UNIVERSITY OF ATHENS

MSc THESIS

Secure multi party computations for electronic voting

Author:
Panagiotis Grontas

Supervisor:
Assistant Professor Aris PAGOURTZIS

A thesis submitted in fulfilment of the requirements for the degree of Master Of Science

in the

Graduate Program in Logic, Algorithms and Computation
Department of Mathematics

Examination Committee

Assistant Professor Aris Pagourtzis

Professor Stathis Zachos

Assistant Professor Aggelos Kiayias

Athens, May 2014

Η παρούσα **Διπλωματική Εργασία**

εκπονήθηκε στα πλαίσια των σπουδών

για την απόκτηση του

Μεταπτυχιακού Διπλώματος Ειδίκευσης

στη

Λογική και Θεωρία Αλγορίθμων και Υπολογισμού

που απονέμει το

Τμήμα Μαθηματικών

του

Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών

Εγκρίθηκε την 7η Μαΐου 2014 από Εξεταστική Επιτροπή

αποτελούμενη από τους:

Ονοματεπώνυμο	**Βαθμίδα**	**Υπογραφή**
Αριστείδης Παγουρτζής	Επικ. Καθηγητή	……………………
Ευστάθιος Ζάχος	Καθηγητή	……………………
Άγγελος Κιαγιάς	Επικ. Καθηγητή	……………………

UNIVERSITY OF ATHENS

Abstract

Graduate Program in Logic, Algorithms and Computation
Department of Mathematics

Master Of Science

Secure multi party computations for electronic voting

by Panagiotis Grontas

In this thesis, we study the problem of electronic voting as a general decision making process that can be implemented using multi party computations, fulfilling strict and often conflicting security requirements. To this end, we review relevant cryptographic techniques and their combinations to form voting protocols. More specifically, we analyze schemes based on homomorphic cryptosystems, mixnets with proofs of shuffles and blind signatures. We analyze how they achieve integrity and privacy in the voting process, while keeping efficiency. We examine the types of social choice functions that can be supported by each protocol. We provide two proof of concept implementations. Moreover, we review ways to thwart stronger adversaries by adding receipt freeness and coercion resistance to voting systems. We build on the latter concept to propose a modification to a well known protocol. Finally, we study two actual e-Voting implementations namely Helios and Prêt à Voter .

Keywords: Electronic Voting, Homomorphic Cryptosystems, Proofs Of Knowledge, Mixnets, Blind Signatures, Coercion Resistance

Περίληψη

Στην παρούσα εργασία, μελετούμε το πρόβλημα της ηλεκτρονικής ψηφοφορίας. Θεωρούμε ότι είναι έκφανση μιας γενικής διαδικασίας αποφάσεων που μπορεί να υλοποιηθεί μέσω υπολογισμών πολλαπλών οντοτήτων, οι οποίοι πρέπει να ικανοποιούν πολλές και αντικρουόμενες απαιτήσεις ασφαλείας. Έτσι μελετούμε σχετικές προσεγγίσεις οι οποίες βασιζονται σε κρυπτογραφικές τεχνικές, όπως τα ομομορφικά κρυπτοσυστήματα, τα δίκτυα μίξης και οι τυφλές υπογραφές. Αναλύουμε πώς προσφέρουν ακεραιότητα και ιδιωτικότητα (μυστικότητα) στην διαδικασία και την σχέση τους με την αποδοτικότητα. Εξετάζουμε τα είδη λειτουργιών κοινωνικής επιλογής που μπορούν να υποστηρίξουν και παρέχουμε δύο υλοποιήσεις. Επιπλέον ασχολούμαστε με την αντιμετώπιση ισχυρότερων αντιπάλων μη παρέχοντας αποδείξεις ψήφου ή προσφέροντας δυνατότητες αντίστασης στον εξαναγκασμό. Με βάση την τελευταία έννοια προτείνουμε μια τροποποίηση σε ένα ευρέως χρησιμοποιούμενο πρωτόκολλο. Τέλος μελετούμε δύο γνωστές υλοποιήσεις συστημάτων ηλεκτρονικής ψηφοφοριας το Helios και το Prêt à Voter .

Λέξεις Κλειδιά Ηλεκτρονικές Ψηφοφορίες, Ομομορφικά Κρυπτοσυστήματα, Αποδείξεις Γνώσης, Δίκτυα Μίξης, Τυφλές Υπογραφές, Αντίσταση στον Εξαναγκασμό

Acknowledgements

Firstly and more importantly, I would like to thank my supervisor, Professor Aris Pagourtzis for his guidance and cooperation while working on this thesis, for always asking questions that challenged my understanding of the subject, as well as for giving me the freedom to pursue aspects that were intriguing to me personally.

In addition, I would like to thank the other members of the committee, namely Professors Stathis Zachos and Aggelos Kiayias for inspiring me with their teaching. My gratitude is also expressed to all the faculty of the $\mu\Pi\lambda\forall$ graduate program for giving me the opportunity to return to studying after 10 years of absence.

I would also like to thank the members of the CoReLab Crypto Study Group for discussions and for listening to my ramblings.

This thesis would not have been possible without the support of my family.

...To Argiro and Lena

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

In this thesis, we deal with the problem of voting. An informal definition of voting can serve as a starting point. We consider voting as a distributed decision making process where each participant selects one or more items from a collection. Afterwards a computation is performed that outputs the most preferred item(s). All participants must accept the result and adhere to it, until the next time the process is repeated. Voting can be modelled as a *multi party computation system*, since multiple parties submit inputs to a system which in turn does some processing and yields the output.

This definition is kept intentionally broad in order to stress that voting is as old as society, since it is a generic decision making process. In spite of this fact, voting is usually associated with the selection of a government to decide for a number of people, for a specified period of time. Furthermore, its implementation has changed considerably through the ages, adapting to the various prevalent technologies of the times. In this thesis, we will examine voting through the computational lens, which is the prevalent scientific and technological instrument of the 20th century and beyond. In a manner typical to the computer science approach, we will focus on the characteristics of the voting process, its inherent requirements and difficulties, in order to be able to give and apply the appropriate solutions.

Electronic voting is a heated area of research, partly because of the importance of the voting process to society and mainly because of the conflicting nature of the system requirements which is augmented by the notorious unreliability of computer systems. However we must keep in mind that all technological tools that have been used during the ages to implement voting systems have suffered similar problems, but due to the lack of the methodological tools that were available, to the public they were accepted and used in practice for centuries. On the other hand the lens of computer science brings out the incompatibilities of the voting process (that are already there) in order to build general systems to reconcile them. As a result the shortcomings of the voting process seem as shortcomings of computer science and technology, when such a statement could not be any further from the truth. To be specific the exact opposite happens: We have a feel for the hardness of voting, and because of the rigorous analysis that was carried out in order to implement it electronically.

1.2 Requirements for voting systems

The multi party computation in a voting system (electronic or not) must be secure i.e. it should exhibit some characteristics in order to fulfil its goal. In general, it must thwart attacks both from the *inside* and from the *outside*. The attacks might be targeted to a specific individual (retail attacks), or to the voting body as a whole (wholesale attacks). In this section, we introduce the security characteristics, in the line of the framework presented in [68].

1.2.1 Integrity

Integrity, in brief, means the voting system should only relay the choices of the voters to the output computation. It should neither alter nor delete votes, or inject votes that do not correspond to actual voters. This is captured beautifully in the phrases:

- Votes should be cast as intended, which is officially called *ballot casting assurance* [15]. In simple terms, this means that the user interface of the voting system should aid the voter to express his choice without interfering in any way.
- Votes should be recorded as cast, meaning that the voting system should internally represent the vote in a manner that does not deviate from the voter choice.
- Votes should be counted as cast, reflected in the famous saying by Joseph Stalin: *"Those who cast the votes decide nothing. Those who count the votes decide everything".*

Integrity is there to correctly determine the winner, to convince the loser [54] and, maybe more importantly, the integrity requirement can be used to give confidence to the participants that the result correctly expresses their will. In order to convince that integrity is preserved the voting system must provide *credible evidence* [54] that can be utilized with the following processes:

- **Individual verifiability.** Each voter must be able to verify that his selection was correctly included in the outcome.
- **Administrative verifiability.** The organizers of the election can verify that everything went as planned [16]. This is because the election official act as trusted proxies of the individuals, so their actions are on their behalf.
- **Universal verifiability.** Any interested party should be able to verify that the result computation was correct, i.e. all the valid inputs and only the valid inputs were included in the output without any change and that the computation adhered to the specification. The observant reader should immediately see that a voting system that provides universal verifiability, also provides administrative and individual verifiability.
- **Eligibility verifiability** Everybody is able to verify that every ballot cast, originated from a voter with the right to vote. This type of verifiability was defined by Kremer et al. in [86]. Until then it was considered part of universal verifiability, but it can studied separately.

A more anthropomorphic manifestation of integrity is *fairness*. The output of the computation should not only be correct, but it should also be fair. This means that all inputs from participants should be treated exactly as the computation specification dictates. This should happen from the moment they cast their votes. It follows that intermediate results should not be available, since they can affect some inputs. Moreover, it should not be allowed to a participant to vote in such a way as to cancel out another input (*vote cancellation*), or it should not be acceptable, that a voter votes exactly as someone else (*vote duplication*). A major consequence of fairness, is that the elections cannot be repeated, since by definition the conditions are not the same. This places additional burden to the security requirements.

1.2.2 Ballot Secrecy

The concept of ballot secrecy is the most important feature of a voting system, at least in public perception. The privacy of the input to the system guarantees that it conveys the true belief of the voter without any influence from external agents. A related yet distinct notion is that of *voter anonymity*, meaning to remove the correlation between the voter and vote. Anonymity is often referred to as unlinkability [90]. Conceptually ballot secrecy and voter anonymity are different, but an implementation might use one to achieve the other. Anonymity can be achieved through ballot secrecy if the result is computed on the secrets. Vice versa, ballot secrecy might not matter if voter anonymity holds, since the latter allows the voter to express his/her will. An important remark here is that ballot secrecy cannot be totally achieved since the result of the computation might leak some information, as in the case of the unanimous vote. Even if only a single voter begs to differ, she is entitled to it and must be protected. Generalizing the above observation, we must note [40] that any corrupted subset of voters can collaborate and deduce the subtally of the honest ones. However no more should be deducible from the outcome.

In the simplest case, privacy means that no further information is deducible from the votes and the tally alone. A more elevated threat is that of *coercion*, that might be involuntary or voluntary. In the first case a voter can be threatened to vote in a particular way, so as not to face dire consequences in the personal, family, professional or societal front. In the second case a voter might volunteer to sell her vote and be rewarded in some fashion. In the more general case [77] the voter might be impersonated by giving up his credential or in the most brute force case the voter might be forced to abstain from voting altogether. In [17] it is argued that little can be done in order to thwart a determined attacker that coerces before the election, as modern technology can beat even the most sophisticated scheme. However post election coercion can be dealt with.

Ballot secrecy hardens coercion, as it makes it difficult for the coercer to validate that the party being coerced carried out its part as was agreed. However individual and universal verifiability can aid the coercer. Since we would like the voter to be able to verify his vote, the system must enable it by somehow providing an indication that can be utilized for verification. But this receipt can also enable the coercer to verify that the coercion was carried out as agreed. As a result, to strengthen ballot secrecy voting systems should also provide *receipt freeness* and *coercion resistance*. Coercion resistance is formally defined in [77] as a game where the attacker must guess whether the victim complied to his directions and has negligible advantage over a random guess.

We see here that two requirements in a voting systems are contradicting. This is a general pattern in voting and it is the main reason that it is such a hard problem to solve, as it leads to complexity.

Another aspect of ballot secrecy that must be taken into account is its duration. It is debated whether, it should be everlasting or apply for a specified period of time. In many implementations that we will see in this thesis, ballot secrecy is conditional on the premise that it depends on the solution of computational problems that currently seem infeasible. However, in 50 years this might not be the case. In some regimes this might lead to negative consequences if the revealed votes are not favorable at the time.

1.2.3 Authentication and Authorisation

The requirements in this section allow voting to fulfill its claim as a decision making process. In more simple terms it allows voting to actually implement the *democracy* [90] criterion, and not merely claim to implement it. To begin with, voting may be a general way to reach a decision, however not everybody should be able to participate. Thus there must be a way for the eligible voters to stand out. *Authentication* serves this purpose. However authentication requires some form of identification and this might contrast the anonymity requirement. As a result solutions can identify the largest possible subset for each participant. The next logical step is *authorisation* which makes sure that the participant behaves as is expected so as not cheat the system. For instance, a voter should vote as many times (usually one) as the election law mandates and no more.

1.2.4 Enfranchisement

All participants eligible to vote should be encouraged to participate. This is easier said than done. For example, in order to be authenticated, voters should issue some form of credentials. These excludes the people that cannot issue these credentials for some reason. In addition, voters should not be intimidated by the voting system and be discouraged to participate. This is particularly true for electronic voting, since technologically illiterate people might find it hard to use the system. Of course solutions should respect the previous requirements. Assistance in voting for example might solve the problem, if ballot secrecy is maintained. Enfranchisement implies that voters trust the system to compute their intent. Complex voting systems might succeed in implementing some or most of the requirements but might fail in convincing the voters. We do not address, the question if the solutions described in this thesis, actually convince voters.

1.2.5 Availability and Efficiency

The voting system should always be available to receive input from the participants and it must output the result of the computation in a reasonable time. Availability is crucial since the repetition of the vote casting phase, sheds doubts on fairness. Availability in the presence of an active and persistent adversary is often called *robustness*. The adversary might pose as a voter or as the authority or both. In addition, lack of availability hinders enfranchisement, since a voting system that has 'ups and downs' in its operation, is perceived as untrustworthy. It is very important to point out that a voting system must be robust in its complete lifecycle and not only on the vote casting stage. For example, a verifiable voting system must have a detailed process to deal with 'alleged' verification problems, or else verifiability can be used against the system.

Efficiency is one of the reasons put forth by electronic voting proponents, especially when the voting population is extremely large. It refers to the resources used by the voting system in all its workflow. Such resources are referred to as *cost* and might be time, money, requirements on infrastructure and people participation etc.

1.3 A brief history of voting

Since voting is a universal decision making process, it can be encountered in all periods of recorded human history. In this section we shall review some of the more widely used voting methods and analyse the from a technological viewpoint as well as the requirements laid out in the previous section. A lot of material presented here, has been drawn from [68] and [54]. Our goal is to demonstrate that voting is aligned with the prevalent technologies of each era, however imprefect they may be.

1.3.1 Early Voting

The simplest and probably one of the earliest methods of voting is done by *a show of hands*, i.e. by raising one's hand if one agrees with the proposition put forth. This method is binary in nature, since the voter has two options, to show or not to show. Tallying takes place by counting raised hands in the presence of all voters. This method guarantees that votes are cast as intended, but counting can be accurate only if the deciding group is small and its composition is fixed. However, universal verifiability is easier to achieve since everybody can count raised hands. It is interesting to note that while this method provides evidence, they are of an ephemeral nature. In case of a dispute the process must be repeated and it is not guaranteed that each voter will behave exactly the same (repetition is not fair). This method provides efficiency in small groups only and enfranchisement since voting is a very easy action without any requirements. The last characteristic is very important and probably the reason that this method is so often used, even today.

However this method has serious drawbacks. First of all it does not provide any secrecy. Everybody can see what the next person has voted. This fact is essential to tallying. This opens up and avenue of coercion and blackmail opportunities. However a very important observation can be made. As this scheme is very easy to implement it can be used for impromptu elections, thus minimizing the preparation opportunity of the coercer. On the other hand it does not thwart coercion after the fact. Another issue with this method is that it is not usually used with any method of authorization or authentication. We have personally witnessed occasions were voting by show of hands has taken place in rooms with open doors and people freely walking in and out. Of course many more can go wrong, as since no authentication is provided. For instance somebody can inject voters at will, thus altering the result.

A similar method in simplicity and use is voice voting (*Viva Voice*). As its name suggest voting takes place by voicing the name of the desired selection (candidate or course of action). As sound is transient the vote must be recorded in order to be counted. A more primitive form of voice voting takes advantage of the amplitude characteristics of sound, as voters orally cast their votes. The sound with the higher amplitude wins. However in the majority of cases the votes are recorded in a more permanent medium such as paper, by an election official.

This method is characterized by individual verifiability as the voter can check whether his vote was correctly recorded. However it lacks universal verifiability as after the recording process nobody can check if the recorded votes correspond to the votes cast. This can be corrected by allowing representatives of conflicting parties to witness the counting. As with the show of hands this method requires no particular effort or abilities for voters and as a result promotes enfranchisement. Counting is efficient only in small groups. Authentication is also provided as an election official records the vote probably in the presence of many witnesses. If this method is used in well established communities, an impostor will be surely recognized. It is evident that this method has serious drawbacks analogous in nature with the show of hands methods described previously.

1.3.2 Paper ballots

The drawbacks of the previous methods were apparent even to its users. As a result a replacement was sought, that would provide mainly secrecy and permanency in the form of physical storage. Many candidate technologies were tried, that culminated in what is now called the *secret ballot* or the *Australian ballot*, the golden standard of voting [17], which was adopted in the late 19th century. An example of such technologies were colored balls placed in an opaque container. Analogues of paper ballots were even found in ancient Greece, where the names of people to be ousted from the city of Athens were written on sea shells, a practice referred to as ostracism. A reason often cited for the delay to the whitespread use of this methodm is that paper ballots would hinder enfranchisement as the literacy of the general public was poor.

Process The general process of voting by paper ballots as used today has evolved in many ways, incorporating other societal changes. As these changes vary by country, the process has different versions in many parts of the world. However a common base line consists of the following phases:

- **Preparation**

  - Voters register and their identity is validated. In some countries this is done when a person receives citizenship, or even when she is born. In other countries registration is done per election. As a result, a catalog of registered voters is prepared.
  - Special places are designated as polling stations. Usually public schools serve this purpose as these places must be easily accessible.
  - The actual paper ballots are printed and transported to the polling stations. The paper ballots must be identical.
  - Containers for the ballots are distributed. These containers come in 2 forms. The first form serves the purpose of providing ballot secrecy and can be a simple envelope. The second container, usually called the*ballot box*, gathers all the ballots cast. As such its operation is essential for the integrity of the election. For example it must be difficult to extract ballots cast before the elections end. In addition the ballot box must be empty in the beginning and must not be allowed to open before the election ends. Consequently, the ballot box must be locked, sealed, transparent and supervised.
  - Personnel must be appointed to handle the preparation process, i.e. seal the ballot boxes, distribute the ballots, prepare the polling station. This trusted personnel includes 'impartial' observers as well as members of all the parties that have an interest in the integrity of the election. Trust is achieved by conflict of interest.

- **Vote Casting**
  - As voters proceed to vote they are identified and authenticated, using some token obtained from the registration phase.
  - They receive a copy of all the ballots and an empty envelope, which aims to provide ballot secrecy. In order to hide the selection and marking process the voter enters a booth, *alone*, where he can secretly select the desired ballot, mark it and place it in the envelope. There he must safely discard the rest of the ballots. Observe that ballot secrecy is mandatory. The voter cannot deny it, even if he wants to. This presents a huge obstacle to coercers. The booth provides secrecy even if the voter does not want it [18].
  - The voter exits holding only the envelope and casts it in the ballot box.
  - The vote casting phase lasts a predetermined period of time, usually from dawn to dusk.

- **Vote Tallying**
  - After the vote casting phase the ballot boxes are unsealed for ballot counting to begin.
  - The sealed ballots are counted and for each one the envelope is removed.
  - The opened ballot is examined and its contents are recorded. Ballots marked improperly are annulled.
  - This process is supervised by the trusted and mistrusting individuals appointed in the preparation phase. As an additional measure to preserve integrity, it is checked the total number of voters authenticated must match the total number of sealed ballots and the total number of votes (including the annulled ones).

- The vote tallying phase happens in parallel in all voting stations. The partial results must then be combined to produce the final tally. It is vital that aggregate values reported must correspond to the actual results of counting.

As this process was being constantly refined it has many advantages. First of all people are accustomed to it and trust it. Another important general advantage is that its security is guaranteed by constitutions that have evolved around it. For example, if a person tries to escort a voter inside the voting booth, he might be arrested on the spot. Observe that this process, does not entirely depend on paper, it has been built around it. However, the versatility of paper, as a vote registering medium, is an additional advantage. Anything can be written on paper and the voters can mark it as they wish. As a result multiple election methods can be supported. It is even possible that the voter, bypasses entirely the proposed candidates and freely vote for his own, by simply writing them down. This practice is called voting with *write-ins* and is applicable in many parts of the world.

Such ballots though are not without problems. An inherent problem that may not be attributed to malice is that the counting process is *inaccurate*. Another problem is that the existence of many of the desired properties of elections described earlier are not verified, but trusted upon. For example a voter cannot be sure that her vote was counted, as the vote is anonymous and she is not present during the counting process. Of course she can trust that the sums of voters and votes agree, but she cannot prove it. Consequently, various attacks have been proposed and successfully carried out. One such attack, *chain voting* bypasses the secrecy mechanism to achieve coercion of individual voters. If the coercer can smuggle a ballot during the preparation phase, he can mark it and hand it to his victim. Upon exit the voter must provide proof of coercion in the form of an identical but unmarked ballot. Another successful attack, firstly used in Italian villages, that we describe later in detail, can be used to break anonymity utilizing unused candidates.

Another substantial problem with paper ballots is that the tallying phase is quite inefficient. In small countries tallying usually finishes overnight, but in larger ones tallying by hand can last for weeks. The duration of the vote counting phase must be as small as possible in order to avoid the revelation of partial results, that may affect the acceptance of the result leading to dire consequences. The process can be sped up by adding more counters but that simply transfers inefficiency to another area, that of cost.

1.3.3 Mechanical Voting

The advent of the industrial revolution and the first machines, brought the cutting edge technology of that era to voting. Mechanical voting machines where introduced to speed up counting and improve accuracy. But these machines subtly changed the semantics of voting. Voting would by done by *proxy*, as the voter authorized the machine to record his vote. The essential difference with paper is that it is considered a passive medium as it does not implement any kind of process, as is the case with voting machines. Mechanical contraptions could have a static behavior. When they were introduced this was seen as a desirable feature, since a machine was considered impartial, contrary to humans.They resembled a mechanical voting booth that recorded votes using levers and gears. There were mechanisms in place to ensure that no double voting would occur. The vote would be stored inside the machine and no copy would be printed. After each vote the machine would update a running total, making the counting of votes unnecessary.

The biggest problem with mechanical voting machines is that they did not provide verifiable evidence. At first this didn't seem to matter as one of their aims was to do away with untrustworthy human counters. During their use however, breakdowns would occur. The machines would stop updating totals, thus losing many votes. Such failures might be silent, as the official sometimes could not be sure if the machine had stopped functioning any some time. A smaller problem with this kind of machines was that it was quite difficult for their designers to ensure that they would accept all types of ballots possible in an election to achieve re usability. The machines were tailor made for the particular elections.

A variation of mechanical voting machines would appear in the 1950s with punched card voting. We note here that lever machines and punched cards, utilized the process that was cemented with the Australian ballot. In punched card voting, the voters would record their votes on special purpose cardboard sheets using a special device that would consist of two parts. The first one resembled a pen and its purpose was to actually make the holes. The second one was 'the user interface', a template that displayed the candidates and guided the voter to cut the hole in the position corresponding to the selection. The voter would slip the card into the template and vote using the pen. The punched ballots would then be transferred to a counting machine, a mainframe computer, which would calculate the tally. Such machines are still used today. Punched card voting offers the same advantages as mechanical voting, while retaining physical proof. The proof however is useless without the template. The most important problem encountered with punched card voting is individual verifiability. Quite often ballot designers would cram a lot of information into a very small space on the ballot, thus resulting in voters getting confused and voting for the wrong candidate. Due to the complexity of the user interface a simple check did not reveal the error. Punched card ballots introduced computers to the voting process.

