_1, \ldots, M_n$, each mix M_i has a public key h_i and a private key x_i. We denote by k the number of candidates.

First, an official authority creates many printed ballots, each ballot consists of two columns: the left column is the list of candidates, lexicographically ordered and shifted with some random shift $\pi \in [1..k]$. This shift is encrypted and the encryption is printed in the bottom of the right column (Figure 4.1).

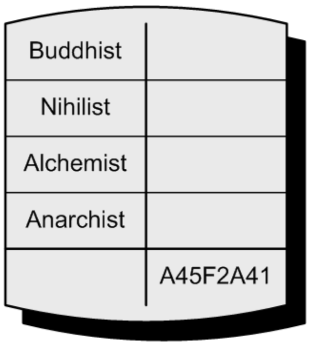

Figure 4.1: Empty Ballot ($\pi = 2$)

The process of creating such a ballot is (we denote by κ a security parameter, e.g., $\kappa = 2^{32}$) :

- Select $g_i \in_R \mathbb{Z}_\kappa$ for $i = 1 \ldots n$.
- Calculate $d_i = hash(g_i) \ (mod\ k)$ for $i = 1 \ldots n$ using a cryptographic hash function.
- Calculate $\pi = \sum_{i=1}^{n} d_i \ (mod\ k)$.
- Select $D_0 \in_R \mathbb{Z}_\kappa$ and recursively calculate $D_i = E_{h_i}(g_i \circ D_{i-1})$ for $i = 1 \ldots n$.
- Print a ballot with a shift π and an encrypted printed shift D_n

This process is done by the official authority independently for each ballot. In order to verify that the ballots are well-formed, a random set of ballots is chosen and tested (opened by the mixes without being counted in the final tally).

When a voter enters the booth, he randomly picks one of those paper ballots. Then, he marks an χ near his chosen candidate (Figure 4.2). The voter detaches the left column and shreds it. He scans the right column and takes it home as a receipt (Figure 4.3). The scanner or a poll-worker signs on the receipt to approve it was casted. Also, in order to test the validity of the ballots, the voter may take home a few empty ballots and check them using a computer or some democratic organization helpers.

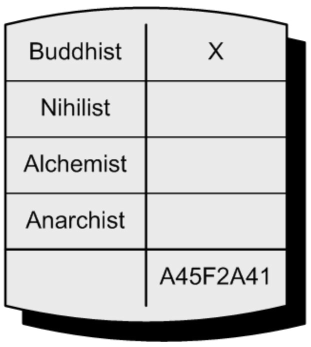

Figure 4.2: Filled Ballot

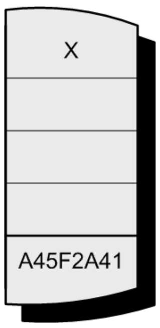

Figure 4.3: Detached Ballot

When elections end, all scanned ballots are published on the public board. We denote by a tuple $(r_j^{(N)}, D_j^{(N)})$ the information of the j-th scanned ballot where $r_j^{(N)}$ is the position of χ. Now, a process of mix-net decryption is engaged. Each mix M_i, in a descending order, performs the next process on its set of inputs $\left\{(r_j^{(i)}, D_j^{(i)})\right\}$:

- Calculates $g_i \circ D_j^{(i-1)} = D_{x_i}(D_j^{(i)})$, then $d_i = hash(g_i) \ (mod\ k)$ and $r_j^{(i-1)} = r_j^{(i)} - d_i \ (mod\ k)$ for each j.
- Publishes a random permutation of the set $\left\{(r_j^{(i-1)}, D_j^{(i-1)})\right\}$

We note that each mix changes the shift in a way that the sum of changes is the initial shift π. That way, the resulting value of of this process, r_0, is the voter's selection relating to the lexicographic order. Anyone can count the results using those values.

After the mixes finish their calculations, a RPC verification of the mixes is engaged (as described in Section 3.4.1).

Protocol properties:

Assuming the authority who is creating the ballots is honest then the protocol is private as long as there is at least one honest mix. It achieves robustness, unforgability, eligibility, unreusability, universal auditability and coercion-resistance (except for a random-coercion attack which we describe next).

4.4 Some problems with bare-handed protocols

The above protocols have several drawbacks (few are mentioned in [KSW05, LTR$^+$06b]). We describe four of them which we think are the most problematic.

4.4.1 Privacy depends on ballot creator

A malicious ballot creator can track the voter's votes or coerce him. Many improvements, modifications and variants of Prêt à Voter were proposed [LTR$^+$06a, LTR$^+$06b, Rey05, AR06, Cha07], none of which proved to solve this problem.

In some protocols the problem is even worse. If someone gets to see an empty ballot and remember its contents he can later on match the voter who used this ballot and his vote. E.g., in Prêt à Voter, if a party can see an empty ballot, it can remember the candidates order along with the encryption. Later on, it can recognize this ballot using the published encryption, match it with the candidates order (on the empty ballot) and match the voter and his vote. This attack is applicable by anyone who sees the full ballot, e.g, the creator, people handling the ballots and the poll-workers.