1.3.4 Electronic Voting

In punched card voting the role of the computer was merely to perform the tally. This was consistent with the view of computers in the 1950s. Computers were used to perform calculations with large amounts of data. Nowadays however the computer's role has changed dramatically, as it is an essential part of everyday life. This change is also reflected in their use in voting. In electronic voting, computers are present in every step of the process, from the casting of the vote to the announcement of the results. Electronic voting comes into many flavors that make up two large categories: the first one reuses the established infrastructure of a typical paper ballot election and the other one merely allows voting from anywhere.

1.3.4.1 Supervised Voting

In electronic voting in polling stations two technologies have been used. We shall give a short description of each and mention their particular short comings. In the next section we shall deal with the problem of electronic voting in a general fashion.

Direct Recording Electronic (DRE) Voting Machines These machines are usually custom built computers that run software that displays ballots, allows the voter to input her selections using some input device (usually touch screen) and stores the vote on a removable memory card. A DRE machine can display many ballot formats and provide accessibility options. Often such a machine can allow the voter to verify the choice by printing a paper receipt like an ATM. However there is usually no option to change the selection after the receipt has been printed. The voter may or may not cast the receipt in a ballot box for parallel counting. After voting has concluded the memory cards are ejected and like punch card voting tabulated by a computer. The memory cards is the easiest attack point for anyone who wishes to tamper with an election. Attacks have been reported were memory cards contain executable code or an initial count of votes before voting begins [129].

Optical Scan Voting Machines In this type of elections, voters mark a paper ballot, which is then scanned by the voting machine. Everything happens electronically afterwards. The votes are stored in memory cards, inside the machine. After election ends the machine prints out its local total and the memory cards are transferred to a tabulator and summed. As in DREs, the original paper ballot is not mandatorily cast in a separate ballot box, to be used for recounting purposes. It is clear to see that the DRE attacks described in [129] and [68] are applicable here as well.

1.3.4.2 Internet Voting

Internet voting is the process where the voter casts the ballot through his personal computer or smart phone and the vote is relayed through the Internet. Unlike supervised voting the voter can cast the ballot in the comfort of their homes or their offices without any official being present. The software used must be platform independent so it usually takes place through a web browser. Internet voting is an example of *remote voting*, a practice that occurs quite often in real elections. An example is vote by mail, where the voter receives the ballots via the postal service or prints them from a web site, fills them at home and mails them back to the election authorities.

Internet Voting has many advantages: Firstly it allows people to make their choices without having to go to the polling station, stand in the queue etc. It can also be accessible from any part of the world, allowing greater turnout and replacing the need for vote by mail. It can also be more usable, as the voting software can provide guidance over the process. As it is implemented in computer software it has significant cost savings in the long run. The administrative costs of setting up an election are also quite low, leading to frequent elections. The proponents of Internet voting have even gone as far as claiming that it will bring back the direct democracy witnessed in classical Athens 2500 years ago!

On the other hand, Internet voting has many other disadvantages. The most important one is that it sacrifices the mandatory imposition of privacy offered in voting booths. Since voting is done without official supervision, it is open to coercion and vote selling as the attacker can be present and oversee the voting process. Moreover, Internet voting enables a weaker kind of coercion to take place: pressure from family or close friends to take place, as the voter is never alone. In addition since the whole process takes place completely in software so must the authentication. This allows for identity theft, an important threat for common Internet transactions.Another important problem is that its security depends on the security of the voters' computer. The security research literature is full of vulnerabilities and attacks leading the safe conclusion that computer client software cannot be trusted for security. Malicious software might alter votes, fail to submit them (without reporting it) and many other thing can go terribly wrong. Things are not as bright on the server side as well. As the voting server resides on an open network it is a target of attacks as well, that are not restricted inside a country, but can come from external opponents as well. Furthermore, the software used for voting might be inaccessible to certain groups of people such as the elderly as they are not comfortable with using a computer.

We can see why many computer security and voting researchers think that Internet Voting is a bad idea [68]. However as we strongly believe that Internet transactions will be the preferred method of interaction in the next 50 years, voting will probably follow. As a result, it is an important challenge to make Internet Voting viable.

1.4 Problems with Electronic Voting

In the previous sections we have described some important problems found in the three major approaches for electronic voting. In this section we shall sum them up, express them in the framework described previously and provide the general framework of the proposed solutions that this thesis will focus on.

To begin with, the main issue of electronic voting is the universal nature of computers. Computers can run any type of software. As a result nobody can be completely sure of exactly what the computer is executing at any moment. To paraphrase the well known Ryle quote: *There are ghosts in the machine*.

As electronic voting is voting by proxy, we delegate our choices to these ghosts and hope that they convey it accurately and secretly. In addition these ghosts implement very complex and conflicting requirements. Software testing is known to be an inexact science and as a result the ghosts are not to blame.

Integrity, to begin with, is difficult to achieve with software. An attacker might inject code to alter the votes cast, transferring them to a candidate of his choice, or to decoy candidates in order to achieve some believable distribution of votes. Alternatively malicious code might cause some votes not to be counted, or lose their proper weight, by initiating the candidate counters to non zero values. One might argue that this can be remedied with the different flavors of verifiability, but they too are dependent on software. For example, in a DRE voting machine the verification receipt, might contain the correct choice, but the memory card might contain an altered one. If universal verifiability includes a recount based only on the contents of the memory cards then the cheating cannot be detected. The moral here is that the production of credible evidence required for integrity, are generated by software. Moreover, it is difficult to guarantee that voting software does not leak intermediate result, that break the fairness premise.

Ballot secrecy and anonymity are also difficult to achieve. DREs provide no ballot secrecy. It is very easy to sort the contents of the memory card with the order of appearance of the voters and deduce their choices. Many systems employ cryptography to provide ballot secrecy, but they must maintain integrity. Of course since the software that implements the cryptographic algorithms, knows of the initial choice, it can intervene and store it unencrypted or relay it to the coercer. Other attacks include the duplication of an encrypted vote, or vote canceling, producing a vote that nullifies an existing one, without of course knowing what it is. Finally as far as anonymity is concerned, how can one draw a non crossing line between authentication and anonymity, if the same system must handle both.

One approach for solving this process has been certification. The producers of the voting system supply the hardware and software specifications to an independent auditor, who runs some tests and decides whether the system meets certain criteria. The problem with certifications, in general, is that the auditor does not evaluate the system as a whole, but tests whether specific claims made by the producer are substantiated or not. As a result it is not essential that the system is evaluated completely but it maybe evaluated in part.

Another approach, a generalization of certification uses the concept of open source software. The creator of voting software releases the source code of the system and independent auditors to analyze it as they wish. It is essentially the universal verifiability approach applied to the election software. This approach was believed adequate in the previous decade, but it suffers from a very important flaw. Nobody can guarantee that the code released and analyzed matches the code that is executed during an actual vote. Of course the same applies for the more restricted form of analysis, namely certification. What must be verified is the performance of the software in relation to the election data.

A more holistic view is needed. Such an approach was proposed in [115]. It is based on the simple idea of software independence which states that the result of an election should not depend on the software that is used to run the election:

Definition 1.1. A voting system is *(weakly) software independent* if an undetected change or error in its software cannot cause an undetectable change or error in the election outcome.

The whole idea is that software problems should not propagate to the election results. To better understand how this can happen, we shall use a simple method to implement it. We refer to the case of DRE machines. Suppose that authorities allow the voter to cast the receipt, printed by the machine, in a ballot box, after verifying that it accurately reflects his vote. In this way a *paper trail* is left behind, which is approved by the voter (such systems possess *voter verifiable paper audit trails* - VVPAT). The resulting system is software independent, because the software can be replaced by a hand count.

Another way to achieve software independence is to use results from cryptography to verify the actions of the system.

A variation of this concept, called strong software independence requires detection and recovery mechanisms to be in place, so that a detected error does not require re running the election. A VVPAT system where the paper trail is considered as an official ballot is strongly software independent. For a system to really by software independent a mechanism must be in place for handling detected verification problems.

Another relevant idea can be summed up in the notion that the elections are the ones needing verification and not the voting system.

1.5 Thesis structure

This thesis' topic is how techniques from cryptography can be used to build electronic voting systems that implement all the features described in this chapter, in a non conflicting manner. We introduce approaches to electronic voting by continuously adding layers one on top of the other. A road map follows:

- In chapter 2 the primitives that a voting system utilises will be introduced. We shall introduce concepts from cryptography and from the theory of social choice. These concepts will be utilized in the rest of the chapters to build larger building blocks to be used in a voting system.
- In chapter 3 we shall introduce the idea of **homomorphic voting systems**. The systems that are described here maintain the ballot secrecy property till the very end, since the actual votes are *never* revealed.
- In chapter 4 we introduce the concept of a **mixnet**, which is a generic tool used to build an anonymous system. It is the electronic counterpart of the traditional ballot box shuffling, that is employed in order to lose track of the correspondence between votes and voters.
- In chapter 5 we describe approaches to electronic voting that can be combined with the before mentioned categories. This chapter can be thought of an enhancement of 4, as the schemes describe offer addons to enhance the privacy requirement. More specifically we deal with the blind signature model and methods to provide everlasting privacy. A large part of the chapter is devoted to methods for countering the threat of coercion. Finally, we combine the approaches presented in this chapter to sketch a novel coercion resistant protocol based on blind signatures.
- In chapter 6 we deal with two implementations of voting systems, namely the Helios voting system for remote elections and the Prêt à Voter voting system for supervised ones. In addition we describe two 'home made' proof of concept implementations of voting protocols.
- In chapter 7 we conclude this thesis and describe important avenues for future work.

Chapter 2

Preliminaries

2.1 Cryptography Building Blocks

Cryptography is the art and science of exchanging secret messages. As such it has an immediate relationship with the ballot secrecy requirement for electronic voting. However the cryptographic techniques employed in electronic voting have become more ambitious. They aim to provide the verifiability requirements that enable the sought trust. It can be argued that the role of cryptography as trust builder is more important than the role of merely keeping secret messages.In this section we shall review the basic building blocks that cryptography provides to a voting system. From the onset we must make it clear that a thorough and rigorous mathematical review of the cryptograpy found in voting systems, isn't even found in classical cryptography textbooks, like [78] Consequently, the review in this chapter, is not about cryptography in itself, but its purpose is to merely lay the ground for the actual focus of this thesis, namely the voting systems that utilise cryptography to achieve the requirements described in Chapter 1. As a result the exposition style, will not be formal in well known cases.

2.1.1 Public Key Cryptography

Cryptography has been used from the ancient times. However all cryptographic systems used had a significant shortcoming. The parties that wanted to communicate securely had to agree to a secret key prior to communicating. All that changed in 1976, when Diffie and Hellman invented public key cryptography [50]. The system proposed, used 2 keys, one for encryption and one for decryption. The encryption key could be made public to enable communication without prior engagement. The decryption key would be kept secret and used by the recipient of the cipher text to recover the original message. More formally [81] a public key cryptosystem is made up of three algorithms:

- **A Key Generation algorithm** that is given a security parameter η and returns the public and private keys.
- **An Encryption algorithm** that is given a message, some randomness and the public key and returns the encrypted message. As we shall see in 2.1.3 any deterministic public key cryptosystem cannot be secure. To this end, randomness is essential to a public key encryption function.
- **A Decryption algorithm** that given the cipher text and the private key recovers the plaintext.

2.1.1.1 The RSA Cryptosystem

One of the most widely used public key cryptosystems was proposed in [113]. It works as follows:

- **Key generation**
  - Select Randomly and Independently Two Large Primes p, q of length η bits
  - Calculate product $n = p \cdot q$
  - Calculate $\phi(n) = (p-1) \cdot (q-1)$
  - Randomly select $e \in \mathbb{Z}_n^*$ st: $gcd(e, \phi(n)) = 1$
  - Calculate reverse $d = e^{-1} \pmod{\phi(n)}$
  - Public key is (e, n) and private key is (p, q, d)
- **Encrypt** The message m is aised to the public key $c = m^e \pmod{n}$
- **Decrypt** The cipher c is raised to the private key $c^d \pmod{n} = m^{ed} \pmod{n} = m$

One way to randomize the encryption function, is by appending random padding to the message. To retrieve the ciphertext without the private key, one must be able to compute $e - th$ roots in $\mathbb{Z}_n$. This is called the RSA Problem. Moreover, if n can be factored than breaking RSA is easy, since we can calculate $\phi(n)$ and recover d using the Euclidean Algorithm.

2.1.1.2 The El Gamal Cryptosystem

Another very popular cryptosystem was proposed in [60] and provides randomised Public Key Encryption from the Diffie-Hellman Key Exchange:

- **Key Generation**
  - Select 2 large primes p, q such that $q \mid (p-1)$ and a generator g of the q order subgroup of $\mathbb{Z}_p^*$ $\mathbb{G}$. Such primes will be henceforth called *safe primes*.
  - Randomly select $x \in_R \mathbb{Z}_q$
  - Calculate $y = g^x \pmod{p}$
  - Return $(pk = y, sk = x)$
- **Encrypt:** Multiply the input message with a randomisation of public key
  - Randomly select $r \in_R \mathbb{Z}_q$
  - Calculate $G = g^r \pmod{p}$
  - Calculate $M = m \cdot y^r \pmod{p}$
  - Return $c = (G, M)$
- **Decrypt** message (G, M) with secret key x is a a function $\mathbb{Z}_p \times \mathbb{Z}_p \times \mathbb{Z}_q \rightarrow \mathbb{G}$
  - return $M/G^x \pmod{p}$

The security of the cryptosystem is based on the difficulty of computing discrete logarithms, which is the problem of computing x given g^x in a cyclic group G. Another way to express this difficulty is throught the Decisional Diffie Hellman Assumption, which states that the tuples (g^a, g^b, g^c) and $(g^a, g^b, g^a b)$ are indistinguishable by an adversary with limited computational power.

The exponential El Gamal variation can be utilised for electronic voting, since it has homomorphic properties 2.1.2. The key generation phase remains the same, but the encryption and decryption phases change:

- **Encrypt** Message m: Raise message to the exponent and multiply with randomisation of public key
  - Randomly select $r \in_R \mathbb{Z}_q$
  - Calculate $G = g^r \pmod p$
  - Calculate $M = g^m \cdot y^r \pmod p$
  - Return $c = (G, M)$
- **Decrypt** message (G, M) with secret key x
  - Return $M/G^x \pmod p$
  - Calculate m from g^m

It is evident that the decryption of exponential El Gamal requires to solve the discrete logarithm problem. This can be done efficiently if we choose the message space with caution.

2.1.1.3 The Paillier Cryptosystem: Origins and Generalisation

The Paillier Cryptosystem was proposed in [107]. It can be said that it is a cryptosystem that is tailor-made for electronic voting, as it has the favourable property of the exponential El Gamal cryptosystem, namely the encryption of a message handles it as an exponent, but without solving a discrete log problem in decryption. Thus, it is interesting to review the evolution of the various cryptosystems that led to [107].

Goldwasser-Micali The forefather of Paillier was the cryptosystem proposed in [63] which is the first provably secure cryptosystem. It can be described as *a proof of concept cryptosystem*, the concepts being probabilistic encryption and semantic security 2.1.3, since encryption and decryption are done bit by bit. Its three phases are described below:

- **Key Generation**
  - Select Randomly and Independently Two Large Primes p, q
  - Calculate the RSA modulus $n = p \cdot q$
  - Select $y \in \mathbb{Z}_n^*$ with $gcd(y, n) = 1$ and more over without any $x \in \mathbb{Z}_n^*$ such that $y = x^2 \pmod n$. Such a y is called a *quadratic **non**-residue*.
  - The public key is N, y
  - The private key is p, q
- **Encryption**
  - For a message $m = m_1 \cdots m_k$ the ciphertext is produced bitwise:

$$c_i = \begin{cases} yr_i^2 \pmod n \text{ if } m_i = 1 \\ r_i^2 \pmod n \text{ if } m_i = 0 \end{cases}$$

    or more compactly: $c_i = y^{m_i} r_i^2 \pmod n$
- **Decryption**
  - When the recipient receives the ciphertext $c = c_1 \cdots c_k$ the decryption is performed bitwise:

$$m_i = \begin{cases} 1 \text{ if } c_i \text{ is a non quadratic residue} \pmod n \\ 0 \text{ if } c_i \text{ is a quadratic residue} \pmod n \end{cases}$$

This operation can be easily performed by the receiver since she knows the factorisation of n.

The security of [63] rests on the quadratic residuosity problem, namely that it is difficult to discern if a $x \in \mathbb{Z}_n^*$ is a quadratic residue without known the factorisation of n.

Benaloh cryptosystem The next step in the evolution of cryptosystems that led to Paillier was the cryptosystem proposed in [40]. This paper directly relates the proposal with the problem of electronic voting. As a result we shall examine it in more detail in the following chapter. In this section we shall deal with the cryptographic details. The cryptosystem proposed in [40] generalises the quadratic residue approach of [63] using $r - residues$. A number y is an $r - residue \pmod n$ if there exists x such that $y = x^r \pmod n$. If the factorisation of n is known, then there is a polynomial algorithm to recognise if a given number is an $r - residue$. Otherwise the problem is considered hard.

- **Key Generation**
  - Select a prime number r
  - Select Randomly and Independently two large primes p, q such that $r \mid (p-1)$ and $r \nmid (q-1)$
  - Calculate the RSA modulus $n = p \cdot q$
  - Select $y \in \mathbb{Z}_n^*$ with $gcd(y, n) = 1$ and more over without any $x \in \mathbb{Z}_n^*$ such that $y = x^r \pmod n$. Such a y is called a *quadratic **non**-residue*.
  - The public key is N, y
  - The private key is p, q
- **Encryption**
  - For a message m the ciphertext is produced by $c = y^m x^r \pmod n$ where x is random.
- **Decryption**
  - Since the factorisation of n is known, the receiver can calculate $\phi(n) = (p-1)(q-1)$
  - Subsequently he raises the ciphertext to $\frac{\phi(n)}{r}$:

$$u = Enc(m)^{\frac{\phi(n)}{r}} = (y^m x^r)^{\frac{\phi(n)}{r}} = y^{m\frac{\phi(n)}{r}} x^{\phi(n)} = y^{m\frac{\phi(n)}{r}}$$

  - If $u = 1$ then the message is decrypted and $m = 0$
  - If $u \neq 1$ then the receiver can iterate over all the possible $t \in \{0, \cdots, r-1\}$ checking if

$$uEnc(t) = Enc(m)Enc(t) = Enc(m+t) = Enc(0) = 1$$

. When such a t is found, the message is decrypted $m = -t \pmod n$.
  - This brute force approach is not efficient for large r. It can be improved using the baby step - giant step algorithm ([140],[135]) and achieve the result in $O(\sqrt{r})$ steps.

Paillier cryptosystem Now we will turn our attention to the Paillier cryptosystem [107] and its generalisation in ([47], [46]), which is usually referred to as the Damgård-Jurik cryptosystem. Both cryptosystems are ideally suited for a voting context.

Firstly we shall describe the Paillier cryptosystem that consists of the standard three phases:

- **Key Generation**
  - Choose two large primes p, q randomly and independently such that $gcd(pq, (p-1)(q-1)) = 1$. It can be proved that this property is easily attained, if p, q are of the same length η
  - Calculate RSA modulus $n = pq$
  - Calculate $\boldsymbol{\lambda} = lcm(p-1, q-1) = \frac{(p-1)(q-1)}{gcd(p-1,q-1)}$ (Carmichael's Function). This function is easy to calculate if we know p, q and has the following very important properties:
    - $\forall x \in \mathbb{Z}^*_{n^2} : x^{\lambda(n)} = 1 \pmod n$ because of Carmichael's Theorem
    - $\forall x \in \mathbb{Z}^*_{n^2} : x^{n\lambda(n)} = 1 \pmod{n^2}$ because $n\lambda(n) = \lambda(n^2)$
  - Select generator $g \in \mathbb{Z}^*_{n^2}$ such that the order of g is a non zero multiple of n.
  - Calculate inverse $\boldsymbol{\mu} = L(g^{\lambda} \pmod{n^2})^{-1} \pmod n$ where $L(x) = \frac{x-1}{n}$. This function is very important in the Paillier cryptosystem. Its role can be summarized as:
    - $L()$ is given elements that are equal to $1 \pmod n$
    - $L()$ *'solves'* the discrete log problem and *'decrypts'*
    - The inverse to be calculated, always exists if g is a valid generator
  - Release the keys. The public key is (n, g) and the private Key (λ, μ)
- **Encryption:** $\mathbb{Z}_n \times \mathbb{Z}^*_n \to \mathbb{Z}^*_{n^2}$
  - Encode message m into $\mathbb{Z}_n$
  - Select random $r \in \mathbb{Z}^*_n$
  - Return $c = Enc(m, r) = g^m r^n \pmod{n^2}$
- **Decryption:** $\mathbb{Z}^*_{n^2} \to \mathbb{Z}_n$
  - Ciphertext $c \in \mathbb{Z}^*_{n^2}$
  - Return $m = L(c^{\lambda} \bmod n^2)\mu \,(mod\, n) = \frac{L(c^{\lambda} \bmod n^2)}{L(g^{\lambda} \bmod n^2)}(modn)$

The security of the Paillier cryptosystem depends on the composite residuosity problem which can be stated as:

Definition 2.1. Given $n = pq$ and $z \in \mathbb{Z}^*_{n^2}$ decide if z is $n - residue$ module n^2, does there exist $y \in \mathbb{Z}^*_{n^2}$ st: $z = y^n \pmod{n^2}$.

It is assumed that there is no polynomial time algorithm to decide the composite residuosity problem. To state it differently, it is assumed that the *Decisional Composite Residuosity Assumption-DCRA* holds, which states that In order to get a hint about the relation of the DCRA with the Paillier cryptosystem, we must observe that if there was an algorithm to decide if $z \in \mathbb{Z}^*_{n^2}$ is the encryption of message 0 then we could solve the composite residuosity problem.

An Exercise:Proving Correctness In order to familiarize ourselves with the Paillier cryptosystem we can check that it operates *correctly* ([132], [104])

We must prove the following theorem:

Theorem 2.2. *For* $c = Enc(m, r) = g^m r^n \pmod{n^2}$ *the decryption operation* $\frac{L(c^\lambda \pmod{n^2})}{L(g^\lambda \pmod{n^2})} \pmod{n}$ *yields* m.

Proof. It is proved in [47], [46] that $g = n + 1$ can allways be used. As a result we shall use this special case. To simplify the notation we shall employ $[c]_{n+1}$ to mean the plaintext that corresponds to ciphertext c using $g = n + 1$. That means $w = Enc_{n+1}([w]_{n+1}, r)$. The main step will be the proof of the lemma:

Lemma 2.3. $\forall\, w\ \in \mathbb{Z}^*_{n^2} : \boldsymbol{L(w^\lambda \pmod{n^2}) = \lambda[w]_{n+1} \pmod{n}}$

Proof. At first we prove that:

Lemma 2.4. $\forall x \in \mathbb{Z}_n : (1 + n)^x = 1 + nx \pmod{n^2}$

Proof. Most terms of the binomial expansion are zero in $\mathbb{Z}_{n^2}$:

$$\begin{aligned} (1+n)^x \pmod{n^2} &= \\ 1 + \binom{x}{1} n + \binom{x}{2} n^2 + \cdots + \binom{x}{x} n^x \pmod{n^2} &= \\ 1 + xn \pmod{n^2} \end{aligned}$$

□

Then we prove that:

Lemma 2.5. $\forall c \in \mathbb{Z}^*_{n^2}$*, and proper generators* g_1, g_2 *:* $[c]_{g_1} = [c]_{g_2}[g_2]_{g_1}$

Proof. Let $y = [g_2]_{g_1}$. This means that: $g_2 = g_1^y b^n$ where $b \in_R \mathbb{Z}^*_n$

Let $z = [c]_{g_2}$. This means that $c = g_2^z d^n$ where $d \in_R \mathbb{Z}^*_n$.

Now $c = g_2^z d^n = (g_1^y b^n)^z d^n = g_1^{zy}(b^z d)^n$ which means that $yz = [c]_{g_1}$

Rewriting gives us $[c]_{g_1} = [c]_{g_2}[g_2]_{g_1}$ □

Moving on to the main lemma: $\forall\, w\ \in \mathbb{Z}^*_{n^2} : \boldsymbol{L(w^\lambda \bmod n^2) = \lambda[w]_{n+1} \bmod n}$

Since $n + 1$ is a proper g, $\forall\, w\ \in \mathbb{Z}^*_{n^2}$: by definition:

$$w = Enc_{n+1}([w]_{n+1}, r) = (n + 1)^{[w]_{n+1}} r^n \pmod{()}n^2)$$

Now decryption computes:

$$w^\lambda = (n + 1)^{\lambda[w]_{n+1}} r^{n\lambda} \pmod{n^2}$$

Because of the binomial terms disappearing $\pmod{n^2}$ we get:

$$w^\lambda = (1 + \lambda[w]_{n+1} n) r^{n\lambda} \pmod{n^2}$$

Since $n\lambda(n) = \lambda(n^2)$:

$$w^\lambda = (1 + \lambda[w]_{n+1} n) r^{\lambda(n^2)} \pmod{n^2}$$

which leaves:

$$w^{\lambda} = (1 + \lambda[w]_{n+1}n)$$

Then: $L(w^{\lambda}) = \frac{w^{\lambda}-1}{n} = \lambda[w]_{n+1}$

□

So during the decryption operation:

$$\begin{aligned} \frac{L(c^{\lambda} \, mod \, n^2)}{L(g^{\lambda} \, mod \, n^2)} &= \\ \frac{\lambda[c]_{n+1}}{\lambda[g]_{n+1}} &= \\ \frac{[c]_{n+1}}{[g]_{n+1}} = \frac{[c]_g[g]_{n+1}}{[g]_{n+1}} = [c]_g &= m \end{aligned}$$

□

The Damgård-Jurik cryptosystem In [47],[46], both a generalisation and a simplification is given. Firstly for each $s \geq 1$ a cryptosystem $\mathcal{CS}_s$ is defined:

Generalisation

- **Key Generation:**
  - Select admissible $n = pq$ with length η bits
  - Choose random j with $gcd(j, n) = 1$ and random $x \in \mathbb{Z}_{n^s}$ and calculate $g = (1 + n)^j x \quad mod n^{s+1}$
  - Calculate $\lambda = lcm(p-1, q-1)$
  - Select d such that
    * $d = 1 \, mod \, n \in \mathbb{Z}^*_{n^s}$
    * $d = 0 \quad (mod \lambda)$
  - **Output:** Public key is (n, g) and private key is d

- **Encryption** $\mathbb{Z}_{n^s} \times \mathbb{Z}^*_n \rightarrow \mathbb{Z}^*_{n^{s+1}}$
  - Encode message m into $\mathbb{Z}_{n^s}$
  - Select random $r \in \mathbb{Z}^*_n$
  - Return $c = Enc(m, r) = g^m r^{n^s} \, (mod \, n^{s+1})$
- **Decryption**
  - Ciphertext $c \in \mathbb{Z}^*_{n^{s+1}}$
  - Calculate $c^d \quad modn^s = (1+n)^{mjd} \quad modn^s$
  - Extract mjd. There is a specific algorithm for this task, which will be presented later (figure 2.1)
  - Calculate $g^d \quad modn^s = (1+n)^{jd} \quad modn^s$
  - Extract jd

  - $m = \frac{mjd}{jd}$

It is obvious that Paillier cryptosystem is the special case $s = 1$. The security properties follow from the security properties of the original cryptosystem.

Theorem 2.6. *$\forall s$ $\mathcal{CS}_s$ is one way if Paillier ($\mathcal{CS}_1$) is one way and semantically secure iff the DCRA is true*

Simplification For the simplification, it must be noted that $g = (1 + n)$ is a valid generator. So we can define the following cryptosystem:

- **Key Generation**
  - Public key is $n = pq$
  - Private key is $\lambda = lcm(p - 1, q - 1)$
- s can be selected at any point in time before encryption, as long as $m < n^s$
- **Encryption** $\mathbb{Z}_{n^s} \times \mathbb{Z}_n^* \to \mathbb{Z}_{n^{s+1}}^*$
  - Choose s that the message m can be encoded into $\mathbb{Z}_{n^s}$
  - Select random $r \in \mathbb{Z}_n^*$
  - Return $c = Enc(m, r) = (1 + n)^m r^{n^s} \quad (mod\, n^{s+1})$
- **Decryption** $\mathbb{Z}_{n^{s+1}}^* \to \mathbb{Z}_{n^s}$
  - Ciphertext $c \in \mathbb{Z}_{n^{s+1}}^*$. Discover s by the length of c
  - Calculate $c^\lambda \quad (mod\, n^{s+1}) = (1+n)^{m\lambda \quad (mod\, n^s)} r^{\lambda n^s} \quad (mod\, n^{s+1}) = (1+n)^{m\lambda} \quad (mod\, n^{s+1})$ because $r^{\lambda n^s} = r^{\lambda n^{s+1} = 1 \quad (mod\, n^{s+1})}$
  - Extract $m\lambda$ from $(1+n)^{m\lambda} \quad (mod\, n^{s+1})$. In order to do this we shall use the $L()$ function $(mod\, n^s)$, taking into account that some terms will be reduced to $0 \quad (mod\, n^s)$

$$\begin{aligned} L((1+n)^{m\lambda} \quad (mod\, n^{s+1})) &= \\ \frac{1 - (1+n)^{m\lambda} \quad (mod\, n^{s+1})}{n} \quad (mod\, n^s) &= \\ m\lambda + \binom{m\lambda}{2} n + \cdots + \binom{m\lambda}{s} n^{s-1} \quad (mod\, n^s) \end{aligned}$$

However, we cannot extract $m\lambda$ as easily as in the case of n^2. Consequently we shall extract it step-by-step, meaning that we shall first extract the value of $m\lambda \quad (mod\, n)$, then $(mod\, n^2)$, etc. This process can be compared to accumulating the digits of a number from the least significant to the most significant (first the final one, then the final two and so on). For convienience, we shall refer to the value being extracted as i. Note that this process means that if we have computed i_{j-1} then $i_j = i_{j-1} + kn^{j-1}$ where $k \in \{0, \cdots n - 1\}$, that is we get the next number by adding a digit to the left. Consequently:

$$\begin{aligned} L((1+n)^i \quad (mod\, n^{j+1})) &= \\ i_j + \binom{i_j}{2} n + \cdots + \binom{i_j}{j} n^{j-1} \quad (mod\, n^j) &= \\ i_{j-1} + kn^{j-1} + \binom{i_{j-1}}{2} n + \cdots + \binom{i_{j-1}}{j} \quad (mod\, n^j) \end{aligned}$$

The crucial step in the above calculation is the replacement: $\binom{i_j}{t}n^{t-1} = \binom{i_{j-1}}{t}n^{t-1}$ This is easy to show since:

$$\begin{aligned}
\binom{i_j}{t}n^{t-1} &= \\
n^{t-1}\prod_{x=1}^{t}\frac{i_j-(t-x)}{x} &= \\
\frac{n^t}{n}\prod_{x=1}^{t}\frac{i_j-(t-x)}{x} &= \\
\frac{1}{n}\prod_{x=1}^{t}\frac{n(i_j-(t-x))}{x} &= \\
\frac{1}{n}\prod_{x=1}^{t}\frac{n(i_{j-1}+kn^{j-1}-(t-x))}{x} &= \\
\frac{1}{n}\prod_{x=1}^{t}\frac{(ni_{j-1}+kn^{j}-n(t-x))}{x} &= \quad (mod\, n^j) \\
\frac{1}{n}\prod_{x=1}^{t}\frac{(ni_{j-1}-n(t-x))}{x} &= \\
\frac{n^t}{n}\prod_{x=1}^{t}\frac{(i_{j-1}-(t-x))}{x} &= \\
n^{t-1}\binom{i_{j-1}}{t}
\end{aligned}$$

As a result:
$L((1+n)^i \pmod{n^{j+1}}) = kn^{j-1} + i_{j-1} + \binom{i_{j-1}}{2}n + \cdots + \binom{i_{j-1}}{j} \pmod{n^j}$
which means that:
$kn^{j-1} = L((1+n)^i \pmod{n^{j+1}}) - (\binom{i_{j-1}}{2}n + \cdots + \binom{i_{j-1}}{j}) \pmod{n^j}$
If we replace kn^{j-1} in $i_j = i_{j-1} + kn^{j-1}$ we get:

$$i_j = L((1+n)^i \pmod{n^{j+1}}) - (\binom{i_{j-1}}{2}n + \cdots + \binom{i_{j-1}}{j}n^{j-1}) \pmod{n^j}$$

which provides a computation algorithm (figure 2.1):

```
i=0
for j in range(1,s+1):
    nj = n**j
    t1=(x%(nj*n)-1)//n
    t2=i
    sum=0
    for k in range(2,j+1):
        sum += binomial(t2,k)*n**(k-1)%nj
    t1=(t1-sum)%nj
    i=t1
```

FIGURE 2.1: Message Extraction Algorithm during Damgård-Jurik Decryption

– After extraction we can retrieve m from $m\lambda$ by multiplying with $\lambda{-}1 \pmod{n^{s+1}}$

Once again, the security properties follow from the security properties of the original cryptosystem.

Theorem 2.7. *$\forall s$ the simplified version is one way if Paillier ($\mathcal{CS}_1$) is one way and semantically secure iff the DCRA is true.*

2.1.2 Homomorphic Cryptosystems

Homomorphic encryption enables computations on encrypted data. It allows to apply a function to the cipher texts that corresponds to another function on the plaintexts. The result can then be obtained by a single decryption. This can be illustrated with the following relation:

$$Enc(m_1) \otimes Enc(m_2) = Enc(m_1 \oplus m_2)$$

In the voting context homomorphic encryption allows us to compute the tally while maintaining secrecy. For simple tallying we would require to evaluate a function on cipher texts that corresponds to adding the plain texts (additive homomorphism). For other ways to compute the tally we would require computation of arbitrary functions on encrypted data.
It is easy to see that:

- The RSA and the El Gamal cryptosystem are multiplicatively homomorphic.
- The exponential El Gamal cryptosystem is additively homomorphic.
- The Goldwasser Micali and the Benaloh Cryptosystems are additively homomorphic.
- The Paillier cryptosystem and the related generalisations and simplification are additively homomorphic.

In this thesis we shall extensively use homomorphic cryptosystems. In what follows, we describe some useful homomorphic properties.

The multiplicative homomorphishm of the El Gamal cryptosystem yields a nice property called *reencryption*, which will be used extensively in Chapter 4. Specifically:

$$\begin{aligned} Enc(m_1, r_1) \cdot Enc(m_2, r_2) &= (g^{r_1}, m_1 y^{r_1}) \cdot (g^{r_2}, m_2 y^{r_2}) \\ &= (g^{r_1} g^{r_2}, m_1 m_2 y^{r_1 + r_2}) = Enc(m_1 m_2, r_1 + r_2) \end{aligned}$$

For $m_2 = 1$ the above relationship implies that:

$$Enc(m1, r1) \cdot Enc(1, r2) = Enc(m_1, r_1 + r_2) = ReEnc(m1)$$

which means that we changed the cipher text corresponding to the original message, simply by adding randomness and without knowing the decryption key. The same can happen in an dditively homomorphic cryptosystem, if we combine with $m_2 = 0$.

Some other interesting homomorphic properties concern the Damgård-Jurik cryptosystem based on the fact that a ciphertext c from $\mathcal{CS}_s$ belongs to $\mathbb{Z}^*_{s+1}$ which places it in the plaintext space of $\mathcal{CS}_{s+t}$, for $t \geq 1$ namely $\mathbb{Z}_{s+t}, t \geq 1$ [8]. More concretely: $\mathbb{Z}^*_{s+1} \subseteq \mathbb{Z}_{s+t}, t \geq 1$. This has the following properties as a result (where Enc_s means encryption in $\mathcal{CS}_s$ and Enc_t means encryption in $\mathcal{CS}_{s+t}$):

- $Enc_t(1)^{Enc_s(m)} = Enc_t(m)$
- $Enc_t(0)^{Enc_s(m)} = Enc_t(0)$
- $Enc_t(0) Enc_t(Enc_s(m)) = Enc_t(Enc_s(m))$

- $Enc_t(Enc_s(0, r_0))^{Enc_s(m, r_m)} = Enc_t(Enc_s(0, r_0)Enc_s(m, r_m)) = Enc_t(Enc_s(m, r_0 r_m))$

These properties enable to transfer from $\mathcal{CS}_s$ to $\mathcal{CS}_{s+t}$ without changing keys (since s can be selected after the key generation phase).

2.1.3 Secure Cryptosystems

The security of a cryptosystem is a very important factor in its use. In order to formalise the exact meaning of the concept of security, a game paradigm is followed, resulting in the following notions and corresponding definitions ([81], [21]):

Semantic Security This is the most basic notion characterising a cryptosystem. Given a ciphertext the (computationally bounded) adversary $\mathcal{A}$ should not learn anything regarding the plaintext. This is formally defined using a the concept of *indistinguisability*. $\mathcal{A}$ selects two challenge messages and obtains an encryption of one at random. $\mathcal{A}$ should not be able to guess to which message the encryption corresponds with probability better than pure chance ($\frac{1}{2}$). Note that in this definition the adversary is entirely passive.

Chosen Plaintext Security (IND-CPA) A more dangerous $\mathcal{A}$ can freely and adaptively encrypt messages of his choice before and after the encryption of the challenges. For a cryptosystem to be secure in this fashion the attacker should still be unable to distinguish between an encryption of two possible plaintext messages of his choice with probability better than a lucky guess.

Chosen Ciphertext Security (IND-CCA) This time the adversary can tamper with the ciphertexts produced. More specifically the adversary can use the system as a decryption oracle, by submitting ciphertexts of his choice and retrieving their decryption. At some point $\mathcal{A}$ will submit two possible plaintext messages of his choice. The goal remains the same: $\mathcal{A}$ should not be able to distinguish which of the plaintext messages got encrypted.

Adaptive Chosen Ciphertext Security (IND-CCA2) The adversary in this game acts as previously. However he might continue using the decryption oracle, after submitting his plaintexts of choice.

IND-CCA and IND-CCA2 are related to homomorphic cryptosystems. For instance if a cryptosystem supports reencryption it cannot be CCA2 secure, since after submitting the challenges, $\mathcal{A}$ will reencrypt the response, and forward it to a decryption oracle, which will cause him to win the game with certainty.

2.1.4 Commitment Schemes

Commitment Schemes are tools that enable a party, to commit to a value, without revealing it (*hiding* property) and without being able to change it (*binding* property). The usefulness of commitment schemes becomes immediately evident with the following classical example of coin flipping over the telephone, due to Manuel Blum:

Alice (the *sender*) and Bob (the *receiver*) are in different locations but want to flip a coin. They agree that Alice selects head/tails and Bob flips and reports the result. A malicious Bob doesn't have to flip the coin, he can just announce that he wins and Alice can do nothing about it. On the other hand, a malicious Alice can require that she doesn't announce her option first, so that Bob cannot make an informed selection. Consequently Bob flips the coin and Alice select the option that results in her winning.

A commitment can solve this problem, due to the hiding and binding properties.

- Alice commits to heads or tails and sends the commitment to Bob. Due to the hiding property Bob cannot learn anything from the token he received.
- Bob flips the coin and announces the result
- Alice reveals her choice. Due to the binding property Alice cannot change her commitment after the fact in order to win.
- They both check the result and accept the winner.

The interaction above, implies two phases for a commitment scheme: The *commitment* phase and the *opening* phase.

In general commitment implementation, referred to as blobs in [24], must satisfy the following properties:

- The sender can commit to a bit - the commitment phase is realisable.
- The sender can open any commitment made by her and convince the receiver. A commitment must open in a unique so the receiver can be convinced that this was the initial value selected by the sender. A bit commitment cannot open both as a 0 and as 1. This is the binding property.
- Commitments reveal nothing about their values. This is the hiding property.
- Commitments must not *leak* any information.

Chameleon Blobs A blob is said to be *chameleon*, if the receiver can do what the second property above prohibits the sender, ie. to open a bit commitment both as 0 and a 1. This twist is very important in some voting schemes.

2.1.4.1 Hash Functions

Hash Functions is a generic cryptographic tool that can also be used for commitments is a function h that maps arbitrary size data (message) to fixed size data (hash) with the following properties [121]:

1. Given the message it is easy to calculate the hash
2. Given the hash it is computationally infeasible to find the message.
3. Given a message m it is computationally infeasible to find another message m' such that $h(m) = h(m')$
4. It is computationally infeasible to find two messages m_1, m_2 such that $h(m_1) = h(m_2)$

The second item defined above can constitute the hiding property of commitments, while the latter items can be used to implement the binding property. Of course, the computational parameter must be noted, as a computationally unbounded adversary can defeat the scheme.

2.1.4.2 Pedersen Commitment

A commitment scheme that is computationally binding and unconditionally hiding was proposed by Pedersen and described in [81]. It is based on the DLOG Problem and works as follows:

- The receiver selects the two safe primes p, q and a generator g from a q order subgroup of $\mathbb{Z}_q^*$.

- In addition he generates a random value $r \in \mathbb{Z}_q$ and calculates $h = g^r$
- The sender receives (p, q, g, h).
- To commit to a value $M \in \mathbb{Z}_q$ she selects $R \in \mathbb{Z}_q$ and calculates $C = g^R h^M$

The verification of the commitment works as follows:

- The sender sends M, R to Bob
- The receiver verifies if $g^R h^M = C$ and accepts if this is the case.

Theorem 2.8. *The Pedersen Commitment scheme satisfies the binding and hiding properties if the DLOG Problem is hard.*

Proof. At first we shall prove that the scheme satisfies the binding property if the DLOG Problem is hard. We shall prove the contrapositive. If it didn't the Sender would be able to change her message to M' without breaking the commitment. This means that:

$$C = g^R h^M$$
$$C = g^{R'} h^{M'}$$

As a result:

$$\begin{aligned} g^R h^M = g^{R'} h^{M'} \Rightarrow \\ g^{R-R'} = hM - M' \Rightarrow \\ g^{(R-R')k} = h \quad \text{where:} \quad k(M - M') = 1 \quad (mod\, q) \end{aligned}$$

This contradicts the DLOG problem as k can be found easily using the Euclidean Algorithm.

For the hiding property, we shall prove that it is unconditional which means that any value $M^* \in \mathbb{Z}_q$ has an equal probability of being the committed one. If the Receiver has unbounded computational abilities than given C, he can compute M, R. However:

$$\begin{aligned} \forall M^* \in \mathbb{Z}_q \quad \exists R^* \quad \text{such that :} \\ C = g^{R^*} h^{M^*} \text{as} \\ C = g^{R+Mr} = g^{R^*+M^*r} \Rightarrow \\ R + Mr = R^* + M^*r \Rightarrow \\ R^* = (M - M^*)r + R \quad (mod\, q) \end{aligned}$$

□

The perfect hiding property will be used extensively in some election schemes described in the following chapters.

2.1.4.3 Designated Verifier Proofs [75]

The chameleon twist in commitment schemes, has many applications in electronic voting. An example was given in [75]. Instead of proofs with public verifiability, voting also requires proofs that can be verified only by a specific voter (the vote caster) in order to evade an aspiring coercer (more on this on chapter 5). Such proofs are called *designated verifier proofs* (short: DV proof) since only the verifier designated by the prover can validate the proof. If under coercion or in any other case the verifier wants to transfer the proof to an attacker, the proof will be rendered useless (under some assumptions of course).

This is implemented using a simple idea. If the prover wants to prove statement Φ to the verifier, she constructs a designated verifier proof by proving the disjunction $\Phi \bigvee DV$ where DV corresponsds to the statement *I am the designated verifier*. In practice, DV corresponds to knowledge or possession of some secret trapdoor information on a chameleon commitment scheme. Only the designated verifier has access to the trapdoor information (a secret key). On this premise, he is convinced of Φ, since the second argument to the disjunction evaluates to false. However, when he presents the proof to a third party, it obtains no extra knowledge since the designated verifier indeed possesses the secret key and the disjunction evaluates to true regardless of the validity of Φ. For the scheme to work, the prover must use a scheme that can be decommitted in many different indiscernible ways. As a case in point, one can use the Pedersen commitment scheme 2.1.4.2 with the public key of the verifier. More specifically (parameters are assumed to come from the appropriate groups with hard DLOG):

- Let $x_V, y_V = g^{x_v}$ be a secret public key pair.
- The prover commits to w by computing $g^w y_V^r$
- To open the commitment the prover sends (w, r)
- The verifier can validate the commitment since he knows x_V
- The verifier can present to an adversary (w', r') such that the commitment is valid

2.1.5 Secret Sharing

In a public key cryptosystem the holder of the secret key has the power to decrypt. This power is critical in voting implemented by a homomorphic cryptosystem, since the holder will be the first to know the election outcome. In that case too much trust is placed on a single entity, allowing for a single point of failure. The secret key should be distributed to as many entities as possible. The entity that distributes the secret is called the *dealer*, while the entities that receive the share are called the *players*.

2.1.5.1 Shamir Secret Sharing

The first method to enable such secret sharing was proposed in [125]. The objective of the scheme is to share a secret element s between l players so that any t subset can recover it, but no $t-1$ element subset can. To achieve this, it utilises as a main idea that of *polynomial interpolation*, using Lagrange coefficients.

- A polynomial P of degree $t-1$ can be reconstructed from t distinct elements $\{(x_i, y_i)\}_{i=1}^t$
- $P(x) = \sum_{i=1}^t y_i \prod_{j=1, j\neq i}^t \frac{x-x_j}{x_i-x_j}$

As a result in order to share a secret:

- Dealer chooses a random polynomial of degree $t-1$ so that $P(0) = s$
- Distributes l pairs $(x_i, P(x_i)), \quad x_i \neq 0$
- t players can reconstruct the polynomial (and recover s), but $t-1$ players cannot

In order to share an integer secret, the scheme can be applied to $\mathbb{Z}_p$ where p is a prime larger than the secret.The polynomial coefficients are selected randomly from $\mathbb{Z}_p$ and all arithmetic operations are done $\pmod p$.

The scheme above assumes that all the players are honest. However this might not always be the case [122]:

- The dealer might give out incorrect shares to all or part of the players. As a result the secret will not be reconstructed. To deal with this threat the players must be able to validate them.
- The player might not present their correct shares during reconstruction. To deal with this threat the shares need to be checkable by everybody.

As a result the basic protocol by [125] needs to be augmented to account for these threats.