4.4.2 Deniable receipts

All bare-handed protocols (including the recent [AR06, Cha07]) use receipts for verifying the ballots are casted as intended. When a voter leaves the voting booth, he takes the receipt home where he can later on check it against the public board. If his ballot was not casted properly, he can detect the problem. This, however, does not necessarily mean he can *prove* the cheating. He could do that only if his receipt is signed.

This leads to another problem, can a bare-handed voter verify that a signed receipt is signed properly? All current digital signatures require a computer device even just for verification. Some ad-hoc solutions can be used (e.g., democratic organizations can check it for the voters, or, using a special paper instead of a signature) but we are not aware of any systematic solution.

4.4.3 Random-coercion

In many of the protocols, the voter can easily be coerced to vote for some random candidate, e.g., in Prêt à Voter [Rya05] the coercer can coerce the voter to vote for the first candidate, or, in PunchScan [Cha07] the voter can be coerced to mark the right bubble and bring the top page.

Furthermore, a more sophisticate random coercion attacks are also widely applicable. E.g., in Prêt à Voter, a coercer can coerce the voter to vote according to some function of the printed encryption, e.g., if the first character of the encryption is a digit then mark the first line, else, mark the third.

We note that both attacks also apply to non bare-handed protocols such as [SK95] and [HS00] where the voter can be coerced to vote according to some function of the published encryptions.

4.4.4 Chain voting

There are two main methods for introducing the ballot to a voter: using a computer which creates a new empty ballot in front of the voter, or, using a big bin of empty ballots and let the voter pick one at random. In the first, we have a privacy problem towards the booth. In the second, we are suspect to chain voting.

Chain voting is coercing a voter to vote using a given ballot. E.g., in Prêt à Voter, a coercer may steal an empty ballot, remember its contents and coerce a voter to use it. Later on he can find this ballot on the public board and see what the voter voted. Furthermore, the coercer can coerce the voter to steal a new empty ballot, so he could coerce another voter (which explains the term *chain voting*).

Chapter 5

Bare-Handed Electronic Voting with Pre-processing

5.1 An intuitive discussion

Let us summarize the situation so far. Someone has to prepare the encrypted ballot. If the voter prepares it at home, then we lose receipt-freeness (because the voter can open his vote) and we are susceptible to coercion-resistance (because the voter can be given the ballot by the coercer). On the other hand, if we ask the booth to encrypt the vote (as in Chaum's and Neff's protocols) we lose privacy towards the booth.

We could also go a middle way: ask the voter to prepare the encrypted vote, and then let the booth re-encrypt it. However, in such a case, the voter has to make sure that the booth properly re-encrypts his vote (e.g., to see that the booth is not multiplying his vote with an encryption of a value other than one) and we do not want the voter to do computations at the booth. A simple solution might be to ask the booth to put the re-encryption and the original vote on the public board, and let the auditors check the calculations, but then we are back to revealing the original vote, and the coercion problems.

The key idea behind our solution is very simple. We borrowed it from the way paper-ballot elections are currently carried out. In paper-ballot elections, privacy and coercion-resistance are obtained by making sure that the voting booth has paper ballots for each of the candidates. In a similar way, we ask the voter to prepare ballots with valid votes for *all* existing candidates. For reasons we explain shortly, we ask the voter to prepare *two* ballots. We also ask him to give a proof that:

- All the votes he prepared are legal and encode an existing candidate.
- He prepared two votes for each candidate, and he *knows* the correspondence between the votes and the candidates.[1]

These proofs can be prepared in advance.

The booth role is to re-encrypt the ballot's votes (we call this *ballot re-encryption*), which is necessary for coercion resistance. This, in turn, forces us to check the booth. The voter does this by asking two ballot re-encryptions and randomly choosing one of the two ballot re-encryptions for testing the booth. The testing itself is done by the auditors using the data that appears on the public board. The voter then uses the other re-encryption of his candidate for the actual voting.

Thus, in the first stage a poll-worker checks the voter can associate votes with candidates, and in the second stage the voter checks the booth properly re-encrypts messages. We use cut-and-choose tests for both. A coercer may potentially use both stages for coercion. The way we bypass these problems is by forcing both tests to apply to *all* candidates. If you prove you can associate a vote to a candidate you reveal information. But if you do that for all candidates you reveal nothing.

The implementation details are important as (not surprisingly) there are some subtle points hiding, as discussed in Section 4.2.2 (regarding active attacks) and Section 4.4 (regarding deniable receipts, random coercion and chain voting).

5.2 The underlaying protocols we use

Our solution is an extension of existing protocols. We take previous protocols as our underlying protocol and use it as our tallying protocol. Our extension simply replaces the vote casting stage.