2.1.5.2 Verifiable Secret Sharing [53]

The dealer includes additional information for correctness verification of the shares. More specifically, commitments to the coefficients are released. If $P(x) = \sum_{i=0}^{t} a_i x^i$, then the dealer releases the commitments $\{c_i = g^{a_i}\}_{i=0}^{t}$, where g is a generator of a group where the DLOG problem is hard.

In order to verify that a share (x_i, y_i) is valid, that is $y_i = P(x_i)$ indeed holds, the interested party checks if:

$$g^{y_i} = \prod_{j=0}^{t} c_j^{x_i^j}$$

which should be valid since:

$$\prod_{j=0}^{t} c_j^{x_i^j} = \prod_{j=0}^{t} g^{a_j x_i^j} = g^{\sum_{j=0}^{t} a_j x_i^j} = g^{P(x_i)}$$

2.1.5.3 Public Verifiable Secret Sharing

A player in a secret sharing scheme might not present their correct shares during the reconstruction phase. To solve this problem, the verification is not restricted to each player's own shares, but the protocol information needs to be verifiable by anyone, ie. publicly verifiable. A representive of such a protocol is presented in [122] and consists of the distribution and reconstruction phases:

- **Distribution**
  - Suppose that the polynomial to aid in sharing is: $P(x) = \sum_{i=0}^{t} a_i x^i$
  - Each participant has a secret key sk_i and a public key $pk_i = G^{sk_i}$ where G is a generator of a group where the DLOG problem is hard. Along with G, another generator g will be used.
  - The dealer commits to the coefficients $\{c_i = g^{a_i}\}_{i=0}^{t}$
  - The dealer encrypts the shares with the public key of the corresponding participant: $\{Y_i = pk_i^{p(i)}\}_{i=1}^{n}$
  - The dealer proves using the Chaum-Pedersen protocol of 2.1.7.2 that the shares are consistent by proving that $\{log_g \prod_{j=0}^{t} c_j^{x_i^j} = log_{pk_i} Y_i = p(i)\}_{i=1}^{n}$.
  - The shares are verified by checking if:
- **Reconstruction**

- Each participant decrypts the shares by calculating: $Y_i^{\frac{1}{sk_i}} = pk_i^{p(i)\frac{1}{sk_i}} = G^{sk_i p(i)\frac{1}{sk_i}} = G^{p(i)}$
- The shares are reconstructed using the Lagrange interpolation technique in the exponents.

2.1.5.4 Secret Sharing Homomorphisms

An important property of some secret sharing schemes was observed by Benaloh in [20]. Suppose a *super secret* S can be computed from m *sub secrets* $\{S_i\}_{i=1}^{m}$ using a function f $S = f(S_1, ..., S_m)$. Depending on the secret sharing scheme, the super secret can be computed while keeping the sub secrets private, thus realising secure function evaluation. This can be done without any assumptions using the following procedure:

- Each of the sub secret holder acts as the dealer in the secret sharing scheme and deals the shares into n entities $\{E_j\}_{j=1}^{n}$ each receiving $\{S_{ij}\}_{i=1,j=1}^{m,n}$. The receiving entities are the same for each holder.
- Each entity combines its shares using a function g and produces: $R_j = g(S_{1j}, ..., S_{mj})$
- A subset of the entities reconstruct S using the secret sharing scheme.

$$S \leftarrow_f \begin{bmatrix} S_1 \\ S_2 \\ \cdots \\ S_m \end{bmatrix} \rightarrow_{share} \begin{bmatrix} E_1 & \cdots & E_n \end{bmatrix} \rightarrow \begin{bmatrix} S_{11} & \cdots & S_{1n} \\ S_{21} & \cdots & S_{2n} \\ & \cdots & \\ S_{m1} & \cdots & S_{mn} \end{bmatrix} \rightarrow \begin{bmatrix} g(S_{11}, \cdots, S_{1n}) \\ g(S_{21}, \cdots, S_{2n}) \\ \cdots \\ g(S_{m1}, \cdots, S_{mn}) \end{bmatrix} \rightarrow S$$

FIGURE 2.2: Secret Sharing Homomorphisms

Such a secret sharing scheme is called $(f, g) - homomorphic$. To be feasible **each g combination of the shares of a secret must be a share of the f combination of the secrets**. For example the secret sharing scheme of Shamir 2.1.5 is $(+, +) - homomorphic$ since the sum of the polynomials use to share the sub secrets is itself a polynomial that can be used to deal shares of the super secret. Another property that must be satisfied is the preservation of the reconstruction threshold. For example, if the sub secrets can be reconstructed using t participants the same must hold for the super secret as well.

2.1.6 Threshold Cryptosystems

If the schemes of the previous sections are applied to the secret sharing of the private key in a public key cryptosystem, resulting in the distribution of the decryption function we get *threshold cryptosystems*. These cryptosystems generate a public key and shares of the secret key. The encryption function proceeds as usual. For the decryption however, all authorities apply their shares to the ciphertext creating intermediate values, that are then combined to retrieve the plaintext. More specifically, a threshold cryptosystem consists of 4 algorithms, which are described in the figures 2.1.6.

2.1.6.1 Threshold El Gamal

The El Gamal cryptosystem can be combined with the secret sharing technique of 2.1.5 in order to share the private key to l shareholders (voting authorities in our case) so that each one possesses a share x_i of it and that decryption proceeds without anybody learning the key. Robustness and privacy is provided by the fact that at least t authorities must cooperate to decrypt, but they cannot recreate the key.

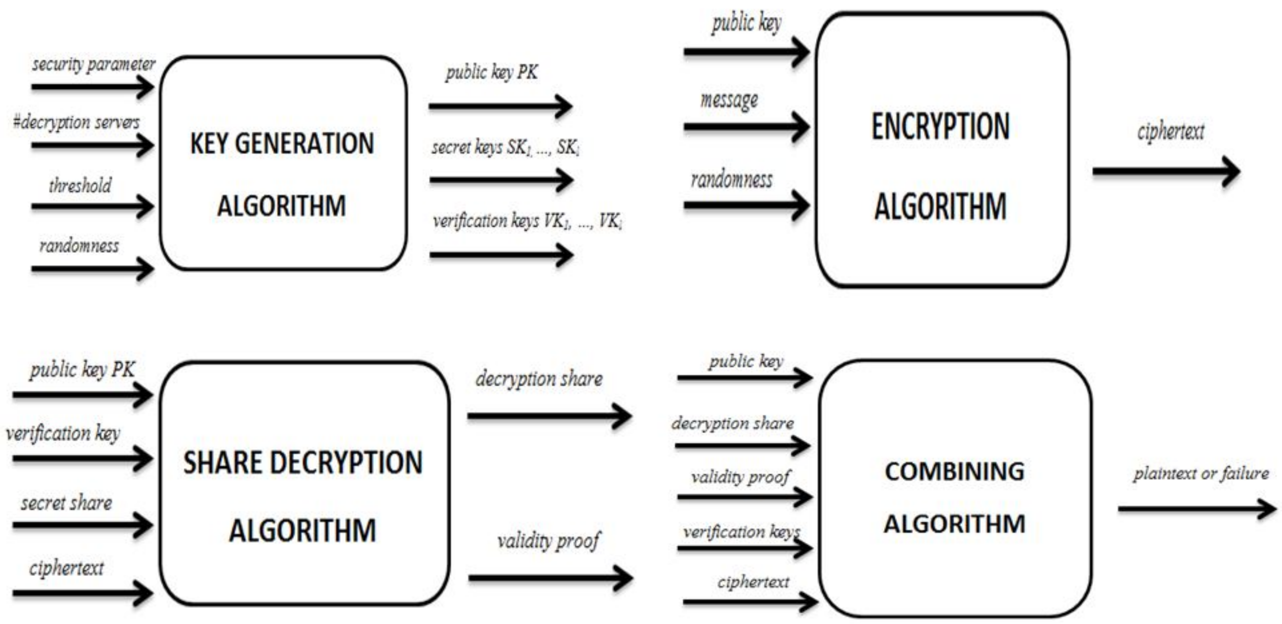


FIGURE 2.3: Threshold Cryptosystem Components

- **Key Generation** A *trusted* dealer splits the key $x \in_R \mathbb{Z}_q$ in l shares $\{x_i = P(i)\}_{i=1}^{l}$ using Shamir's scheme and distributes them to the authorities. The public key can be computed as always as $y = g^x$
- **Encryption** Encryption proceeds as in regular El Gamal yielding the ciphertext $c = (G, M)$
- **Share Decryption** The authorities extract the first part of the ciphertext and calculate $c_i = G^{x_i} \quad (mod\, p)$ and publish their results
- **Combination** The combinator gathers t decryption shares and multiplies them yielding $\hat{c} = \prod_{i=1}^{t} c_i^{\lambda_i} \quad (mod\, p)$ where $\lambda_i = j(j-i)^{-1} \quad (mod\, q)$ are the Lagrange coefficients. As a result $\hat{c} = G^{P(0)} = G^x$
- **Decryption** The ciphertext is decrypted as usual by computing $\frac{M}{\hat{c}}$

Alternatively the construction in [109] can be used to eliminate the need for a trusted dealer. More specifically:

- **Key Generation**
  - Each authority i chooses $x_i \in_R \mathbb{Z}_q$ to be his share of the key.
  - Moreover, he chooses polynomial $f_i(z) = \sum_{j=0}^{t-1} f_{ij} z^j$ where $f_{i0} = x_i$ and $f_{ij} \in_R \mathbb{Z}_q$.
  - For each coefficient he broadcasts $F_{ij} = g^{f_{ij}}$.
  - Each participant i sends $s_i j = f_i(j)$ to participant j.
  - Each participant verifies the shares he received against the broadcasted ones, by checking if $g^{s_{ij}} = \prod_{l=0}^{t-1} F_{jl}^{il}$
  - Each authority i commits to the shares by announcing $y_i = g^{x_i}$
- **Encryption** Encryption can proceed as in regular El Gamal.

- **Share Decryption** and **Combination** To decrypt the ciphertext (G, M) each auhority calculates $w_i = G^{x_i}$. The plaintext can be uncovered as $\frac{M}{\prod_{i\in\Lambda} w_j^{\lambda_i}}$. A technical detail remains, namely that $x_i = log_G w_i = log_g y_i$ which can be done using the Chaum-Pedersen protocol that is described later.

2.1.6.2 Threshold Versions of Damgård Jurik cryptosystem

Similar constructions have been proposed for the RSA cryptosystem [126]. We are not interested in the scheme per se, but it yields a nice property by allowing some secret sharers holding pieces of an exponent, to raise an item to it as a whole $\pmod n$ where $n = pq$.Based on the this scheme [56] proposed a threshold version of the Paillier cryptosystem. This approach was simplified in [46] and applied to the Damgård Jurik cryptosystem. It comes as no surprise, that security of the threshold versions depend on the security of the original cryptosystem. In Chapter 7 we implement a proof of concept implementation of a voting scheme that utilities this approach.

- **Key Generation**
  - Two safe primes $p = 2p' + 1, q = 2q' + 1$ and their products $n = pq$ and $m = p'q'$ are formed
  - The secret key d is created by using the chinese remainder theorem on the equations $d = 0 \pmod m$ and $d = 1 \pmod{n^s}$. It should be noted here that the above step means that s should be selected during the key generation phase, and not during the encryption phase. However a larger s can be selected in this phase and a smaller one can be used later.
  - The secret key is shared to l parties using 2.1.5 where all operations take place in $\mathbb{Z}_{n^s m}$. The shares will be denoted s_i
  - The public key is $(n, s, \Delta = l!)$
- **Encryption** Encryption of message m takes place as in 2.1.1.3 namely $c = (1+n)^m r^{n^s} \pmod{n^{s+1}}$
- **Share Decryption** Given the ciphertext c, each shareholder computes $c_i = c^{2\Delta s_i} \pmod{n^{s+1}}$
- **Combination**
  - The combinator selects t valid shares and computes $\lambda_i = 2\Delta \prod_{j=1, j\neq i}^{t} j(j-i)^{-1} \pmod{n^s m}$.
  - Then he calculates $\prod_{i=1}^{t} c_i^{\lambda_i}$ utilising the secret sharing scheme.
- **Decryption**
  - In effect the combinator has found $c^{4\Delta^2 d} = (1+n)^{4\Delta^2 m}$ since:
    * $d = 0 \pmod\lambda \Rightarrow 4\Delta^2 d = 0 \pmod\lambda$. Consequently: $r^{4\Delta^2 d} = 1 \pmod{n^s}$ because of Carmichael's theorem
    * $d = 1 \pmod{n^s} \Rightarrow 4\Delta^2 dm = 4\Delta^2 m \pmod{n^s}$
  - The extraction algorithm 2.1 is used to retrieve $4\Delta^2 m \pmod{n^s}$
  - The message is found by multiplying with $(4\Delta^2)^{-1} \pmod{n^s}$

If we assume dishonest players or a dishonest dealer then a validity proof should be added to the share decryption process.

2.1.7 Zero Knowledge Proofs

In *zero knowledge (ZK)* protocols, proposed in [62], 2 parties namely *the prover* and *the verifier* interact. The prover wants to convince the verifier about the knowledge of a secret (often referred to as witness), without disclosing (any part) of it. A typical use of such a protocol is for authentication. The prover is the user that would like to login to a system. In order to validate her identity she must provide a password. This means that she must send some form of the password over the network and that the system (the verifier) must compare the received value with a form suitable for storage. This interaction opens a window of opportunity to attack the password. Security degrades when the protocol is executed multiple times [128]

Zero Knowledge Protocols, are probabilistic in nature and can convince the verifier that the prover knows the password without requiring it to be transmitted or stored in any form. Attacks, of course, are not excluded. To be useful, zero knowledge proofs must have three properties:

- Completeness: Honest prover convinces honest verifier with overwhelming probability
- Soundness: A dishonest prover that does not possess the secret, cannot succeed with overwhelming probability in convincing an honest verifier. More formally if a dishonest prover can convince an honest verifier, then a polynomial algorithm can be constructed that can extract the secret from the protocol interactions.
- Zero Knowledge: The verifier cannot learn anything from the protocol and as a result cannot impersonate the prover. More formally, if the protocol does not leak information than an entity that does not possess the secret (simulator) can construct valid instances of the protocol. Based on the concept of simulated transcripts, the following variations of zero knowledge can be observed.
  - **(Perfect) Zero Knowledge**: The real and simulated transcripts are indistinguishable to a third party with unlimited computational power, which means that they indeed have the same distribution.
  - **Computational Zero Knowledge**: The real and simulated transcripts are indistinguishable to a third party with computational power limited to probabilistic polynomial algorithms.
  - **Statistical Zero Knowledge**: The real and simulated transcripts come from probability distributions with negligible statistical distance.
  - **Honest Verifier Zero Knowledge (HVZK)**: In this setting the verifier is honest, however performs additional computations except the ones necessary for protocol execution.

An illustrating example to further clarify the use of a zero knowledge protocol is described in: [100]

- **Prover** holds two *identical* boxes of different color
- **Verifier** is color blind
- **Prover** wants to convince the Verifier that the boxes have different color

The solution to this problem can be achieved with the following protocol:

1. **Prover** gives the boxes to the verifier
2. **Verifier** hides the boxes behind her back, one box per hand
3. With probability $\frac{1}{2}$ verifier switches boxes in each hand, behind her back

4. **Verifier** reveals boxes
5. **Prover** can tell whether the verifier switched hands
6. Repeat n times to decrease cheating probability to $\frac{1}{2^n}$
7. **Verifier** is convinced, that the boxes have different color, without ever knowing what it is

In voting systems Zero Knowledge Protocols can be used so that a voter might demonstrate that his vote is valid without revealing it and that an entity has performed according to a specification without revealing the result of the computations.

2.1.7.1 The Schnorr Protocol and variations

The Schnorr Protocol enables the prover to demonstrate knowledge of the discrete logarithm without revealing it and is depicted in the figure below, as described in [81] and [123]:

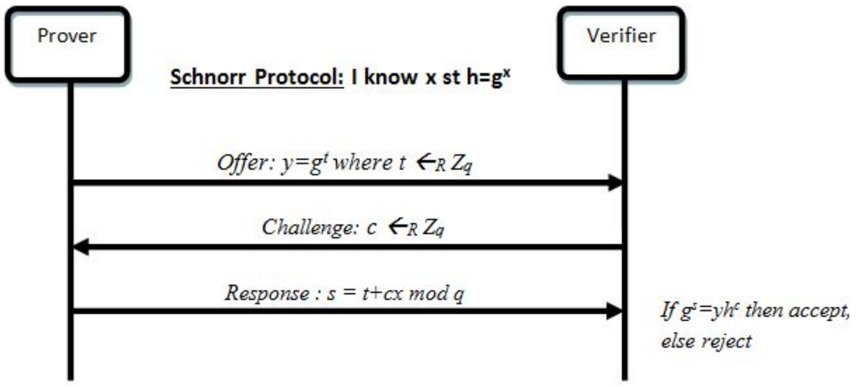

Prover
Verifier
Schnorr Protocol: I know x st h=g^x
Offer: y=g^t where t ←R Zq
Challenge: c ←R Zq
Response : s = t+cx mod q
If g^s=yh^c then accept, else reject

FIGURE 2.4: The Schnorr Protocol

The common inputs to the protocol are two safe primes p, q and a generator g of a group of order q. The prover knows x such that $h = g^x$ and wants to prove this knowledge without revealing x. The protocol consists of three phases:

- **Offer:** Randomly select $t \in_R \mathbb{Z}_q$, calculates $y = g^t$ and sends y its to the verifier
- **Challenge:** The verifier randomly selects $c \in_R \mathbb{Z}_q$ and sends c it to the prover
- **Response:** The prover calculates $s = t + cx \pmod q$ and sends s to the verifier
- The verifier accepts if $g^s = yh^c$

The protocol is complete as an honest prover can always convince an honest verifier since: $g^s = g^{t+cx} = g^t g^{cx} = yh^c$. The protocol is sound [123], as a prove that does not know x can cheat the verifier with probability $\frac{1}{q}$ which is negligible. This is because a cheating prover can only execute the following steps:

- Select randomly a challenge c'
- Randomly select $t \in_R \mathbb{Z}_q$ and offer $y = g^t h^{-c'}$
- If the verifier chooses $c = c'$ then the prover can answer by sending $s = t$ and the verifier will accept since $yh^{c'} = g^t h^{-c'} h^{c'} = g^t = g^s$. However, this is the only choice that causes the verifier to accept and the probability of this happening is exactly $\frac{1}{q}$.

However the protocol cannot be proved zero knowledge. The only case a simulator can handle is the one described above where c' is chosen in advance and $c = c'$. In order to obtain this particular value the verifier must be run (and stopped) on average q times which is exponential in the security parameter. It can however be proved honest verifier zero knowledge, since the distributions producted by an honest verifier is identical to the one produced by the simulator [123].

A similar proof is given for the Paillier cryptosystem [14], amd a simplification is described in [46], who note that if $x = Enc(m, r) = (1+n)^m r^{n^s} \quad (mod\, n^{s+1})$ then $x(1+n)^{-m} = r^{n^s} \quad (mod\, n^{s+1})$. So the only thing to be proved is that $u = x(1+n)^{-m}$ is an encryption of zero, or equivalently that the prover knows r such that u is a n^s power. This can be done using the protocol below:

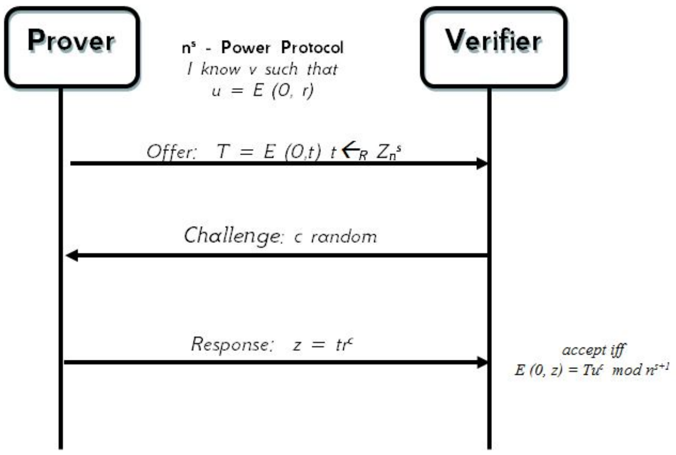


FIGURE 2.5: Proof of knowledge of randomness

The completeness of the proof is easy to see, since $Enc(0, z) = z^{n^s} = t^{n^s} r^{cn^s} = Tr^{n^{s^c}} = TEnc(0, r)^c = Tu^c$. It is also proved honest verifier zero knowledge in [46].

2.1.7.2 The Chaum-Pedersen Protocol

A variation of the Schnorr Protocol can be used to show that two discrete logarithms are equal [33]. The common inputs to the protocol are two safe primes p, q and two generators g_1, g_2 of a group of order q. The prover knows x such that $h_1 = g_1^x$ $h_2 = g_2^x$ and wants to prove this equality without revealing x. The protocol consists of three phases:

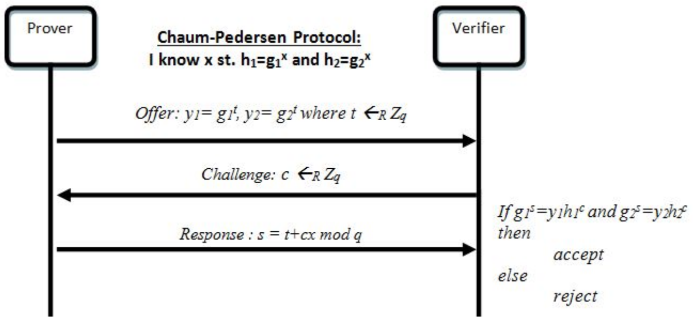

Prover
Verifier
Chaum-Pedersen Protocol:
I know x st. h_1=g_1^x and h_2=g_2^x
Offer: y_1= g_1^t, y_2= g_2^t where t ←_R Z_q
Challenge: c ←_R Z_q
Response : s = t+cx mod q
If g_1^s=y_1h_1^c and g_2^s=y_2h_2^c
then
accept
else
reject

FIGURE 2.6: The Chaum-Pedersen Protocol

- **Offer:** Randomly select $t \in_R \mathbb{Z}_q$, calculates $y_1 = g_1^t, y_2 = g_2^t$ and sends y_1, y_2 to the verifier
- **Challenge:** The verifier randomly selects $c \in_R \mathbb{Z}_q$ and sends it to the prover.

- **Response:** The prover calculates $s = t + cx \pmod q$ and sends it to the verifier
- The verifier accepts if $g_1^s = y_1 h_1^c$ and $g_2^s = y_2 h_2^c$

This protocol can be proved complete, sound and honest verifier zero-knowledge using the same techniques used in the previous protocol [123].

The Chaum-Pedersen protocol in conjuction with the Schnorr protocol can be used to prove that a tuple c_1, c_2 is an ElGamal encryption of message m. If this is the case then $(c_1, c_2) = (g^r, mh^r)$ which means that $log_g c_1 = log_h(\frac{c_2}{m})$.

2.1.7.3 Witness Hiding and Proofs of Partial Knowledge

A variation of zero knowledge proofs that has many applications in electronic voting systems was proposed in [52]. *Witness Indistinguishable (WID)* and *Witness Hiding (WH)* proofs relax the condition of no information leakage found in ZK proofs. WID proofs dictate, informally, that if the prover has a choice of multiple witnesses for the same value, the verifier cannot find out which one has been used in the transcript. This means that the protocol might leak some information, but not enough to determine the witness. In WH protocols the cheating verifier cannot fully uncover the witness from the transcript.