[1]This is necessary because the coercer might give the voter a set of valid ballots but without telling him which encrypted vote corresponds to which candidate. We therefore ask the voter to show a poll-worker he can match votes with candidates.

We focus our attention on previous protocols that use ElGamal. We select protocols which have a separate phase for casting votes and a separate phase for tallying. We require that the casting ends with a published encrypted vote that can not be opened easily (e.g., without knowing all trustees' secret keys). We mention that a more general abstraction can be made, but in order to simplify our ideas and proofs, we focus on those specific conditions and protocols.

We can use the [CGS97] protocol (using threshold encryption for tallying) or the [SK95] protocol (using mix networks for tallying) as our underlying protocols. Both use ElGamal encryption. The immediate benefit of using ElGamal is its homomorphic property.

5.3 A formal description of the voting process

Pre-voting :

V prepares two ballots at home. Each ballot is printed on both sides (front and back) and contains records for each of the candidates.

Say there are D candidates. For every $i = 1 \dots D$, V picks a random string r_i and prepares an encrypted vote $y_i = E(m_i; r_i)$ for the i'th candidate m_i (where the specifics of this encoding function E depends on the underlying protocol), along with a NIZKP that y_i indeed encrypts a legal candidate.[2]

On the front side of the ballot, V prints D rows containing the D values y_i in a random order. On the back side, V prints D rows containing the D tuples (m_i, r_i) using the same random order. Also, on both sides, the voter's name (and a serial number if needed) appears in plaintext. See figure 5.1.

Voting (and verification) :

V identifies himself with an ID. He shows in front of a poll-worker (using a scanner for instance) the front sides of his two ballots, and this is published along with the voter's name on the public board for universal verification. See Figure 5.2.

The booth B and the auditors check that the non-interactive, zero knowledge

[2]Such non-interactive, zero-knowledge proofs are described in Section 3.5 and in [CGS97].

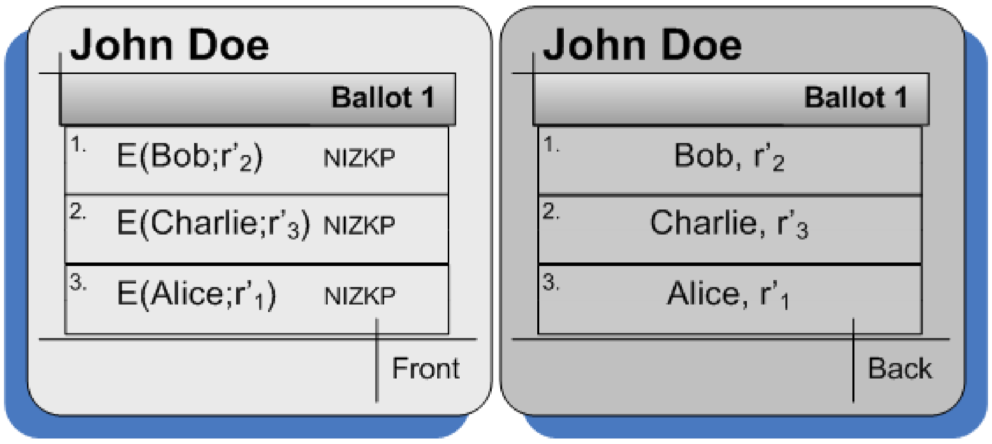

Figure 5.1: The ballot is printed on both sides. The back side contains (in plaintext) the candidates' names along with the random strings used for encrypting their corresponding votes. The front side contains the encrypted votes $E(m_i; r_i)$ along with a NIZKP that those encrypted values are valid.

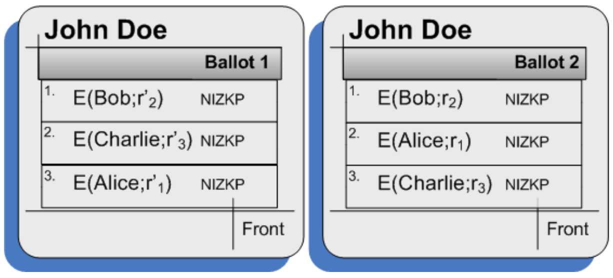

Figure 5.2: The voter shows the front sides of his two ballots in front of the poll-worker.

proofs are correct and all the votes on the front sides of the two ballots are legal. A poll-worker picks a random number $i \in \{1, 2\}$ and publishes i on the public board. The poll-worker asks the voter to scan the back side of the i'th ballot, and it is sent to the public board. The booth and the auditors check that the back side matches the front side. This guarantees that the voter knows how to open his ballots.[3] We denote by P the remaining ballot. The voter now enters the booth.

[3]Another subtle point is the following. A coercer might supply the voter with legal ballots whose back side is covered with a scratch area, and tell the voter to vote with a non-scratched ballot. The voter is able to show the back side of the test ballot (by first scratching it) but must keep the other ballot covered, effectively enforcing a random vote [Ano07]. We solve this problem by doing this test in front of a poll-worker.

Casting a vote :

Say the voter V wants to vote for the candidate that appears on the c'th row of P, $c \in \{1, .., D\}$. Then, V sends the number c to B. The value c is *not* posted on the public board (to avoid coercion).

Re-encryption (and verification) :

Say the front side of P has the D values $\{e_1, \ldots, e_D\}$. B computes *two* re-encryptions of the front side of P, i.e., two sets $P^{(0)} = \left\{e_1^{(0)}, \ldots, e_D^{(0)}\right\}$ and $P^{(1)} = \left\{e_1^{(1)}, \ldots, e_D^{(1)}\right\}$, where $e_i^{(0)}$ and $e_i^{(1)}$ are obtained by multiplying e_i by a random encryption $E(1; U)$ of 1. Then, the booth picks two random permutations $\pi_0, \pi_1 \in S_D$ and publishes $\pi_0(P^{(0)})$ and $\pi_1(P^{(1)})$ *on the public board*, where $\pi(P)$ is the set P, with the D rows of P permuted according to π. The booth also publishes on the public board a NIZKP that all rows in $\pi_0(P^{(0)}) \cup \pi_1(P^{(1)})$ are re-encryptions of some vote given in P. Finally, the booth also tells the voter, over the *recordable private channel*, the values $c_0 = \pi_0(c)$ and $c_1 = \pi_1(c)$. The voter publishes a bit $b \in \{0, 1\}$ and the booth reveals a permutation $\Pi_b \in S_D$ on the public board (and if the booth is honest then $\Pi_b = \pi_b$), along with the randomness used to create the re-encryptions in $P^{(b)}$. The *auditors* check correctness and the *voter* checks that $\Pi_b(c) = c_b$, i.e., that the booth's permutation is consistent with the ordering the booth declared to the voter.

Publishing a vote :

The booth publishes $c_{\bar{b}}$ over the public board and the vote is taken to be $P_{c_{\bar{b}}}^{(\bar{b})}$. The voter V checks that the published value matches $c_{\bar{b}}$ that was sent to him over the recordable, private channel. If everything so far is correct, V and B shred the channel's record (and in particular they shred $c, c_b, c_{\bar{b}}$) and V leaves the booth.

This completes the voting stage. Notice that the voter can pre-compute the votes and the non-interactive proofs in the ballots, and can come to vote at the booth bare-handed, carrying only his two ballots of votes.

5.3.1 Tallying

We can use the tallying suggested either in [SK95] or [CGS97]. For concreteness we work with [CGS97]. We replace the vote casting phase of the original protocol with the bare-handed extension above. As described before, [CGS97] protocol is based on threshold encryption, so we take the ElGamal public key H to be the threshold public key. Now, when a voter or a booth encrypts or re-encrypts a message, only a group of at least t trustees will be able to decrypt it. Tallying is done exactly as in [CGS97]. We use the NIZKP for making sure that the voter appears with a ballot having all possible candidates. We note that these NIZKP already appear in the original protocol of [CGS97] for checking the validity of the vote and so they should be combined.

5.3.2 Informal proof of correctness

Unforgeability, auditability and universal auditability -

The adversary here is an arbitrary coalition of the voting booth and (may be even all) the trustees. We start with the voting booth. It sends P'_0, P'_1 on the recordable channel. If the booth cheats about even one re-encryption of the ballot's re-encryptions and if this re-encryption is selected as P'_b, it is caught cheating. We now turn to the trustees. They can only cheat in the decryption. But because of the NIZKP they must give, they can cheat only with a negligible probability. Similarly, the voter can try to cast an illegal vote. Again, using the NIZKP (which was also in the original protocol) we catch a malicious voter with probability close to one.

The above argument assumes the NIZKP is a "proof", i.e., even a computationally unbounded adversary can not convince a verifier to accept a false claim (except for some small probability). Indeed, if OWF exists, every language in NP has a NIZKP [FLS90]. However, the NIZKP obtained using such techniques have some large polynomial complexity, and are impractical. Another way to go is to use the Fiat-Shamir heuristic [FS86], but then we can not claim anything formal, and we probably do not have unforgeability against unbounded adversaries.

Privacy -

If there are at most $t-1$ dishonest trustees then the encryptions are, by definition, semantically secure (based on the ElGamal threshold encryption security).