In [42] a general framework is given for converting a ZK proof to a WID proof. The ZK proof must be generated from a three round honest verifier protocol with special soundness. The method employed uses a simulator and a secret sharing scheme and in simple terms it proceeds as follows:

- Let $W = \{w_1, \cdots, w_n\}$ the set of alternative witnesses.
- For the actual witness used the prover calculates the offer dictated by the ZK protocol.
- For the alternate witnesses the prover calls the simulator, which returns the relevant offers that would cause the verifier to accept in a simulated transcript.
- The prover sends all the offers computed in the previous steps to the verifier.
- The verifier sends a random challenge.
- The prover interprets the challenge as a secret to be shared. The shares of the secret will be the random values employed by the simulator.
- The prover calculates the rest of the shares and the appropriate responses.
- The verifier validates the responses.

To be more concrete we shall make the Schnorr protocol witness indistinguishable [123]. The prover must prove that he knows x_1 such that $h_1 = g^{x_1}$ or (exclusively) x_2 such that $h_2 = g^{x_2}$ without revealing which one. Assume that the prover has a witness for the first case.

- **Offer**
  - For the actual witness the prover calculates $y_1 = g^{t_1}$ where $t_1 \in_R \mathbb{Z}_q$
  - For x_2 the prover calculates $y_2 = g^{t_2} h_2^{-c_2}$ where t_2 is a random value and c_2 to be interpreted as a random share of the challenge.
  - The prover sends y_1, y_2 to the verifier.
- **Challenge**

  – The verifier challenges with the secret to be shared $c \in_R \mathbb{Z}_q$

- **Response**
  – The prover calculates the other part of the share $c_1 = c - c_2$
  – The response for the actual protocol is calculated as $z_1 = t_1 + c_1 x$ and for the simulated protocol as $z_2 = t_2$.
  – The responses z_1, z_2 and the shares c_1, c_2 are sent to the verifier.

- **Verification**
  – The verifier checks that the challenge was shared correctly $c = c_1 + c_2$
  – The verifier checks the offers conform the ZK protocol i.e. if $g^{z_1} = y_1 h_1^{c_1}$ and $g^{z_2} = y_2 h_2^{c_2}$

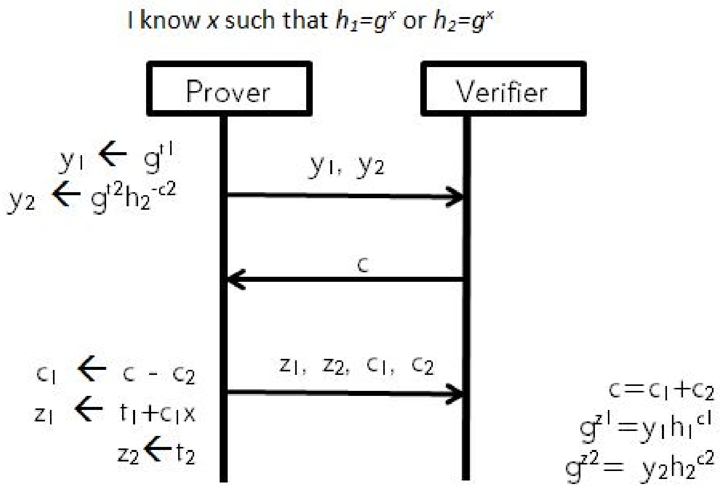


FIGURE 2.7: Schnorr Proof (partial knowledge)

2.1.7.4 The Fiat Shamir Heuristic

The protocols described above are interactive because they require active participation of the verifier. In [55] such interactive protocols can be made non interactive, as the prover can incorporate the role of the verifier and produce the proof transcript all by himself. The transcript can then be validated by any interested party. The key idea of this transformation is that instead of requiring an external random challenge, the prover can create it himself by applying a pseudorandom function to the offer. A common candidate for a pseudorandom function is a hash function. In essence the Fiat Shamir Heuristic transforms a proof scheme to a signature scheme. For practice the non interactive version of the schnorr scheme is:

- Randomly select $t \in_R \mathbb{Z}_q$, calculates $y = g^t$ and sends y its to the verifier
- Generate $c = H(y)$ where H is a hash function yielding values in $\mathbb{Z}_q$
- Calculate $s = t + cx \quad (mod\, q)$

The script of the protocol, that is what should be distributed, becomes (c, s). Anybody can verify it by checking if $c = H(g^s h^{-c})$, which is how the scheme is used in practice. In addition, in order to make the proof dependent on the prover the hash function include some identification information for the prover.

2.1.7.5 Range Proofs

In many scenarios a proof must be given that a ciphertext (or a commitment) corresponds to a number that lies in a specific interval. For example if there are C candidates that are represented by indices $1, \cdots, C$ (or $0, \cdots, C-1$) a valid selection is an integer in the particular range. A a result a voter must provide a relevant proof of vote validity without of course exposing the actual vote. The details of such a proof largely depend on the cryptosystem used, but there are some general techniques available. We shall deal with proofs regarding homomorphic cryptosystems. The relevant bibliography is [23], [88], [87], [66], [26].

- The simplest and most general way such a proof can be achieved, is a disjunction of C WID proofs, that $\mathfrak{c} = 0 \vee \mathfrak{c} = 1 \vee \cdots \vee \mathfrak{c} = C - 1$.
- Another classical case assumes that C is a power of 2, ie. $\mathfrak{c} \in \{0, \cdots 2^l - 1\}$ the voter must prove that her choice is an l-bit number. That is $\mathfrak{c} = \sum_{i=0}^{l-1} b_i 2^i$ where $b_i \in \{0, 1\}$. The prover can release the encryptions of each individual bit and prove that all ciphertexts correspond to a 0 or 1. The final proof will be a conjuction of l WID proofs. If this is not the case than
- More generally for arbitrary ranges $\{a, \cdots, b\}$ it suffices to prove that $\mathfrak{c} - a > 0$ and $b - \mathfrak{c} > 0$ [23]. As a result we must prove that a number is positive. This can be done using a theorem of Lagrange stating that any positive integer is the sum of four squares. A proof is required that a number is a square.

More details will be given in 3 where the case of homomorphic voting will be examined.

2.1.7.6 Plaintext Equivalence Tests [73]

A *Plaintext Equivalence Test*, henceforth referred to as **PET**, is a cryptographic primitive introduced in [73] which aims to convince a set of participants that two ciphertexts indeed encrypt the same plaintext. It is meant to operate in a distributed setting, which means the the decryption key is shared among the participants.

More specifically, in the case of El Gamal, the inputs to the protocol are two ciphertexts $(G, M), (G', M')$ and the output is true if the encrypt the same plaintext and false otherwise. The main idea of the protocol is that the ciphertext $C_{PET} = (\frac{G}{G'}, \frac{M}{M'})$ will be the encryption of 1 if the ciphertexts encrypt the same plaintext. Otherwise it will decrypt to a random integer. More specifically, after all participants have agreed to C_{PET}:

- Each player selects a random integer z_i and commits to it using the Pedersen commitment scheme.
- Each player i blinds C_{PET} using the random value z_i thus producing $(G_i, M_i) = (\frac{G}{G'}^{z_i}, \frac{M}{M'}^{z_i})$
- Everybody proves that the (G_i, M_i) are indeed blinded with the random values.
- The players calculate $(\prod_i G_i, \prod_i M_i)$ and decrypt by combining their shares of the key.
- If the decryption yields 1 the test return *true*, otherwise the test return *false*.

2.1.8 Blind Signatures

Blind signatures were presented by David Chaum in [29] as a means of realising anonymous electronic payments. The concept is very simple and can be presented with a simple analogy. Consider a consumer who wants to pay a merchant some money to purchase a good [30].

- The consumer put a blank sheet of paper into an envelope. On it, there is a carbon sheet attached.
- He sends the envelope into the bank requesting the amount of money that the good costs.
- The bank validates that the consumer has the money into his bank account and signs the envelope.
- The consumer validates the signature and empties the envelope. Due to the properties of the carbon paper the signature is transferred to the enclosed sheet.
- The merchant is paid with the signed sheet of paper.
- She presents it to the bank.
- The bank validates its own signature and pays the merchant the appropriate amount.
- The transaction is complete. Since the bank's signature is transferred to many identical blank sheets, the bank cannot trace who purchased which items.

This analogy can be formalised as follows [81]: A blind signatures scheme is a set of algorithms $(Blind, Sign, Verify, Unblind)$ such that:

1. The message m to be signed is blinded using the Blind Algorithm and some randomness. $b = Blind(m, r)$
2. Afterwards it is signed $S_b = Sign(b)$ in a way that the signature is transferred to the original message.
3. The Unblind function is used on the blind signature obtaining: $S_m = Unblind(S_b)$
4. The signature is verified by executing $Verify(S_m)$

More concretely if Sign and Verify are the classical RSA signatures, we get:

1. $b = Blind(m, r) = r^e H(m) \pmod n$
2. $S_b = Sign(b) = r^{ed \pmod{\phi(n)}} H(M)^d \pmod n = rH(m)^d$
3. $S_m = Unblind(S_b) = S_b \frac{1}{r} = H(m)^d \pmod n$
4. The signature is verified by executing
   $Verify(S_m) =$ if ${S_m}^e = H(m)$ then $True$ else $False$

The anonymity provided by blind signatures has made them an interesting candidate for electronic voting.

2.1.9 Visual Cryptography

Visual Cryptography encrypts written material in a **perfectly secure** way. Decryption is done by human visual system. The ciphertext and the secret key are printed in a special way on two pages of papers. The ciphertext can be transmitted by fax or published. The secret key is printed on transparent paper. Each sheet of paper on its own is indistinguishable from random noise, but when placed on top of each other we have decryption. An accurate analogy to describe visual cryptography would be a visual one time pad The scheme can be extended to k of n threshold secret sharing scheme [97].

In voting visual cryptography is used to issue voter receipts that make coercion difficult. In a system proposed by Chaum [31] the receipts consist of two layers laminated together. A printer prints the selection of the voter but does not conclude the job. The voter selects to keep top or bottom layer and then the printing concludes. The machine keeps the same layer in digital form. Neither layer is readable on its own, but both encode the vote. The other layer is securely destroyed by the voting personnel and the voting booth.

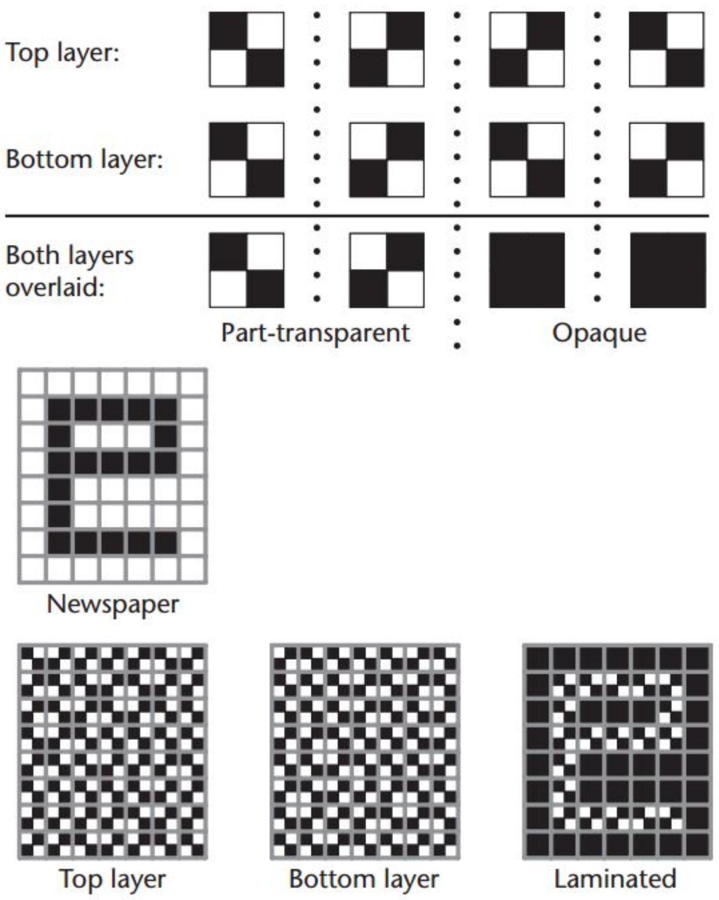


FIGURE 2.8: Visual Cryptography [31]

2.2 Voting Theory

As we mentioned in the introduction, voting is a general decision making process that enables a group of individuals to decide for a course of action or for a representative. The voting process is a two step process. The interested parties indicate their opinions for the various alternatives. Afterwards a computation is applied to the inputs and a decision is reached. If everybody agrees then the decision is simple. However if there are varying choices then a choice must be made that reflects the '*collective preference*' of the group. Voting theory is the field of social choice theory that deals with this problem.

In this section we shall describe the basics of voting methods, borrowing heavily from [105] and [106]. A voting method consists of two parts:

- Allow the voter to express his opinion
- Compute on the opinion

Now based on variations of these 2 actions, some prevalent voting methods are:

Plurality Count This is the simplest and most widely used method. Each voter expresses her opinion by selecting a single candidate, her most preferred one. This can be modelled by creating a vector consisting of c elements. Then the element with index $\mathfrak{c}$ gets the value 1 and the rest retain the value zero. Subsequently, all the voters' choices are aggregated and the candidates are sorted based on the collective preference. In practice, this means that all the vector are added-up, creating the results vector. The favourite candidate's index is the largest element of the vector. One problem with plurality count is the existence of the *spoiler effects* [38], where a candidate that cannot be elected affects the final result. For instance in the 2000 elections in the USA, the result would be different if there was not a third candidate, since the citizens that voted for him, would alternatively pick the candidate that ranked second, hadn't they voted for the one that ranked third, in a numbers that would swing the result [93].

Approval Voting In this voting method, the participants are asked to select more than one candidates, with all the selections being equal in weight. Since no ranking is used, the selection of a candidate from a voter carries less weight than other voting methods. As a result the voter selection is called *approval*. The candidate that gathers the most approvals wins the election. To represent the choice, a vector of binary elements can be used where the one value indicates approval. If we use a vector of ones adding up and looking for the maximum number of approvals will yield the winner.

Borda Count Plurality count, forces the voter to concentrate her opinion on a single choice, while approval voting cannot discern which approval is the strongest. There exist voting methods that allow the voters to convey more information about their thinking by employing candidate ranking. One such voting method is the Borda Count, where the voters:

- Rank the candidates in their order of preference
- The winner between C candidates is computed with the following process:
  - The candidate ranked in position $i, i \in \{1 \cdots C\}$ receives $C - i$ points (or in a variation $C - i + 1$)
  - The Borda score is calculated for each candidate c as $BS(c) = \sum_{i=1}^{C}(C - i) \cdot \sum_{j=1}^{N} r(j, c, i)$ where $r(j, c, i) = \begin{cases} 1, \text{ if voter } j \text{ has ranked candidate } c \text{ in position } i \\ 0 \text{ otherwise} \end{cases}$
  - To put it more simply, the voter rankings' are summed per candidate
  - The winner is the candidate with the highest Borda Score

Limited voting is a voting method that combines the Borda count and approval voting. The voter gives a specified number of approvals to the candidates, with the twist that the same voter can choose a candidate more than once, thus creating a free form ranking.

Runoff Voting In runoff voting the candidates are asked to express their opinion in stages, where in each stage they are faced with a reduced set of candidates. Runoff elections might actually be conducted in multiple phases, which means that election might be conducted many times. Alternatively the reduction might be reflected in the voters' ballots using their ranking. This can happen in two ways:

- The candidate with the most last position rankings is the one selected for elimination (*Comb Rule*)
- The candidate with the fewest first position rankings are removed (*Hare Rule*)

Single Transferable Vote (STV) The objective of STV Voting is to reduce vote waste, for candidates that cannot win the election, or have already been elected. Each voter gets a single vote, but 'this vote' can be used by many candidates. Each voter ranks candidates. Ranking selects the candidates to receive the votes in case favourites are eliminated. When a candidate is eliminated, 'his' votes are transferred to the next ones in the ranking.

Tallying begins, by partitioning ballots according to the first choice of the voter. The candidate with the fewest votes is eliminated. As a result his votes are transferred according to the second choices. This process is repeated until there is only one candidate left, who is declared the winner of the election. If during this process a ballot with all the candidates eliminated is found, it is discarded.

A variation of STV, includes quotas, which is the number of votes that assure election. If a candidate has reached the quota, he cannot be eliminated. In order to make better utilisation of the ballots, the surplus of votes has to be transferred, as well.

In STV, the voter should not provide a complete ranking of the candidates. This can be used against the protocol in what is called the Italian attack. The unused preferences that are left out of a complete ranking can be used as a covert channel, with ower ranked preferences providing the identify of the voter. For instance, an election might mandate maximum number of choices is α. A potential coercer, might require a voter to vote for his β preferred candidates. As proof he might require a particular permutation of the $(\alpha - \beta)!$ left, to be present when the ballots are counted.

Condorcet Voting In this voting method, the voter provides a full ranking of the candidates. All candidates engage in pairwise comparisons (duels). The *Condorcet winner* (loser) of the election is the one whole wins (loses) all the duels. It is evident from the description above that an election might not have a Condorcet winner, so an additional processing is required. This additional processing is referred as *completion method*:

- **Black's procedure**: If there is no Condorcet winner, then the winner of the election is the Borda winner.
- **Dodgson's method**: If there is no Condorcet winner, then then winner of the election is the one that requires the fewest swaps in order to be declared the Condorcet winner. A swap is made in the initial ranking of the candidates by the voters.

Chapter 3

Homomorphic Voting Systems

3.1 Introduction

In chapter 2 we defined the homomorphic property of cryptosystems and secret sharing schemes, which allows operations to be performed on ciphertexts/secret shares that result in potientially different operations performed on the corresponding plaintexts/secrets. If, for instance, the ciphertexts, encrypted under the public key of an election authority, are combined in a way that results in the plaintexts being added, then the election authority can, using its private key, decrypt the combination revealing the election outcome. This property implies that the authority can calculate the election result, without any processing on the actual votes. Only the ciphertexts are used. As a result, ballot secrecy is maintained throughout the election process, which is very desirable. Of course, in this case ballot secrecy depends on some cryptographic assumptions. In theory using secret sharing homomorphisms, these assumption can be lifted. Of course due to the verifiability requirements this is not always the case.

Systems that utilise this homomorphic property have been proposed from the onset of electronic voting. There are, however, many caveats in such efforts, a situation that can be easily predicted from the discussion in Chapter 1. For example, a malicious voter can encrypt the equivalent of 1000 votes, thus adding 1000 votes to the preferred candidate, resulting in *unfair* elections. As a result, ballot secrecy is not enough for fair elections. This means that the homomorphic cryptosystems employed for this purpose need to be augmented with verifiability and robustness in order to qualify for voting protocols, which is the subject of the current chapter. Before we begin we must note that the development of homomorphic protocols was parallel to the mixnet based protocols that we studied in the previous chapter. As a result ideas presented here were used in mixnets and vice versa.

3.2 Early Schemes based on Residuosity Cryptosystems

The first to create a verifiable voting scheme from homomorphic cryptosystems were Cohen(Benaloh) and Fischer in [40]. This basic scheme was improved in [19]. The cryptosystem used is additively homomorphic and is described in 2.1.1.3. It is based on the $r-th$ residuocity problem and is applicable to *yes-no* elections. A more important contribution of this work however is the concept of the bulletin board which permeates all subsequent work. It is modelled after early traditional election systems, like the show of hands where everybody could verify the result, changed however to accommodate for privacy.

Bulletin Board It can be broadly defined as *broadcast channel with memory*. More specifically it:

- Allows appending and reading

- Integrity Preserving, meaning that its contents cannot be modified
- Authenticated, which means that 'it' knows the initial association of voter and vote (of course in encrypted form). The protocol dictates if the vote is stored with the user actual name, a user id, or a user digital signature.
- Accessible to every participant
- Tamper Proof
- Resistant to DoS attacks
- Implemented via Byzantine agreement algorithms. The exact implementation is out of the scope of this thesis (and of the majority of papers for electronic voting).

The participants are N processes that are designated as voters and a centralised authority referred to as the government. The system includes a bulletin board, a source of randomness (beacon) and a timer. The beacon is used for verifying the election. More specifically during various points of the protocol, it provides a sequence of η bits as random challenges to the participating entities. Depending on the values of these beacon bits, the entities either reveal their parameters or perform some calculations that bind them to the correct execution of the protocol. The verifiability of the protocol is a direct consequence of the fact that an entity cannot predict the beacon bits. The timer's function is to signal the execution of the protocols' phases, which are:

1. **Parameter Generation**: The government releases the set of security parameters for the cryptosystem. In particular r is a prime number, $n = pq$ where $r \mid p - 1$ and $r \nmid q - 1$ and a y such that, there exists no x for which $y = x^r \pmod n$. For verifiability the government must prove that the parameters released, possess the above properties. To this end, the government generates η_1 'bundles' of security parameters and a random one is chosen as the one actually used in the election. The randomness is provided by the beacon. The probability of cheating by the government in this phase is $\frac{1}{\eta_1}$ where η_1 is a security parameter.

2. **Ballot Preparation**: Each voter i prepares a ballot that consists of a yes-vote and a no-vote in random order. The yes vote is the encryption of 1, $y \cdot f^r \pmod n$, and the no vote is the encryption of 0: $g^r \pmod n$ where f, g are random values, with $gcd(f, n) = gcd(g, n) = 1$. It must be noted, that the order of the yes and no votes in every ballot constructed by the protocol is random.

   In reality, in a manner analogous to the previous step the voter generates $\eta_2 + 1$ ballots one of which, randomly selected, will be actually used. The ballot used is called the master ballot. The voter must prove that the master ballot is of the correct type, which means that it consists only of a correct encryption of 1 and 0, and not say of 100 and -100. The proof of correctness proceeds as follows:

   - Let $(y \cdot \hat{f}^r \pmod n), \hat{g}^r \pmod n)$ be the ballot to be used by the particular voter.
   - η_2 random bits $\{b_j\}_{j=1}^{\eta_2}$ are generated by the beacon.
   - If $b_j = 1$ the j-ballot is revealed and is unused.
   - If $b_j = 0$ then the voter reveals $\frac{f}{\hat{f}}$ and $\frac{g}{\hat{g}}$

   The main insight for this procedure is that due to the homomorphic property of the cryptosystem used, both $\frac{f}{\hat{f}}$ and $\frac{g}{\hat{g}}$ are $r - th \quad roots$ of encryptions of zero. Indeed $\frac{g}{\hat{g}} = (\frac{g^r}{\hat{g}^r})^{\frac{1}{r}} = (\frac{E(0,g)}{E(0,\hat{g})})^{\frac{1}{r}} =$

$E(0, \frac{g}{\hat{g}})^{\frac{1}{r}}$. The same rationale applies to $\frac{f}{\hat{f}}$. This means that the corresponding components of the master and the test ballots are the same. As a result, if the voter can produce a pair of potential encryption of zero, it means that the master ballot is of the same type as the ballots for which the beacon has sent $b_j = 0$. Since the voter can only guess the beacon bits with negligible probability, it follows that the master ballot is of the same type as all the test ballots, which are encryptions of one and zero in random order.

3. **Voting**: The voter i selects one of the options of the ballot as the vote cast v_i, either $y \cdot \hat{f}^r \pmod n$ for yes or $\hat{g}^r \pmod n$ for no.

4. The government collects the valid votes, (the ones that conform to the parameters), computes the tally and proves that it is correct. The tally is computed by multiplying the valid ballots cast. Since the cryptosystem is additively homomorphic this corresponds to adding the yes votes. The no votes will be calculated using subtraction from the total votes cast. More specifically:

$$\prod_{i=1}^{N} v_i = \prod_{i=1}^{N_1} y \cdot \hat{f_i}^r \prod_{i=1}^{N_0} \hat{g_i}^r = y^t \cdot x^r$$

   The result is an encryption of the sum of the yes votes. The numbers t, x are unknown. The government must calculate them, by applying the decryption algorithm described in 2.1.1.3. This time the value of the randomness x should be calculated as well to prove correct decryption.

5. For the scheme to provide verifiability the tallier must convince that his operations are correct. This is done in the spirit of the protocol by using the beacon.

   - The tallier generates η values c_i coprime to n and publishes $C_i = c_i^r$
   - The beacon generates η bits.
   - For all beacon bits with value 1 the tallier reveals c_i and for all bits with value 0 he reveals $c_i' = c_i x$. The server cannot pass the test if he does not know the randomness used. The potential verifiers check that for the tally released: $y^t c_i'^r = C_i \prod_{i=1}^{N} v_i$

This protocol is robust as no coalition of voters can disrupt an election. In addition it is verifiable. It also provides privacy against other voters. The problem with the scheme presented here is that since the government can calculate the parameters for the final tally, it can progressively calculate the parameters for any intermediate tally, thus violating voter privacy.

A solution to this problem was given a year later. The main idea is to split the function of the government into k entities, the *tellers*. It views the voter as the dealer in a secret sharing scheme (2.1.5). The voter splits the vote into k parts and distributes them to the k telles. If the scheme is homomorphic the authorities aggregate the shares and reconstrunct the result, which produces the tally. This approach was proposed by Benaloh in [19]. Secrecy is achieved by vote sharing.

Each teller j releases its own parameters (n_j, y_j) and each vote consists of k encryptions. During tallying each tallier extracts its own components from the votes submitted and multiplies them together as in [40]. The product is similarly encrypted and the partial sum is extracted. The election tallying is finally calculated as the sum of the partial tallies mod the residuosity parameter r. Of course, protocols are included to verify the validity of the ballot and the final tally. More specifically:

1. **Parameter Generation**: The general parameter r is a number larger than the total voters. Like the government in [40] each teller selects its parameters $n_j = p_j q_j$ where $r \mid p - 1$ and $r \nmid q - 1$ and r non-residue y_j

2. **Parameter Verification**: The tellers demonstrate that indeed y_j is a non residue $\pmod{n_j}$ and the voters prove that they can construct a valid ballot. To prove the validity of a master ballot:

    - A number of test ballots are generated
    - Some of them are randomly decrypted
    - The rest are proved of equivalent in type with the master ballot

    To prove the validity of y_j

    - Each voter randomly selects a vote from a ballot
    - Each teller decrypts its own correspondent component of the vote
    - The voter verifies the decryption
    - If all work out the tellers are honest and voting can proceed

3. **Voting**: The voters generate their ballot and validate it as proposed in the previous step and designate their votes.

4. **Tallying**: Each teller t_j computes the products of the components of the votes $w_j = \prod_{i=1}^{N}(z_i)_j \pmod{n_j}$. Since t_j knows p_j, q_j it can compute τ_j such that $w_j = y_j^{\tau_j} x_j^r \pmod{n_j}$ thus revealing its partial subtotal. The tally of the election is computed as $\sum_{j=1}^{k} t_j \pmod{r}$

3.3 Discrete Logarithm Based Schemes

The complexity of [19] is $O(\eta k^2 N)$ where η is a security parameter. The overhead in the scheme stems from the way the vote is validated, where for each tallier η ballots are generated. More efficient schemes can be constructed using the technique in [42] which was described in 2.

3.3.1 Improving the work required in secret sharing homomorphisms

Another protocol of this form was presented in [43] and relies on Pedersen commitments and secret sharing. The work required by the voter is $O(\eta k)$ and by each individual authority $O(\eta N)$. The scheme requires the existence of a bulletin board and private communication channels between voters and authorities in the form of a public key cryptosystem. The voter initially commits to a vote and posts the commitment to the bulletin board, alongs with a proof of validity. More accurately the voter commits to a masked version of the vote, which enables the actual vote to be revealed in a later time. In order to open the commitments the voter uses the private channels with the authorities, distributing the actual votes so that privacy can be maintained. The authorities accumulate the votes by employing the homomorphic property of the cryptosystem and announce the tally along with a proof of validity.

In more detail the scheme in [43] proceeds in the following phases:

- **Ballot Construction**
    - The $i-th$ voter v_i selects a masked vote b_i as a random value in $\{-1, 1\}$
    - The voter commits to b_i using the Pedersen commitment scheme by calculating $B_i = g^{r_i}h^{b_i}$
    - The voter proves that $b_i = 1$ or $b_i = -1$ by transforming an appropriate ZK proof to a proof of partial knowledge
        * If the voter selects $b_i = 1$ then the ballot becomes $B_{i1} = g^{r_i}h$. She must prove that she possesses the randomness r_i to generate the ballot.

* The voter calculates $y_{i1} = g^{t_{i1}}$
* The prover must create a *bogus* proof using the simulator for $b_i = -1$ which will be $y_{i,-1} = g^{t_{i,-1}}(Bh)^{-c_{i,-1}}$ where $(Bh)^{-c_{i,-1}}$ will be cancelled out.
* The voter sends $B, y_{i1}, y_{i,-1}$ to the verifier
* The verifier generates the random challenge c_i
* The prover splits the challenge $c_{i1} = c_i - c_{i,-1}$
* The prover calculates the normal response $z_{i1} = t_{i1} + c_{i1}r_i$ and the bogus response $z_{i,-1} = t_{i,-1}$ and sends $z_{i1}, z_{i,-1}, c_{i1}, c_{i,-1}$
* The verifier validates all the relations below:
  · $c_i = c_{i1} + c_{i,-1}$
  · $g_{z_{i1}} = y_{i1}(B/h)^{c_i1}$
  · $g_{z_{i,-1}} = y_{i,-1}(Bh)^{c_{i,-1}}$
* The process is analogous if the voter selects $b_i = -1$. In that case the bogus proof will be y_{i1}.

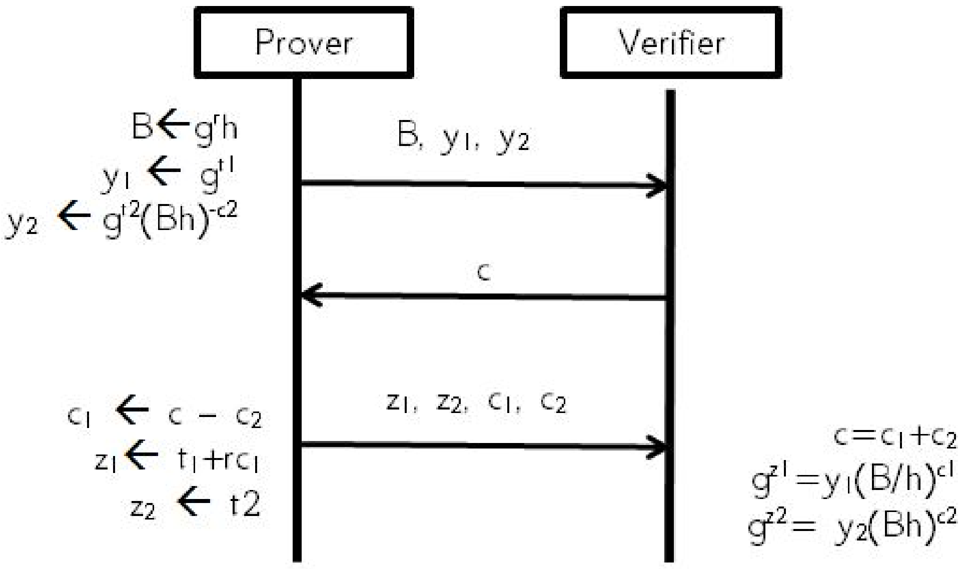


FIGURE 3.1: Proof of validity for yes ballot in [43]

* It should be noted that in both cases the proof can be made non interactive using the Fiat - Shamir technique.

– The voter shares r_i, b_i by computing $t-1$ degree polynomials $G_i(x), H_i(x)$ and commits to the coefficients (this is the Pedersen verifiable secret sharing scheme). Let's denote the commitments $B_{il} = \{g^{r_{il}}h^{b_{il}}\}_{l=1}^{t-1}$
– The voter posts B_i, the validity proof and the coefficient commitments to the bulletin board, where the proof is checked by all interested parties
– The k shares of (r_i, b_i), $\{r_{ij} = G_i(j), b_{ij} = H_i(j)\}_{j=1}^{k}$ are sent to the authorities where they are validated. Each authority A_j checks that $g^{r_{ij}}h^{b_{ij}} = B_i \prod_{l=1}^{t-1} B_{il}^{j^l}$ where B_i, $\{B_il\}_{l=1}^{t-1}$ are retrieved from the bulletin board. If everything is ok the above relation holds since: $B_i \prod_{l=1}^{t-1} B_{il}^{j^l} = g^{G_i(j)}h^{H_i(j)}$.
– It must be noted that the communication in the previous step happens through private channels, bypassing the BB. Since this action is not universally verifiable, it opens up the possibility for *didn't receive/didn't send* complaints, that might block the election.

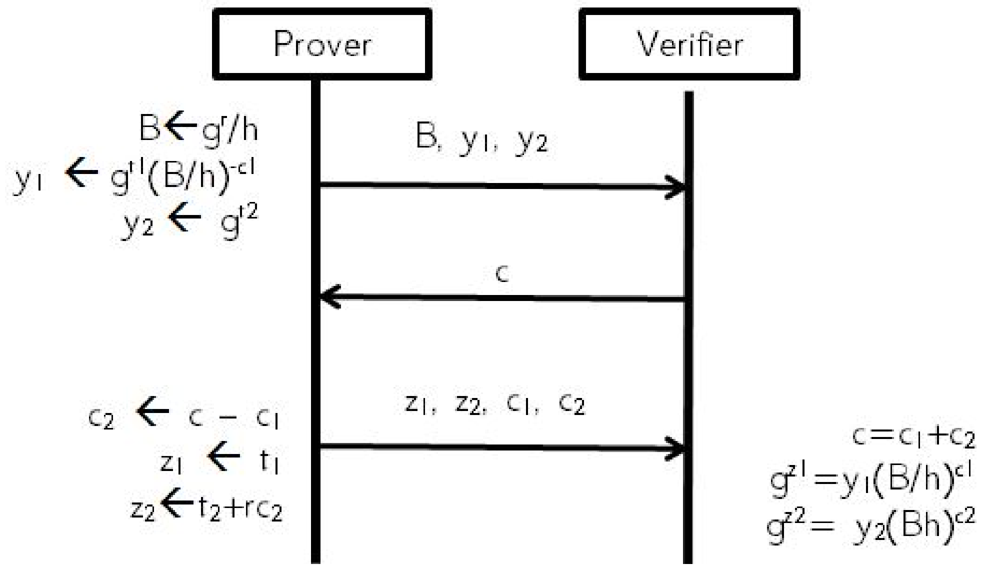


FIGURE 3.2: Proof of validity for no ballot in [43]

- **Vote Casting**
  - The voter forms the desired vote $v_i \in \{-1, 1\}$ by selecting s_i such that $v_i = s_i b_i$. Due to the use of masked ballots the vote casting phase can be separated from the ballot construction phase and carried out as late as possible.
- **Tallying** *using secret sharing homomorphism*
  - The authorities sum their shares r_{ij}, b_{ij} for all the voters' selected values s_i. Authority A_j posts $S_j = \sum_{i=1}^{N} r_{ij} s_i$ and $T_j = \sum_{i=1}^{N} b_{ij} s_i$ and posts them to the bulletin board
  - The partial sums are validated by any interested party that may operate as a prospective tallier. The validation procedure is similar to the one performed by the authorities at the last step of the ballot construction, but this time it is applied to all the voters. More specifically each authority checks if $g^{S_j} h^{T_j} = \prod_{i=1}^{N} (B_i \prod_{l=1}^{t-1} B_{il}^{j^l})^{s_i}$
  - The final tally is computed from the t correct partial tallies of the authorities forming a set A as $T = \sum_{j \in A} T_j \prod_{l \in A-\{j\}} \frac{l}{l-j}$

3.3.2 An optimal scheme for the voter

The second protocol utilising the construction of protocols for proofs of partial knowledge of [42] was presented in [44] and is optimal as far as the voter is concerned, meaning that the voter does as little as possible. This is why it is dubbed as *vote 'n' go* or *single pass* scheme. It relies on El Gamal encryption. The work required by the voter is $O(\eta)$, linear in the security parameter, while the work of each of the k authorities is $O(\eta N)$.

The election scheme of [44] is simple and efficient. Each voter posts her choice encrypted using El Gamal and provides a proof that the vote is valid. The private key corresponding to the public key of the encryption is shared between the election authorities in a threshold manner. The private key is never reconstructed even when the final tally is computed.

In more detail the protocol proceeds in the following phases:

- **Ballot Construction** The vote is cast in two phases in a similar way as in [43]. During the phase of the ballot construction a subgroup of order q and a generator G is selected. A yes vote will be represented as $m_y = 1$ and a no vote as $m_n = -1$ The voter selects again a random $b \in \{1, -1\}$ and prepares the encryption $(x, y) = (g^r, h^r G^b)$. The voter must prove that $b = 1$ or $b = -1$ without of course revealing which one is it. This boils down to a disjunction of two discrete logarithm equalities, namely that $log_g x = log_h(y/G)$ for $b = 1$ or $log_g x = log_h(yG)$ for $b = -1$. This can be done with a partial knowledge version of the Chaum-Pedersen protocol, depicted in the 3.3 and 3.4.

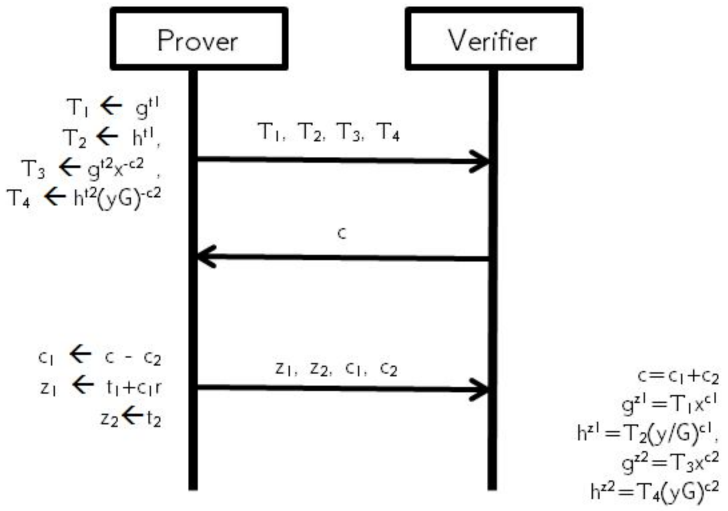


FIGURE 3.3: Proof of validity for yes ballot in [44]

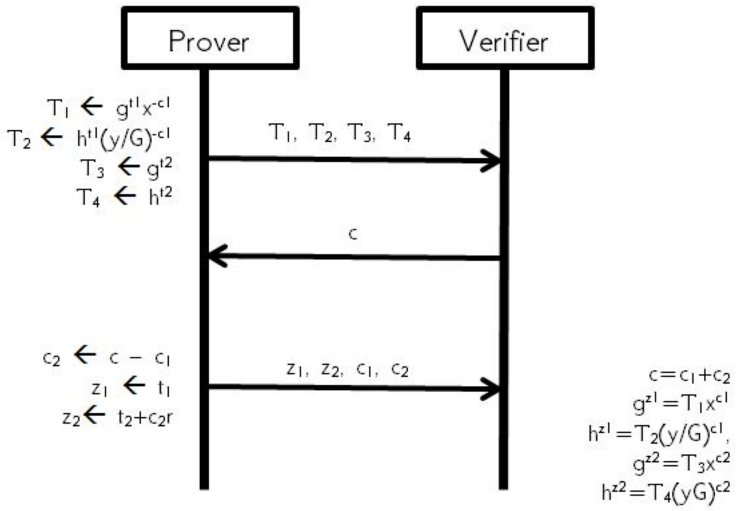


FIGURE 3.4: Proof of validity for no ballot in [44]

Again the Fiat Shamir heuristic can be applied to make the proofs non interactive. In order to avoid vote duplication, the hash of the offer must contain a unique identifier for each voter, such as an id.

- **Vote Casting** The voter forms the desired vote $v_i \in \{-1, 1\}$ by selecting s_i such that $v_i = s_i b_i$ and posts it to the bulletin board.

- **Tallying** When the voting period has ended the talliers check the proofs. All the votes are then multiplied forming $(A,B) = (\prod_{i=1}^{N} g^{r_i}, \prod_{i=1}^{N} h^{r_i} G^{v_i})$. The talliers then execute the share combination and decryption portion of the threshold El Gamal cryptosystem and calculate $W = \frac{A}{B^x}$. Due to the homomorphic properties of the cryptosystem we have $W = G^{\sum_{i=1}^{N} v_i} = G^T$ where T is the tally of the election. In order to retrieve the tally the discrete logarithm $log_G W$ must be computed. This of course is a hard problem. The solution proposed involves trying all the possible values of G and returning the one that matches. More specifically, since $-N \leq T \leq N$, a tallier can successively compute in order $G, G^2, G^3, \cdots$ and so on, until W is found.

3.3.3 Voting using Publicly Verifiable Secret Sharing (PVSS)

The scheme in 3.3.1 has the *didn't send / didn't receive* problem. To deal with cheating authorities and/or cheating voters while maintaining secrecy the verifiability property needs to be public. That is, the validity of the dealer shares should be validateble by everybody. As a result the authors of [122] have created a voting protocol based on their publicly verifiable secret sharing scheme and the bulletin board:

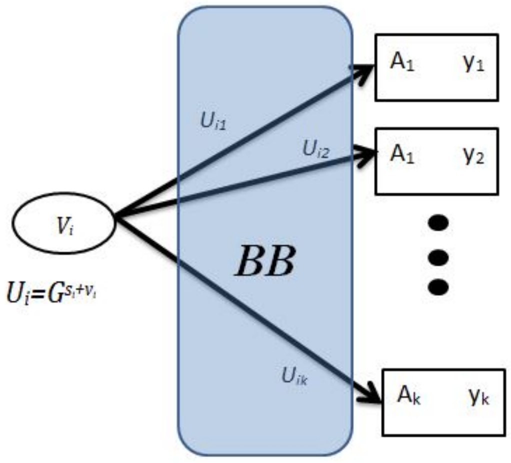


FIGURE 3.5: Voting with Secret Sharing [122]

- Each voter in the scheme, selects the vote $v_i \in \{0,1\}$. In order to create the secret to be shared, she selects a random element s_i and create $U_i = G^{v_i + s_i}$. To share the vote she creates a polynomial $p_i(x) = \sum_{l=0}^{t} a_{il} x^l$. The commitments to the coefficients are denoted $\{C_{il} = g^{a_{il}}\}_{i=0}^{t}$
- The voter constructs a proof of partial knowledge [42] that the vote is valid using the commitments of the previous step. More specifically the voter proves the statement $log_G U_i = log_G C_{i0} \vee log_G U_i = 1 + log_G C_{i0}$. The voter includes in the bulletin board the value U_i
- Each authority A_j has an ElGamal public key $y_j = G^{x_j}$
- Each share $p_i(j)$ is encrypted using the public key of the corresponding authority $Y_{ij} = y_j^{p_i(j)}$
- The voter provers using the Chaum Pedersen Protocol that $log_g(g^{p_i(j)}) = log_{y_j} Y_{ij}$ thus achieving public verifiability.

- The reconstruction phase will not happen per voter, but per authority. Each authority will multiply all the voters' shares producing $Y_j = \prod_{i=1}^{n} Y_{ij} = \prod_{i=1}^{n} y_j^{p_i(j)} = y_j^{\sum_{i=1}^{n} p_i(j)}$
- The result of the reconstruction phase is $G^{\sum_{i=1}^{n} p_i(0)} = G^{\sum_{i=1}^{n} s_i}$. From multiplication of the secrets we get $\prod_{i=1}^{n} U_i = G^{\sum_{i=1}^{n} s_i + v_i}$
- A division yields $G^{\sum_{i=1}^{n} v_i}$. The tally can be computed with exhaustive search since there are n possible values for $\sum_{i=1}^{n} v_i$

This scheme lies in between [44] and [43]. It resembles the second due to the secret sharing approach and the first due to the optimality (everything takes place through the bulletin' board). Their only difference is that the sharing in [44] happens only between the authorities, while in [122] it takes place through the voters. As a result the protocol of [44] may be used when the talliers are few and the voters are orders of magnitude larger, and when the talliers do not change between elections. In contrast [122] is more versatile, can be used when the talliers change between elections and when the voter might be a tallier in smaller scale elections.

3.4 Paillier Based Schemes

The election scheme of [44] is very simple and efficient, as we saw in the previous section. It has however one serious drawback - the need to calculate a discrete logarithm. However the other components are generic and as a result the protocol can be adapted to various cryptosystems. In [46] the generalisation of the Paillier cryptosystem is used. The steps of the protocol are as follows:

- The voters encrypt their votes and prove that the encryption is either a yes vote or a no vote. $E_i = E(v_i, r_i) = (1+n)^{v_i} r_i^{n^s} \quad (mod\, n^{s+1})$ where $v_i \in \{0, 1\}$
- The votes are posted on a bulletin board along with proofs that they contain a valid vote
- Each tallying authority computes the product of the votes
- The authorities jointly decrypt the product of the votes using a threshold scheme like the one in [56] that we described in Chapter 2, thus revealing the election result. This time no extra processing, such as a discrete log computation is required.

The missing piece in the protocol is a description of how to prove whether an encryption corresponds to a yes or no vote. In order to achieve this the protocol in **??** will be used as a basis, as depicted in figure 3.6.

The ciphertexts in the case of $yes - no$ votes will be $E_i = (1+n) * r^{n^s}$ and $E_i = r^{n^s}$ respectively. As a result the voter must prove that $\frac{E_i}{(1+n)}$ or E_i encrypts the message zero. Since the tally is computed $(mod\, n^s)$ then $n^s > N$ must hold.

3.5 Optimisations and extensions to multiple candidates

The homomorphic schemes presented in the previous sections are described mainly for $yes - no$ elections and can be used without any changes when there are two candidates. They can be extended. however. when there are $C > 2$ candidates. In such a case the voter can select either $1 - out - of - C$ candidates or $t - out - of - C$ candidates. As is the general case in the schemes studied in this section the secrecy of the vote leads to a need to prove that it is of valid form and that the voter followed

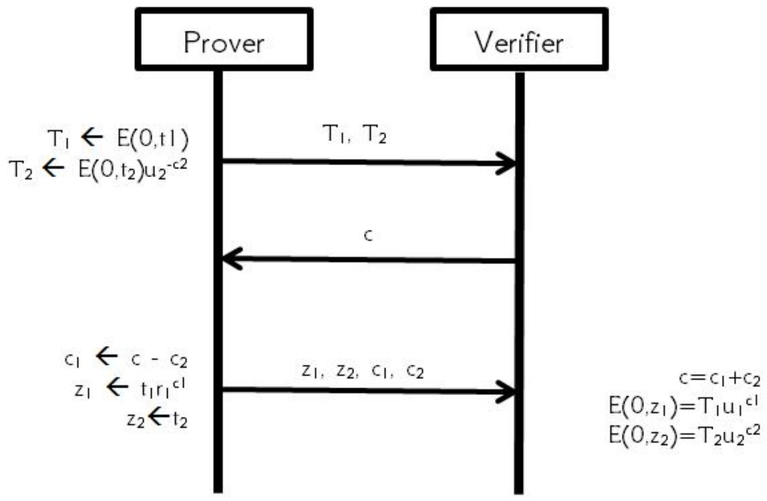


FIGURE 3.6: Proof of partial knowledge of n^s power [46]

the protocol exactly, which, in our case means, that she voted precisely $t - out - of - C$ *different* candidates.