Coercion-resistance -

We assume a coercer prepared the voter's two ballots and directed him to act in a specific way. We first notice that -

Claim 1 *If one of the paper ballots the voter prepares is not legal the voter is caught with probability close to one. Also, if one of the two ballots the voter prepares does not contain a vote for each candidate, or, if the voter can not match the corresponding back and front parts of a ballot, then the voter is caught with probability close to half.*

One can argue that probability half is not small enough. However, notice that this means that if a coercer tries to coerce t people, then with high probability (except for probability $2^{-\Omega(t)}$), about $t/2$ of them will be caught, and so with high probability the coercer himself will be detected.
If the voter holds a valid vote for each candidate, and he can associate encryptions with candidates, he can, in particular, vote to any candidate he likes. We now show he can not prove to the coercer what choice he had made. After the voter leaves the booth, the private channel transcripts are shredded. An outsider only sees the published information on the public board which contains the voter's selected bit b, the published re-encrypted vote and one set of re-encryptions which is opened in full (and so is independent of the value c). In fact, the third item can be efficiently simulated, and so does not add any information. The first item, the selected bit, can be chosen in any way the coercer directed. The second item, the re-encryption of the actual vote, is an ElGamal re-encryption of one out of k votes and using randomness and keys that the coercer and the voter do not have. Thus, this re-encryption is computationally indistinguishable from re-encryption of any of the other votes. In particular, the voter can claim he sent any c and the coercer will accept with the same

probability.

Our protocol uses re-encryption (given by both the voter and the booth) for privacy and coercion-resistance, and NIZKP (from both the voter and the booth) to deal with active attacks. One should wonder at this point how do we sustain coercion-resistance when using NIZKP that are not receipt-free. The answer is that the whole process is receipt-free, because a zero-knowledge proof is given for *each* candidate. A voter that has a vote for each party, does not reveal anything about his final preference.

Robustness -

Here we check what happens when players are caught cheating or stop cooperating. A voter might act maliciously by accusing a voting booth for no reason. A booth might violate the protocol. Both can stop cooperating. In either case, we can find the cheating party by examining the recorded conversation between the voter and the booth. In the case of the trustees, we need that at least t trustees will work all the way till the end of the protocol, as in the original protocol.

Unreusability and eligibility -

For both these properties we rely on the physical identification of a voter with an ID. Given this an auditor can use the information about a voter coming to vote (that appears signed by the committee on the public board), and the voting records (we mention again that the selected vote is placed on the public board along with the identity of its voter, and is signed by the booth), to verify that every voter voted only once, and there are no unaccounted votes.

Efficiency -

The voter has to compute a constant number of ElGamal encryptions and NIZKP, and similarly the booth (per vote). Each trustee decrypts only one message and publishes only one NIZKP. Another interesting benefit is that we do not require any public key infrastructure from the voter.

We now look at the voter's actions in the booth. He gives the front sides of the ballots and the back side of the selected ballot. He also needs to verify that the published values are correct, and for that he does a simple visual comparison between the given and the published strings. Similarly, he has to verify that c_0, c_1 match the published values.

Bare-handedness -

We look at the voter's actions in the booth. The voter gives the front sides of the two ballots and the back side of the selected ballot. The voter then picks his vote c by looking at the back side of his remaining ballot and choosing the row number of the candidate he supports. Then, the voter selects a random bit. Finally, the voter has to compare two integers (each between 1 and D) for checking the booth. We believe all of this can be done by humans without the help of a computing device.

5.4 Physical requirements

5.4.1 The physical assumptions are problematic

We now turn to discuss in detail the two physical assumptions underlying our protocol. We begin with the public-board assumption. Let us distinguish between two variants:

Stronger assumption : The public board is publicly visible to *everyone, everywhere.*

Weaker assumption : The public board is publicly visible to *everyone.*

In our protocol we use the stronger assumption, e.g., when we assume the voter can check that the value the booth uploads to the public-board matches the value he holds.

Most current protocols, however, only assume the weaker assumption. To compensate for that they use a signed receipt that the voter can later on check against the public board (e.g., in Chaum's visual scheme and in Prêt *à* voter [CRS05] and many other protocols). However, as we noted above, there is no digital signature which can be verified by a bare-handed voter. As explained above, verifying the signature only after the event has the disadvantage that disputes can not be settled. E.g., if a voter claims the information on the public board does not match what was sent to him on the recordable, private communication channel, and that the signature that was given to him is not valid, then there is no way to determine whether the booth is cheating, or the voter is falsely trying to frame the booth.

Our protocol is not different to the other protocols and could also use both variants. Possible ways for implementing it are:

- Signatures on special paper that the booth can not later deny, or,
- Big screens that can be seen by both the voter and the voting committee.

However, it would be much simpler if we could weaken the requirement, making it easily feasible.

The same situation occurs with the recordable, private communication channel. Such a channel implicity appears in many previous bare-handed protocols. We have two requirements from such a communication channel:

- (Recordability) At the request of one of the participants the channel can be examined by an auditor.
- (Privacy) At the end of the conversation, if the two parties agree, the recording is erased and lost.

Often, previous protocols drop the recordability requirement. In such case there is no way to settle disputes. For example, in Chaum's visual scheme the privacy property of the channel is preserved, and is reflected by the fact that the voter shreds one of the two papers given to him. The recordability property, on the other hand, is not preserved, and as a result the booth can ignore the voter's selection and do anything he wants, and the voter has no way of proving it. As before, signatures do not help here due the bare-handedness (the booth can give an illegal signature as a receipt).

5.4.2 A practical version using shredding

We now modify the protocol, simplifying the interaction between the voter and the booth with the goal of demanding less from the private, recordable channel. Our modification is similar to ideas used in Prêt *à* voter [CRS05] and in the recent Scratch and Vote protocol [AR06].

The modification is as follows: The protocol begins as before. The voter prepares two ballots, one is tested, and the remaining ballot P is used for the actual voting. The booth then prepares, as before, two re-encryptions $P^{(0)}$ and $P^{(1)}$ of P. Here we deviate from the previous protocol. The booth prints a ballot with two columns. The j'th row of the ballot consists of $P_j^{(0)}$ in the left column and $P_j^{(1)}$ in the right column (each with a NIZKP that the re-encryption is legal). The order of the rows in the re-encryptions $P^{(0)}$ and $P^{(1)}$ is the same as the order of P. Also, the values in each column are signed by the booth and covered with a scratch surface (see Figure 5.3).

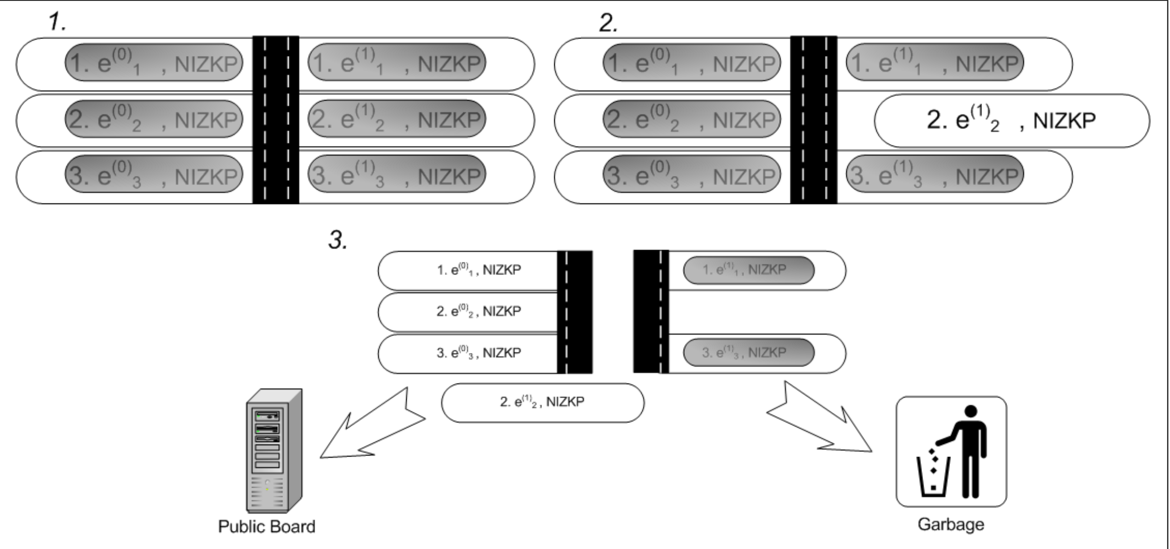

Figure 5.3: 1. The two column are covered. 2. The voter selects a column b, tears his vote and scratches it. 3. The test column is scratched in front of the poll worker. The vote and the test column are sent to the public board. All the rest is shredded.

Next, we do the following:

- The voter picks a random $b \in \{0, 1\}$, scratches off the row of his candidate in this column (recall, that this is determined by the row of the candidate in the back side of P) and publishes it as his vote.[4]

[4]If we assume the communication with the poll-worker is public, then the voter also separates all the rows of his chosen column. He does that in order to hide his chosen row from outsiders.

- The voter surrenders the other unscratched pieces to the poll-worker. He shows the poll-worker that only one piece (that of his candidate) is scratched.[5]
- The remaining pieces of column b are shredded. Also, the voter (or the poll worker) scratches off the other column. He publishes it and takes it home as a receipt. The booth reveals the randomness used to create the re-encryptions in this column, and the *auditors* check correctness.

The protocol is similar to the one described in Section 5.3 but what we gain here is a simpler interaction. In particular, we get rid of the recordable, private channel because the interaction is just from the booth to the voter. Also, we use scratch surfaces in order to prevent random coercion.

One problem that exists with this protocol is that we can not settle disputes. Consider for example the scenario where the auditors discover the information in the scanned column is inconsistent with the data the booth publishes, and the booth claims the voter did not scan the information the booth sent him. The protocol does not supply a way to determine whether the voter is honest and the booth is dishonest or vice versa. As discussed before, this problem implicitly appears in all previous protocols (and in particular in [CRS05] and [AR06]) and is a reflection of the fact that the channel that we use is not private, recordable channel. There are several pragmatic suggestions for solving this problem (e.g., the booth prints its data on a special paper, or, the booth signs on the back of each vote and those signatures are tested before and after the voting).