The simplest way to extend the schemes, was presented early in [19], and works by creating a super ballot for C yes-no elections. If $1 - out - of - C$ candidate is to be voted, then this super ballot will contain 1 yes vote and $C - 1$ no votes. In order to tally the elections there must be C counters where the corresponding ballots are aggregated.

The idea of the multiple counters can be applied to the framework of [44]. In the first step of the protocol however an addition must be made in order to prove the validity of the vote. The Chaum Pedersen Protocol must be used with C generators and a proof must be given that exactly one of them was used. The final tally is calculated by solving C discrete logarithms since the decrypted value will be of the form $G_1^{T_1} \cdots G_C^{T_C}$. It is easy to see that the vote size in the above cases will be C times the vote size of the corresponding $yes - no$ protocols. The most important problem is that declaration of the winner requires the calculation of a discrete log of a larger number than the single vote case, which is a difficult computational task.

In the Paillier version of [44] the voter ballot must consist of exactly t yes votes and $C - t$ no votes. As a result the entry for voter i in the bulletin board will contain C votes and C proofs namely $\{E_{ij} = E(v_{ij}, r_{ij})\}_{j=1}^{C}$ and proofs that $\frac{E_{ij}}{(1+n)}$ E_{ij} are encryptions of zero, where $j \in \{1, \cdots C\}$. However the voter must take an extra step to prove that there are exactly t yes votes or equivalently that $\prod_j E_{ij} = (1+n)^t \prod_j r_{ij}^{n^s} = E(t, r)$. To allow the verification despite of the encryption the voter computes the product of the randomness $r = \prod_j r_{ij}$ and posts it to the bulletin board. Now every interested verifier can compute $\frac{E(t,r)}{r} = (1+n)^t$ which is public information. As far as the other case is concerned, namely that the voter may choose up t votes, an easy solution will be to include some dummy candidates. The votes remaining to reach number t are case on the dummy votes.

An improvement has been proposed in [14] and independently in [46] that reduces the vote size. Both approaches utilise the Paillier cryptosystem, which has an efficient decryption process, however the first one relies on the *standard* version, while the second one uses the generalised version. The main idea in both approaches is that a number D is selected and used as a *single counter* for voting. In order to vote for candidate c, instead of encrypting c, the system encrypts D^c. As the encrypted votes are multiplied, the homomorphic property creates the tally to be decrypted as $T = \sum_{c=0}^{C} t_c D^c \quad (mod\, n^s)$, a number in $base - D$. In a single decryption we can obtain T and retrieve the digits, which will be the individual subtallies of each one of the C candidates. In order to implement this idea the protocol parameters must be tweaked and some new proofs must be added. This is were the two approaches differ.

In [14] D is a number larger than the number of voters N, for example $D = 2^{\lceil log_2 N \rceil}$. The tallying process is hierarchical and takes place in three levels: local, regional and national. There are authorities in each level respectively A_{ij}, A_i, A with public keys pk_{ij}, pk_i, pk. The corresponding private keys are shared in a threshold manner. In order to vote for candidate c the voter creates 3 encryptions of D^c E_l, E_r, E_n with the public keys of the local, regional and national authorities. In addition the voter creates 3 proofs that c is a valid candidate index by proving that it lies in the interval $0, \cdots, C-1$ and a proof that all ciphertexts encrypt the same plaintext.

In the first case, the proof can be considered a WID variation of the proof in Chapter 2 that a voter knows the message that corresponds to an encryption. Here the voter wants to prove that he knows the index c so that the encryption $e_c = g^{m_c} r^n \pmod{n^2}$ corresponds to a m_c from the set $\{m_1, m_2, \cdots, m_C\}$. Of course, the actual message m_c remains hidden. The verifier is convinced though that the encryption comes from the set. The proof consists of the following steps:

- The prover creates
  - $c-1$ simulated offers $T_j = (\frac{g^{m_j}}{e_c})^{t_j} r_j^n \pmod{n^2}$ where $t_j \in_R \mathbb{Z}_n^*$, $r_j \in_r \mathbb{Z}_n$, $j \in \{1, \cdots, C\} - \{c\}$
  - and a single valid one $T_c = r_c^n \pmod{n^2}$

  and sends all T_i's to the verifier.
- The verifier sends a random challenge t
- The voter computes $t_c = t - \sum_j t_j$ and $T_c = r_c r^{t_c} g^{t_c \text{ div } N}$ and sends (r_i, t_i) to the verifier
- The verifier validates that
  - $t = \sum_i t_i$
  - $r_i^n = T_i(\frac{e_c}{g^{m_i}})_i^t \forall i$

The tallying occurs in three levels. First the local authorities compute and decrypt (using the share combination algorithm) the product of the local votes (the ones encrypted with the local authority public key), calculating the local partial sum. Each local authority computes without decrypting the product of the votes encrypted with the regional and national public keys and forwards the computation to the regional level. The regional authority combines the regional products provided by the subordinate local authorities and decrypts using the combination algorithm. Then it validates that the sum of local authorities' decryptions match the result of its own decryption. The same process happens at the national level, where the election result is computed.

The approach in [46] tweaks the system based on the value of s. More specifically $\sum_{c=0}^{C} t_c D^c < n^s$ must hold, so $ClogD < \eta s$. More importantly, for the vote to be considered valid the voter **must prove**:

- the ballot encrypts properly a number of the form D^c
- the exponent $c \in \{0, ..., C-1\}$

Proving that the encrypted vote corresponds to a proper power of the basis D involves the following steps, based on the classical range proof technique that we first referred to in 2.1.7.5. To sum it up, the prover should commit to all the ηbits of the witness, then prove that each committed value corresponds to a 0 or a 1 while maintaining that exactly η commitments are used (nothing is left out).

This will be accomplished with the help of a new protocol that enables a prover to convince a verifier that encryptions e_a, e_b, e_c correspond to plaintext a, b, c such that $ab = c \pmod{n^s}$. The protocol is depicted in the figure below:

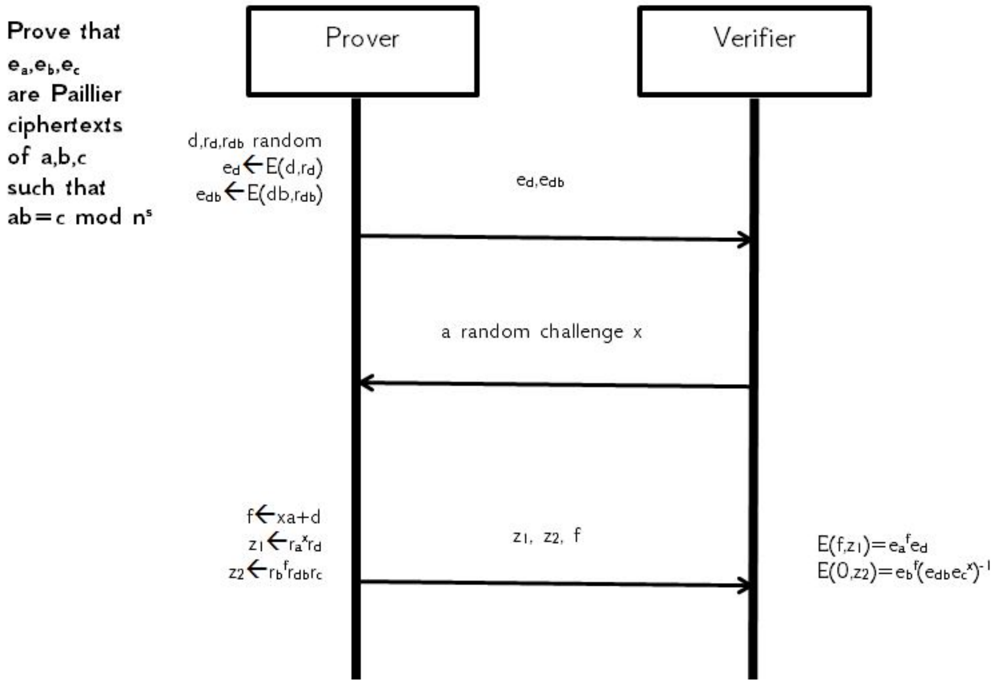


FIGURE 3.7: Proof of Paillier encrypted product [46]

- **The number of candidates is a power of 2** $C = 2^{l+1}$
  - Convert the candidate index power to its binary representation $c = b_0 2^0 + b_1 2^1 + \cdots + b_l 2^l$ where $b_j \in 0, 1$.
  - Then $D^c = D^{2^{0^{b_0}}} D^{2^{1^{b_1}}} \cdots D^{2^{l^{b_l}}}$ which means that D^c is a product of powers of two or ones.
  - The voter encrypts each product term providing encryptions $e_0 = D^{2^{0^b_0}}, e_1 = D^{2^{1^b_1}}, \cdots, e_l = D^{2^{l^b_l}}$
  - There are $l+1$ encryptions, exactly the same as the number of bit - length of C. As a result it is proved that $0 \leq c < C$.
  - The voter uses the proof of partial knowledge of the protocol in 3.6 to prove that either $\frac{e_i}{(1+n)^1}$ or $\frac{e_i}{(1+n)^{D^{2^i}}}$ is an n^s power, which we described earlier when discussing ballot validity.
  - The voter calculates the partial products $E_i = \prod_{x=1}^{i} D^{2^{x^b_x}}$ such that $E_l = D^j$
  - The voter uses the partial products with the 3.7 proof with $a = E_{i-1}, b = e_i, c = E_i$ to conclude the proof.
- **The number of candidates is not a power of 2** The first three steps of the previous case are repeated. However we cannot depend on the number of bits to conclude if $c < C$ so we must add extra steps.
  - Convert, again the candidate index c to its binary representation $c = b_0 2^0 + b_1 2^1 + \cdots + b_l 2^l$ where $b_j \in 0, 1$.

- Then $D^c = D^{2^{0^{b_0}}} D^{2^{1^{b_1}}} \cdots D^{2^{l^{b_l}}}$
- The voter encrypts each product term providing encryptions $e_0 = D^{2^{0}{}^{b}_{0}}, e_1 = D^{2^{1}{}^{b}_{1}}, \cdots, e_l = D^{2^{l}{}^{b}_{l}}$
- Define $\beta_i = \frac{1}{(D^{2^i}-1)} \pmod{n^s}$
- Compute $e'_i = \frac{e_i}{(1+n)}^{\beta_i} \pmod{n^2} = Enc(b_i)$ since $b_i = 0$ or $b_i = 1$
- $C = (B_l \cdots B_0)_2$
- $c < C \Leftrightarrow \exists i > 0 : B_i = 1$ and $b_i = 0$ and $\forall j > i : B_j = b_j$
- Need to prove equality of j indices and inequality of i
- $B_i = b_i \Leftrightarrow z_i = \frac{(2B_i-1)(2b_i-1)+1}{2} = 1$
- $Z_i = z_l \cdots z_{i+1} B_i(b_i - 1) = 1 \Leftrightarrow c < C$
- Use the product proof with $a = Z_{i+1}, b = z_i, c = Z_i$
- Use WID of n^s power protocol to show that $\exists i : Z_i = 1$

All the above hold for the case of a single selection. They can be extended for multiple selections by allowing t ballots and checking that they don't encrypt the same candidate choice. This can be done by computing the pairwise differences of the votes and proving that each one is not an encryption of zero.

3.6 Variations for alternate voting methods

A homomorphic voting system is an ideal candidate for elections that compute on alternate social choice functions. The reason behind this, is the fact that the votes remain secret during the entirety of the process. This is important because such an alternate function might convey more information about the voter opinion, than a traditional yes or no vote, or a one out of C vote, making relevant the italian attack mentioned in Chapter 1. If the vote remains secret through the entire process, such an attack is made infeasible. Moreover the level of secrecy required is is an interesting question. A further topic of difficulty in such systems are the increased efficiency difficulties from the complexity of the required zero knowledge proofs. Such protocols are therefore worthy of further examination.

Borda Count In [133] a protocol is given to homomorphically implement a Borda count. The scheme, is quite restrictive, releasing only the winner of the election, without revealing either the order of the losing candidates or the points they accumulated. The rationale behind this level of secrecy is the fact, that if the total points of a candidate are known, then $N-1$ voters might collude in order to reveal the points awarded by the remaining voter. In addition, it utilises a distributed setting a la [44], taken to its extreme (full privacy), namely the secret key for the decryption of the tally is shared between all voters. This contradicts, the privacy of the score of each candidate, as according to the motivation of full privacy if $n-1$ voters can collude, than maybe elections are not necessary.

- The voting process begins by registration and calculation of the El Gamal secret keys and the joint public counterpart.
- Each ballot is an encrypted vector $B_i = \{Enc(2^{p_{ic}})\}_{c=1,i=1}^{C,N}$ where p_{ic} are the points awarded to candidate c by voter i. From the specification of the Borda protocol it follows that $p_{ic} \in \{1, \cdots C\}$.
- The voter must prove that $p_{ic} \in \{1, \cdots C\}$ and that each component of the ballot is an encryption of $2^{p_{ic}}$

- The authority has the single task to multiply the components of all the ballots for each voter $\prod_{i=1}^{N} B_i = \{\prod_{i=1}^{N} Enc(2^{p_{ic}})\} = \{Enc(2^{\sum_{i=1}^{N} p_{ic}})\}$. The homomorphic property has placed the total points of each candidate in the exponent.

- The authority must find the component with the largest exponent without exposing the exact value of the collected points. To this end a comparison operator must be implemented between 2 ciphertext c_1, c_2, for which the only available information is that they each encrypt a power of 2, say m_1, m_2, the total points for a candidate according to the protocol. This operator will be used to find the maximum value and will utilise the homomorphic nature of the cryptosystem:

  – The ciphertexts are divided $c = \frac{c_1}{c_2} = Enc(2^{m_1 - m_2})$ where $m_1, m_2 \in \{N, \cdots, NC\}$.

  – Consequently $m_1 - m_2 \in \{-N(C-1), \cdots, N(C-1)$.

  – As a result, $m_1 \geq m_2 \Rightarrow c \in \{\{Enc(2^i)\}_{i=0}^{N(C-1)})\}$

  – The tallier must compute the encryptions in set $\{Enc(2^i)\}_{i=0}^{N(C-1)}$ accompanied by a proof of knowledge.

  – Afterwards the tallier computes the ciphertexts $c_i = \{\frac{c}{Enc(2^i)}\}_{i=0}^{N(C-1)})$. This calculation utilises public values and can be verified by all participants. It is evident that if $m_1 \geq m_2$ one of the items of the set should be the encryption of 1.

  – A straightforward decryption would reveal the total points. To avoid this, the items in the set are mixed and then threshold decrypted (in a manner similar to [73]).

  – If an item with the value 1 is found in the list then we can deduce that $m_1 \geq m_2$. Otherwise it follows that $m_1 < m_2$

- Now that a comparison operator has been implemented, the winner can be found using $O(C)$ comparisons.

Another approach to homomorphically implement the Borda count is given in [66]. This approach, is more relaxed as it does not aim to hide the votes that were received by each candidate. All the preferences are aggregated in a single vote using Baudron counters. Moreover, the fact that each ranking is a permutation π of the set of candidated indices $\{0, \cdots, C-1\}$ is also used. As a result the input of the voter is a number in base-D with digits in the set of candidates, namely $v = \sum_{i=0}^{C} -1\pi(i)D^{i-1}$. To prove the correctness of the vote, the following non interactive protocol is used:

- $\mathcal{P}$ proves that π is indeed a permutation. To do so, a protocol proposed in [65] will be used. Since the proof is aimed at proving the correctness of a permutation,it is applied in verifying the shuffle of a mixnet. It will be described in greater detail in its natural place, in Chapter 4.3.4.

- $\mathcal{P}$ uses a typical proof of knowledge of the plaintext.

Chapter 4

Mixnets

4.1 Overview

In this chapter we shall study extensively the concept of a **mix net**, firstly introduced by David Chaum in [28]. The mix net is a general anonymity primitive that is, itself, constructed using building blocks from Chapter2. As such, it has been used in systems that provide anonymous services i.e. browsing, auctions and of course elections.

A mixnet $\mathcal{MN}$ anonymises a set of input items, by garbling and shuffling them. It is composed of entities called **mix servers**, $\{M_j\}_{j=1}^{k}$. Each mix server receives a set of input items $\{v_i\}_{i=1}^{n}$. To anonymise them, it breaks the link between input and output, by changing their exit position and form. More formally, an item entering as v_i will exit as $v'_{\pi(i)}$ where π is a random permutation. Since an item traveling through the mix net always changes exit position and form it cannot be tracked. Of course to be of some use, an authorized recipient should have a mechanism to reconstruct the input items from the output items. The mechanism used for this task is cryptography. Upon entrance the input items are encrypted. Inside the mix net the items are processed and in the output they are decrypted. When they are used for voting the mix servers communicate through a bulletin board $\mathcal{BB}$, akin to the one assumed in chapter 3. Traditionally the mix servers operate in sequence. However, on a practical level the mix servers can operate in parallel [49]. For k mix servers the votes are split into k distinct groups. Each group is assigned to one mix server that performs the shuffling and transformation. After the operation each group is assigned to next mix server (group i to server $(i+1) \pmod{k}$). In the last phase the first group will be processed by the $k-th$ server.

In the context of voting, the anonymised items using the mixnet are the votes. They enter the mixnet in encrypted form. However, since the bulletin board is authenticated, each vote can be linked to a voter (maybe each vote is digitally signed). The authentication information is validated and stripped, and the votes are then processed. The shuffling and garbling that takes place, break the link between vote and voter. Thus in the output the votes can be decrypted and counted. This general idea has many variations (and unfortunately caveats), which we shall review in this chapter. However it is important to stress from the onset, that the decryption of votes opens up an array of possibilities regarding the social choice functions that can be used. This is an important departure from the framework described in the previous chapter.

In the theory of mixnets, one honest mix server suffices to achieve privacy. However as we shall see, there are attacks, that can use implementation specific flaws, and break the anonymity of the senders even in the presence of an honest mix server.

The research on mix nets is very rich and they can be considered a mature technology. the literature is full of different notations. One objective of this chapter is to homogenize the notation.A thorough exposition is given in [3]. We build on it and try to enrich with newer approaches.

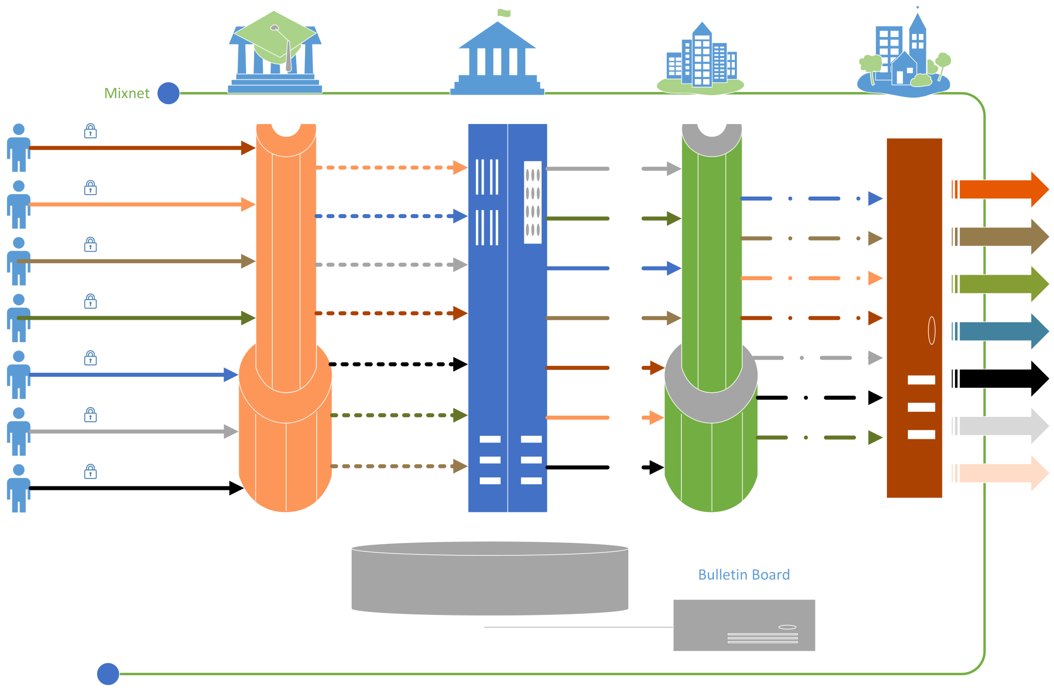


FIGURE 4.1: Mixnet Operation

4.2 First Generation Mixnets

4.2.1 RSA Mixnets

The original mix net idea proposed in [28], operates in layers, akin to onion peeling. As a result the implementation of mix nets for anonymous browsing is also called *onion routing*. Each mix server has a key pair (pk_j, sk_j) for the RSA cryptosystem where pk_j is published.

- Each sender (the voter in our case) encrypts the message by employing the public keys in reverse order and input the messages to the mix net.

$$L_0 = \{Enc_{pk_1}(Enc_{pk_2}(\cdots Enc_{pk_k}(v_i, r_i)\cdots, r_2), r_1)\}_{i=1}^{n}$$

- Each mix server removes a layer of encryption using its private key and removes the randomness applied to each input. This actions changes the form of the input message.

- For the shuffling part, a random permutation is applied to each item and the result is outputted. For example the output of the first mix server is

$$L_1 = \{Enc_{pk_2}(\cdots Enc_{pk_k}(v_i, r_i)\cdots, r_2)\}_{i=\pi_1(1)}^{\pi_1(n)}$$

- This process is repeated. In the end, the last mix server has fully decrypted the items.

$$L_k = \{v_i\}_{i=\pi_k\circ\cdots\circ\pi_1(1)}^{\pi_k\circ\cdots\circ\pi_1(n)})$$

- In the case of voting, the votes can be counted.

The Chaumian mix - net anonymizes the input messages, by breaking the link between input and output, but there are some remarks to be made:

- The communication between the mix servers takes place through the bulletin board, which is available to all interested parties.
- M_k has access to the plain text.
- A mix server can block the mixing process by refusing to decrypt.
- The number of encryptions needed and the ciphertext size are proportional to the number of mix servers, since each mix server adds its own randomness.

4.2.2 Reencryption Mixnets

A variation of the Chaumian mixnet was presented in [108]. It is based on the idea of reencryption for El Gamal that was described in chapter 2. Reencryption implements the transformation requirement and thus makes decryption redundant. It comes in two flavours, one combining the Chaumian idea with re-encryption and one employing solely re-encryption. Again each mix server has its own key pair. In El Gamal notation:

- The input list is

$$L_0 = \{(g^{r_{0i}}, v_i(y_1 \cdots y_k)^{r_{0i}})\}_{i=1}^N$$

  This is exactly the same idea as previously, with the only difference that the encryptions are not layered, but combined using an aggregate public key $\prod_{i=1}^k y_i$ where y_i is the public key of each mix server. As a result the ciphertext size and the number of encryptions needed are constant and do not depend on the number of mix servers.