Other than the problem discussed above (which is common to other protocols in the area) the protocol enjoys the same properties as the one in Section 5.3. Thus, our protocol is as practical as the other protocols in the field, while enjoying true privacy even with respect to the booth.

[5]The reason the voter has to show a poll worker that all other rows are still covered is to avoid vote-buying. Otherwise, a voter can be paid for voting with an encrypted value that starts, say, with a specific sequence, effectively forcing a random vote.

5.4.3 Reducing the ballot's size

We can reduce the ballot's size using two techniques from [AR06]: pseudo random number generators and bar-codes.

Many of our computations require long random numbers. First, we can replace the long random numbers with much shorter numbers used as seeds for a pseudo random generator. The generator's results are used as the required long random numbers. Then, we can use bar-codes to replace string format.

Chapter 6

Conclusions

Chaum and Neff introduced bare-handed voting. Their protocols are coercion-resistance, but in their protocols the voter has to tell the booth what his vote is. We introduce the notion of bare-handed voting with pre-processing and show a simple bare-handed extension of existing protocols. Technically, we use ElGamal re-encryptions over the protocols of [CGS97] or [SK95]. Our technique uses NIZKP (which are not receipt-free) but is still receipt-free because these proofs are given for *all* candidates. This part is similar in spirit to current paper ballots elections. Moreover, testing the booth with a cut-and-choose test is common practice and appears in many protocols. We use a similar idea here to test also the voter, and therefore we get privacy (even against the booth), coercion-resistance and unforgeability.

6.1 Future work

Several challenges remain open. Our protocol (similar to previous protocols) requires two physical assumptions: first, that a public-board is available, and second, that some kind of private, recordable channel exists between the booth and the voter.

Another interesting problem is: can we build a human-verified digital signature (a signature which can be checked by a human without a computer device)? This problem relates to the public-board realization. If we could build such a public-board which anyone can see anywhere, then we do not need human-verified signatures (because other authorities, with computers, could check it and publish their results on the public board). Similar, if we could find such a signature, then a signed receipt will be enough and could replace the stronger variation of a public board.

Other challenges concern to our protocol specifically: Can we ease the voter's part in the voting process? Can we support write-in ballots? Can we relax our requirements from the public-board?

Last, the biggest challenge of electronic voting is still open: find an unforgeable and private electronic voting protocol which the average voter can easily use. The voter may use any reasonable devices (e.g., a smart-card or big screens) but still be confident that his vote is properly casted and counted.

Bibliography

[Ano07] Anonymous reviewers, 2007.

[AR06] Ben Adida and Ronald L. Rivest. Scratch and vote: Self-contained paper-based cryptographic voting. In *Workshop on Privacy in the Electronic Society (WPES)*, pages 29–40, 2006.

[BFP$^+$01] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *PODC*, pages 274–283, 2001.

[BT94] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections. In *STOC*, pages 544–553, 1994.

[CG96] Ran Canetti and Rosario Gennaro. Incoercible multiparty computation. In *FOCS*, pages 504–513, 1996.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, pages 103–118, 1997.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[Cha82] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.

[Cha04] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security and Privacy*, 2(1):38–47, 2004.

[Cha07] David Chaum. Punchscan, http://punchscan.org, April, 2007.

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO*, pages 89–105, 1992.

[CRS05] David Chaum, Peter Y. A. Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In *ESORICS*, pages 118–139, 2005.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, pages 13–25, 1998.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *FOCS*, pages 308–317, 1990.

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *ASIACRYPT*, pages 244–251, 1992.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1985.

[Gro03] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. In *PKC*, pages 145–160, 2003.

[Har03] Beverly Harris. *Black Box Voting: Vote Tampering in the 21st Century.* www.blackboxvoting.org, 2003.

[Hir01] Martin Hirt. *Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting.* PhD thesis, ETH Zurich, 2001.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT*, pages 539–556, 2000.

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

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

[KSRW04] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy*, pages 27–40, 2004.

[KSW05] Chris Karlof, Naveen Sastry, and David Wagner. Cryptographic voting protocols: A systems perspective. In *USENIX Security*, pages 33–50, 2005.

[LTR$^+$06a] David Lundin, Helen Treharne, Peter Y. A. Ryan, Steve Schneider, and James Heather. Distributed creation of the ballot form in pret a voter using an element of visual encryption. In *Workshop On Trustworthy Elections (WOTE)*, 2006.

[LTR$^+$06b] David Lundin, Helen Treharne, Peter Y. A. Ryan, Steve Schneider, James Heather, and Zhe Xia. Tear and destory: chain voting and destruction problems shared by pret a voter and punchscan and a solution using visual encryption. In *Workshop on Frontiers in Electronic Elections (FEE)*, 2006.

[MBC01] Emmanouil Magkos, Mike Burmester, and Vassilios Chrissikopoulos. Receipt-freeness in large-scale elections without untappable channels. In *IFIP Conference on Towards The E-Society (I3E)*, pages 683–694, 2001.

[MN06] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *CRYPTO*, pages 373–392, 2006.

[Nef01] Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *ACM Conference on Computer and Communications Security (CCS)*, pages 116–125, 2001.

[Nef04] Andrew Neff. *Practical High Certainty Intent Verification for Encrypted Votes, www.votehere.com/vhti/documentation/vsv-2.0.3638.pdf*, 2004.

[NS94] Moni Naor and Adi Shamir. Visual cryptography. In *EUROCRYPT*, pages 1–12, 1994.

[Oka98] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Workshop on Security Protocols*, pages 25–35, 1998.

[Ped91] Torben P. Pedersen. A threshold cryptosystem without a trusted party. In *EUROCRYPT*, pages 522–526, 1991.

[Ped92] Torben P. Pedersen. *Distributed Provers and Verifiable Secret Sharing Based on the Discrete Logarithm Problem.* PhD thesis, Aarhus University, 1992.

[Pfi94] Birgit Pfitzmann. Breaking efficient anonymous channel. In *EUROCRYPT*, pages 332–340, 1994.

[Rey05] David J. Reynolds. A method for electronic voting with coercion-free receipt. In *Workshop on Frontiers in Electronic Elections (FEE)*, 2005.

[RS07] Ronald L. Rivest and Warren D. Smith. Threevotingprotocols: Threeballot, vav, and twin. In *Electronic Voting Technology Workshop (EVT)*, 2007.

[RSA83] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 26(1):96–99, 1983.

[Rya05] Peter Y. A. Ryan. A variant of the chaum voter-verifiable scheme. In *Workshop on Issues in the Theory of Security (WITS)*, pages 81–88, 2005.

[Sch99] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *CRYPTO*, pages 148–164, 1999.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth. In *EUROCRYPT*, pages 393–403, 1995.

[Tho] Thomas edison biography at http://www.thomasedison.com/biog.htm.

[Wik07a] Wikipedia. 2004 united states presidential election controversy, voting machines, October, 2007.

[Wik07b] Wikipedia. Electronic voting examples, October, 2007.

[ZS93] Yuliang Zheng and Jennifer Seberry. Immunizing public key cryptosystems against chosen ciphertext attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):715–724, 1993.

Appendix A

Publicly verifiable secret sharing

A publicly verifiable secret sharing scheme has two improvements over the secret sharing scheme from Section 3.2: anyone can verify that the dealer sent valid shares, and, when a player reveals his share, anyone can verify that it is his real share. We show a *Publicly Verifiable Secret Sharing* protocol, presented by [Sch99]:

- A group G_q of prime order q and two generators $g, h \in G_q$ are publicly selected (e.g., by choosing random primes p and q such that $q|p-1$ and taking G_q to be the unique subgroup of $\mathbb{Z}_p^*$ of order q).
- Each player P_i $(i = 1 \ldots N)$ selects a secret key $x_i \in_R \mathbb{Z}_q^*$ and publishes a public key $h_i = h^{x_i}$.
- Let s be the secret that the dealer wants to share. The dealer picks a random $t-1$ degree polynomial with coefficients in $\mathbb{Z}_q$ and set $a_0 = s$ -

  $$\alpha(x) = \sum_{j=0}^{t-1} a_j \cdot x^j \tag{A.1}$$

  He publishes t commitments $C_i = g^{a_i}$ for $i = 0 \ldots t-1$, and, encrypted shares $S_i = h_i^{\alpha(i)}$ for $i = 1 \ldots N$.

  He proves using a non-interactive ZKP that for each $i \in [1..N]$ that -

  $$log_g(\prod_{j=0}^{t-1} C_j^{i^j}) = log_{h_i} S_i \tag{A.2}$$

We note that $\prod_{j=0}^{t-1} C_j^{i^j} = \prod_{j=0}^{t-1} (g^{a_j})^{i^j} = \prod_{j=0}^{t-1} g^{a_j \cdot i^j} = g^{\sum_{j=0}^{t-1} a_j \cdot i^j} = g^{\alpha(i)}$ and therefore fulfilling equation A.2 proves that the published commitments and shares are valid.

When t players wish to reconstruct the secret, each publishes $s_i = S_i^{1/x_i}$ and a non-interactive ZKP that $log_h h_i = log_{s_i} S_i$ in order to prove that his share is correct. Lets denote by $P_{k_1}, \ldots, P_{k_t}$ the group of t players. Now, the secret can be calculated using lagrange interpolation, but instead of summing s_i-s we multiply them in the next form -

$$\prod_{j=1}^{t} s_{k_j}^{\lambda_{k_j}} = \prod_{j=1}^{t} (h^{\alpha(k_j)})^{\lambda_{k_j}} = h^{\sum_{j=1}^{t} \alpha(k_j) \cdot \lambda_{k_j}} = h^s \tag{A.3}$$

where the λ_{k_j}-s are the corresponding lagrange coefficients.