- M_j partially decrypts each item in

$$L_{j-1} = \{(g^{\sum_{t=0}^{j-1} r_{ti}}, v_i \cdot (\prod_{t=j}^k y_t)^{\sum_{t=0}^{j-1} r_{ti}})\}_{i=1}^N$$

  by dividing with $g^{x_j \cdot \sum_{t=0}^{j-1} r_t}$ and rerandomises with r_{ji} producing

$$L_j = \{(g^{\sum_{t=0}^{j} r_{ti}}, v_i \cdot (\prod_{t=j+1}^k y_t)^{\sum_{t=0}^{j} r_{ti}})\}_{i=1}^N$$

- Then a random permutation π_j is applied.
- As previously. the last mix server fully decrypts the items.

The second flavor uses only reencryption and only one decryption takes place. Each M_j:

- The input list is $L_0 = \{(g^{r_{0i}}, v_i \cdot y^{r_{0i}})\}_{i=1}^N$ - that is, there exists a single key pair for all mix servers.
- re randomises each item in

$$L_{j-1} = \{(g^{\sum_{t=0}^{j-1} r_{ti}}, v_i \cdot y^{\sum_{t=1}^{j-1} r_{ti}})\}_{i=1}^N$$

  producing

$$L_j = \{(g^{\sum_{t=0}^{j} r_{ti}}, v_i \cdot y^{\sum_{t=1}^{j} r_{ti}})\}_{i=1}^N$$

- applies a random permutation π_j

The decryption key is shared by a number of trusted parties (can be the mix servers themselves) in a threshold manner (ala [44]). In this variation all mix servers are of equal power since no one has access to the plain text without the cooperation of others. The remaining combination, namely a decryption only mixnet, cannot be applied when the underlying cryptosystem is El Gamal, as it would create a way to track the messages, since the first part of each ciphertext would remain unchanged. In other cryptosystems such as Paillier, it would be feasible, though.

This mixnet can provide anonymity and individual verifiability in elections, the former by definition, the latter by embedding redundant information in the form of randomness in each vote and searching for it in the output list.

However this randomness can be used as a receipt to prove how one voted [84]. The issue of receipt freeness will be extensively studied in chapter 5.

4.2.3 Tagging Attacks

The first attacks on mix nets were described in [112] and [111] for Chaumian and reencryption mix nets respectively. These attacks are quite powerful and permeate the entire mixnet literature. Their nature is active. The main idea that underlies them is to tag a particular item, in a way that is *shuffling* - and *garbling* - *proof*. As a result, the mixing can be bypassed, by looking for the tag in the output. Thus the sender's privacy can be broken. To create the tag, they benefit from the malleability implied by the homomorphic properties of the underlying cryptosystems. For example $E(m)^k = E(m^k)$ in the case of multiplicative homomorphism.

In [112] the tagging attack is employed against a Chaumian mix net, where randomness of length b is applied to message of length B, for the reasons explained in 2.1.1.1. As a result the mix net input becomes $Enc(m,r) = (r \cdot 2^B + m)^e \pmod n$ and the output is m.

The attack works as follows:

1. $\mathcal{A}$ wants to track message m through the mixnet, which *travels* as $Enc(m,r)$
2. $\mathcal{A}$ chooses a small factor f
3. $\mathcal{A}$ input the following *tagging* message $Enc(m,r) \cdot Enc(f) = (r \cdot 2^B + m)^e \cdot f^e \pmod n$. The encryption $Enc(m,r)$ is freely available in $\mathcal{BB}$
4. $\mathcal{MN}$ operates as usual, thinking that its input is of the general form: $(r^* \cdot 2^B + m^*)^e$
5. This also applies to the tagging message, which means:

$$(r^* \cdot 2^B + m^*) = (r \cdot 2^B + m) \cdot f \Rightarrow$$
$$(m^* - f \cdot m) \cdot 2^{-B} = (f \cdot r - r^*)$$

6. $\mathcal{MN}$ outputs both m^*, m in the decrypted list
7. $\mathcal{A}$ know the randomizer space, namely that :$r, r^* \in \{0, \cdots, 2^b - 1\}$ This implies that $(f \cdot r - r^*) \in \{2^{-b} + 1, \cdots, f \cdot (2^b - 1)\}$
8. $\mathcal{A}$ can search the output list for messages (m, m^*) for which $(m^* - f \cdot m)$ has a value in $\{2^{-b} + 1, \cdots, f \cdot (2^b - 1)\}$
9. [112] prove that such messages can be found with high probability.

In [111] the tagging attack is employed against a reencryption mix net.

- Again the attacker $\mathcal{A}$ wants to track input v_i for participant P_i.
- He can retrieve the initial encryption $c_{i0} = (g^R, v_i \cdot (y_1, \cdots, y_k)^R)$.
- Each mix server M_j receives the input: $c_{ij} = (g^{R'}, v_i \cdot (y_j, \cdots, y_k)^{R'})$ which is a tuple (t, u)
- For some random x known to her, $\mathcal{A}$ can generate $c''_{ij} = (t^x, u^x) = (g^{R'x}, v_i^x \cdot (y_j, \cdots, y_k)^{R'x})$
- $\mathcal{MN}$ output will contain both v_i^x, v_i due to the homomorphic nature of the underlying cryptosystem.
- $\mathcal{A}$ can retrieve all decrypted output messages and raise them to the x.
- Then she can check for duplicates. When a duplicate is found, $\mathcal{A}$ deduces that she has found the message tracked from the output list, since $v^x_{\pi(i)} = v^x_i$. As a result she learns the vote cast in $v_{\pi(i)}$

In both attacks, it is assumed that $\mathcal{A}$ can inject messages of her preference into the $\mathcal{BB}$. This is reasonable despite the authenticating nature of the bulletin board. For instance, the attacker can cooperate with a legitimate voter, and issue the tracking message using his credentials.

4.3 Verifiable Mixnets and proofs of shuffles

The attacks on the previous section raised some awareness on what can go wrong with mixnets.

- At the initial encryption stage, an encryption of a vote not known by the user might enter the mixnet. As we saw $\mathcal{A}$ copies the encryption of a legitimate vote and uses it to track a message. However there are other possibilities as well. For instance, in *vote copying*, $\mathcal{A}$ can copy exactly the ciphertext found in $\mathcal{BB}$. This, in some scenarios can have the effect of *vote canceling*, as an exact duplicate of a vote is found.
- More importantly it raises the following question. If a user has so much power, if she cheats, what can be said about a cheating mix server. More specifically processess all the encrypted votes. A cheating mix server might change them by applying the copying attack, or it might drop some of them all together and replace them in the output, controlling in effect the complete election. Additionally a cheating mix server might not apply any permutation, reencrypt the votes in a known manner allowing an attacker to get hints about the link between the voter and the vote.

The problems regarding cheating mix servers stem from the fact that the they are viewed a sort of black box, whose actions must be blindly trusted. However this powerful entity has a pivotal role in the election, so we cannot simply trust it. A mix server must verify that it operated correctly. To do this, we can use techniques from chapter 2, namely zero knowledge proofs in order not to give away the actions of both the voter and server. More specifically:

- The voter must provide a zero knowledge proof, that he knows the contents of the vote, and that his input is not simply a reencryption of some other input.
- Each mix server must provide a zero knowledge proof that he correctly applied re encryption or processing required by the protocol specification
- Each mix server must provide a zero knowledge proof that he shuffled correctly and that no messages were altered, dropped or replaced.

The general idea in proving that a voter knows the contents of the vote is a Schnorr based protocol to prove knowledge of discrete logarithm.

In order to prove correct shuffling, we must prove [114] that each encrypted vote in the input, appears in the output, which means equivalently that each encrypted vote in the input, has a re-encryption in the output, which corresponds to the following NP statement:

$$\begin{aligned}\exists \text{ permutation } \pi \text{ on } \{1,\cdots,n\}, \exists s_1,...,s_n,\\ \forall \text{ vote } i \in \{1,\cdots,n\},\\ \forall \text{ mix server } j \in \{1,\cdots,k\}:\\ \text{If } C_{i,j} = (a_{i,j}, b_{i,j}) \text{ and } C_{\pi(i),j+1} = (a_{\pi(i),j}, b_{\pi(i),j}) \text{ then}\\ a_{\pi(i),j+1} = a_{i,j}\cdot g^{s_i} \text{ and } b_{\pi(i),j+1} = b_{i,j}\cdot y^{s_i}\end{aligned}$$

As the above statement is in NP, there exists a corresponding zero knowledge proof. The most practical protocol to prove the above statement is the Chaum-Pedersen protocol [33] (repeated many times) to prove equality of discrete logarithms.

These observations gave birth to the second generation of mixnets, *verifiable mixnets*. The schemes of this category provide universal verifiability and as a result individual verifiability.

4.3.1 Sako Killian mix net

The first universally verifiable mixnet was proposed by Sako and Killian in [84]. It builds upon an ElGamal based re-encryption [108], for which:

- The $i-th$ voter's choice is v_i
- Each mix server M_j has its own ElGamal secret key: x_j and public key: $y_j = g^{x_j}$
- The input to the mix net is the list $L_0 = (G_0, M_0) = \{(g^{r_0}, v_i \cdot (y_1 \cdots y_k)^{r_0} \quad (mod\, p))\}_{i=1}^{n}$
- Each Mix server before permutting
  - Rerandomises its input list using his randomisation factor r_j producing $G_j = G_{j-1}\cdot g^{r_j} = g^{r_0+r_1+...+r_j}$
  - Partially decrypts by adding the random exponent and dropping one public key $M_j = M_{j-1}\cdot (y_j...y_k)^r_j / G_j^{x_j}$
- The last mix server decrypts $M_k/G_k^{x_k} = m$ permutes and announces.

As we mentioned before, verifiability requires that each mix server verifies its operation. In this case it **must**:

- Prove correct partial decryption
- Prove correct reencryption and shuffling

The solution given in [84] was a cut and choose protocol with $\frac{1}{2}$ soundness, which means that a mix server might cheat in reencryption and shufflin gand getaway with it. In order to avoid this situation however the protocol can be repeated η times reducing the cheating probability to $(\frac{1}{2})^{\eta}$, but increasing the complexity.

More specifically in order to prove the correctness of the partial decryption, each mix server must prove knowledge of secret key and correct decryption without of course revealing anything. It assumes the role of a prover in the cut and choose protocol. More specifically it must prove:

1. that $H = G^x mod p$ where the common input is $(G, H, g, y = g^x mod p)$
2. The mix server sends $(y', G') = (g^r \pmod p), G^r \pmod p)$ where $r \in_r \mathbb{Z}_p$
3. The verifier with probability $\frac{1}{2}$ requests r or $r' = r - x$
4. The prover complies with the request, providing either r or r'
5. If the r is revealed to the verifier, then it checks that $y' = g^r \pmod p$ and $G' = G^r \pmod p$. This is the cheating case, since the private key is not used
6. If r' is revealed then the verifier checks that $y' = g^{r'} y mod p$ and $G' = H \cdot G^{r'} mod p$

The verification can be batched, executing one time for each mix server and not per vote:

- We proved that for each message $H = G^x mod p$
- Instead of proving correct encryption for each vote, a mix server proves correct encryption of all the votes.
- The verifier chooses random exponents c_i (one for each vote).
- Calculate $\prod_i H^{c_i}$ and $\prod_i (G^x)^{c_i}$
- Prove that $\prod_i H^{c_i} = \prod_i (G^x)^{c_i}$
- The proof implies that all encryptions are done correctly

This cut and choose general idea can be employed to prove correctness of the shuffle. The trick is that each mix server generates another secret permutation and another set of randomization values. It uses them to perform a *secondary shuffle*, both reencryption and permutation. When challenged, it reveals either the secondary shuffle or the difference between primary and secondary shuffle. To be more specific:

- The common input is (g, w, A, B) where
  - g, w are constants
  - $A = (a_i^{(1)}, a_i^{(2)})$ is the input encryption tuple to the mix server
  - $B = (a_{\pi(i)}^{(1)} \cdot g^{r'_{\pi(i)}}, a_{\pi(i)}^{(2)} \cdot w^{r'_{\pi(i)}})$ is the output reencrypted tuple
- To prove that B can be generated from A, the mix server $\mathcal{P}$ generates a new permutation λ and randomization values t_i.
- $\mathcal{P}$ sends $C = (a_{\lambda(i)}^1 \cdot g^{t_{\lambda(i)}} mod p, a_{\lambda(i)}^2 \cdot w^{t_{\lambda(i)}} mod p)$
- $\mathcal{V}$ requests (λ, t_i) or $(\lambda' = \lambda \circ \pi^{-1}, t'_i = t_i - r'_i)$ with probability $\frac{1}{2}$
- $\mathcal{P}$ complies by revealing the requested items
- On receiving the values, $\mathcal{V}$, checks C by definition, if λ, t_i was requested or
- if λ', t'_i was requested then for $B = (b_i^{(1)}, b_i^{(2)})$, checks if:

$$C = (b^{(1)}_{\lambda'(i)} \cdot g^{t'_{\lambda'(i)}}, b^{(2)}_{\lambda'(i)} \cdot w^{t'_{\lambda'(i)}} modp)$$

i.e.. $\lambda(i) = \lambda' \circ \pi(i)$ or $\lambda(i) = \lambda \circ \pi^{-1} \circ \pi(i)$

It is easy to see, that the verification complexity of the Sako-Killian mixnet is $O(\eta k N)$ for error proability $\frac{1}{2^\eta}$.

An attack There was an implementation error in [84] which allowed an attacker to break voter privacy even when there is only one honest mix server M_j and without need for colluding voters. This attack was documented in [94].The attack happens before M_j performs the shuffle. The first $j-1$ corrupt mix servers trace the message to be revealed. The last $k-j$ servers reveal the message. In more details:

- Voter Alice sends to M_1 her input v as $(G_1, M_1) = (g^{r_0}, v \cdot (\prod_{t=1}^{k} y_t)^{r_0})$
- Honest Mix Server receives: $(G_j, M_j) = (g^{r_0+R}, v \cdot (\prod_{t=j}^{k} y_t)^{r_0+R})$ where $R = \sum_{t=1}^{j-1} r_t$
- Honest Mix Server follows the protocols and reveals in order to prove that he can partially decrypt correctly. $H_j = G_j^{x_j} = (g^{r_0+R})^{x_j}$
- The rest of the corrupted servers contribute their private keys x_j and compute:

$$\frac{M_j}{H_j \cdot \prod_{t=j+1}^{k} G_j^{x_t}} = \frac{v \cdot (\prod_{t=j}^{k} y_t)^{r_0+R}}{(g^{r_0+R})^{x_j} \cdot (g^{r_0+R})^{\sum_{t=j+1}^{k} x_t}} = v$$

There is a simple countermeasure to this attack which is first to perform the shuffling and then the re-encryption.

In [1] a more efficient variation of the [84] protocol was proposed. To put it simply, [1] made universal verifiability of [84] *independent* of the number k of mix servers ($O(\eta N)$ for error probability $\frac{1}{2^\eta}$), by having them present a single aggregate proof which is checked by the verifier. It assumes a reencryption based El Gamal mixnet, where the decryption key is split among the mix servers. The scheme employs a bit commitment scheme, that allows each server to commit to the secret permutation and the randomization values used. The aggregate proof scheme works as follows:

- Each mix server, $\mathcal{P}$, cuts by performing a secondary mix with permutation λ_j and random values $\{t_{ij}\}_{i=1}^{N}$, thus producing a new output list.
- Verifier issues a random bit c as his challenge
- If $c = 0$:
  - $\mathcal{P}$ commits to λ_j, $\{t_{ij}\}_{i=1}^{N}$ and distributes the commitments to the rest of the servers.
  - After all commitments are distributed they are opened and are validated by each participant
  - The last mix server publishes the composition of λ of λ_j and the sum of random values following the permutations namely $\hat{t}_{ik} = \sum_{j=1}^{k} t_{i\lambda_j \cdots \lambda_1(i)}$
  - Each mix server validates the items published by the last mix server
- If $c = 1$:
  - $\mathcal{P}$ publishes the permutation $\phi_j = \pi_j^{-1} \phi_{j-1} \lambda_j$ and randomisation values $\{w_{ij} = w_{\lambda_j(i)(j-1)} + t_{\lambda_j(i)(j-1)} - r_{\phi_{j-1}\lambda_j(i)(j-1)}\}_{i=1}^{N}$ where the initial values are the identity elements.
  - The final mix server publishes $\phi_k = \pi_k^{-1} \cdots \pi_1^{-1} \lambda_k \cdots \lambda_1$ and $\{\hat{w}_{ik}\}_{i=1}^{N}$

As can be seen in response to both challenges the last mix server publish an aggregate value. As a result the will be only one verification. Since the protocol has $\frac{1}{2}$ soundness it can be repeated η times. A similar proof can be given for correct decryption. While the verification complexity decreases to $O(\eta N)$, the communication complexity of the scheme increases.

4.3.2 Permutation and Sorting Networks

The first efforts to eliminate the costly cut and choose based verification of [84] and [1] were presented in [92] and in the *Millimix network* ([91]). The former is based on permutation networks, which is a $N \times N$ construction that receives as input a list of N items and outputs them in permutted order for a specific permutation of N items. The latter is based on sorting networks, which are permutation networks for which the selected permutation sorts the input items based on the permutted indices. Both can be recursively constructed from 2×2 elementary constructions, called *switching gates* or *comparators* respectively, and utilise the homomorphic properties of El Gamal.

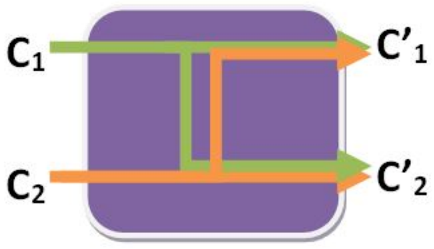


FIGURE 4.2: Reencryption in a 2×2 mixnet

A simple 2×2 mixnet can be constructed from such a switching gate. Its input is two ciphertexts $C_0 = Enc(m_0, r_0)$ and $C_1 = Enc(m_1, r_1)$. It applies a reencryption to the inputs resulting in $C'_0 = Reenc(C_0) = Enc(m_0, r_0 + r'_0)$ and $C'_1 = Reenc(C_1) = Enc(m_1, r_1 + r'_1)$. The output is C'_b and C'_{1-b} where $b_R \in \{0, 1\}$. In a comparator the output is not selected randomly but it implements a function f such that: $f(x,y) = \begin{cases} (x,y), & \text{if } x < y \\ (y,x), & \text{if } x \geq y \end{cases}$, where x, y are the permutted indices. In both cases, if the encryption scheme is semantically secure, then an adversary cannot distinguish which input C_0, C_1 encrypts m_0 and m_1. This implies that they cannot tell which reencrypted output corresponds to m_0, m_1 as well.

This simple 2×2 mixnet can be made verifiable by using a providing a proof that each C'_b is a reencryption of either C_0 or C_1. More spefically:

Proposition 4.1. *The ciphertext* $C'_1 = (G'_1, M'1) = (g^u, m'_1 \cdot y^u)$ *is a reencryption of the ciphertext* $C_1 = (G_1, M_1) = (g^t, m_1 \cdot y^t)$

Proof. C'_1 is a reencryption of C_1 iff they both encrypt the same message, meaning that $m'_1 = m_1$. By dividing the parts of both ciphertexts we get:

$$\frac{G'_1}{G_1} = \frac{g^u}{g^t} = g^{u-t}$$
$$\frac{M'_1}{M_1} = \frac{m' y^u}{m y^t} = y^{u-t}$$

Equivalently we need to show that $\frac{G'_1}{G_1}, \frac{M'_1}{M_1}$ have the same logarithm in bases g, y respectively, that is $log_g \frac{G'_1}{G_1} = log_y \frac{M'_1}{M_1}$. This can be achieved using the Chaum-Pedersen protocol we reviewed in 2.1.7.2.

This is the approach defined in [92]. A simpler but equivalent method is described in [72], where it is noted that if C'_1 is a reencryption of C_1 then the ciphertext $(\frac{G'_1}{G_1}, \frac{M'_1}{M_1})$ is an encryption of 1 with randomness $r = u - t$. Then $Y = \frac{G'_1}{G_1}^{z_1} \frac{M'_1}{M_1}^{z_2} = (g^{z_1} y^{z_2})^r = G^r$, where $z_1, z_2 \in_R \mathbb{Z}_q$. Proof of reencryption now becomes equivalent to running the Schnorr protocol for knowledge of discrete log r of Y in base G. □

The correctness of the permutation corresponds to the following statement:

Proposition 4.2. *Prove that $\{C'_0, C'_1\}$ is a reencryption of a permutation of $\{C_0, C_1\}$ without revealing the correspondence.*

Proof. We must prove that C'_0 reencrypts C_0 and C'_1 reencrypts C_1 *or* that C'_0 reencrypts C_1 and C'_1 reencrypts C_0.

$$(C'_0 = Reenc(C_0) \wedge C'_1 = Reenc(C_1)) \bigvee (C'_0 = Reenc(C_1) \wedge C'_1 = Reenc(C_0))$$

What is required is the execution of 4 Chaum-Pedersen protocols. In order not to leak the permutation however this proofs must be converted to witness indistinguishable using the framework we described in section 2.1.7.3.

The above relation is equivalent to:

$$(C'_0 = Reenc(C_0) \vee C'_0 = Reenc(C_1)) \bigwedge (C'_1 = Reenc(C_1) \wedge C'_1 = Reenc(C_0))$$

since:

$$(C'_0 = Reenc(C_0) \vee C'_1 = Reenc(C_0)) \text{ and } (C'_1 = Reenc(C_1) \vee C'_1 = Reenc(C_0))$$

are true and cancel out.

This is a conjunction of 2 proofs of knowledge of discrete logarithm (see 2.7). □

Alternatively [91] one can prove that $C'_0 = Reenc(C_0) \vee C'_0 = Reenc(C_1)$ and that $C_0 C_1 = Reenc(C'_0 C'_1)$.

Having created a private, verifiable and correct 2×2 mixnet we can generalise it to a $N \times N$ mixnet. As a result, the resulting mix network as a whole will simulate the permutation: $\pi = \pi_n \circ \cdots \circ \pi_1$. In the literature for permutation networks it is well known that such a permutation network can implement all permutations and requires $\Omega(nlogn)$ gates [41]. One way to compose the switching gates is the Benes Network ([92]).

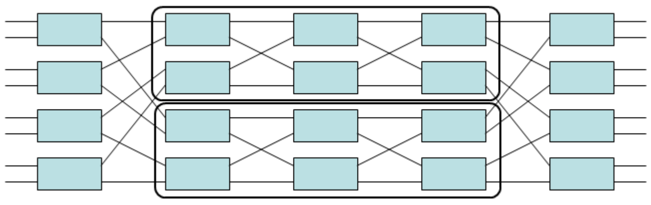


FIGURE 4.3: Recursive construction of a permutation network (Modified from `http://goo.gl/iNiwXB`)

The question that remains is how to distribute the switching gates to the mix servers. Firstly both propositions on mixing with permutation networks agree on the use of threshold El Gamal. That is, all the mix servers obtain a share x_j of the decryption key x and calculate their respective public key $y_j = g^{x_j}$ which is a share of the public key y that is used to encrypt the votes, which are subsequently written in the $\mathcal{BB}$. Subsequently the mixnet operates. There are two modes of operation, borrowing from [108]:

- **Reencryption** [91] lets each M_j uses the network in its entirety. [92] assigns a subset of the gates to each mix server. Both proceed in the same serial manner however: After the execution of the protocol from M_j, verification of the generated proofs take place. If the proofs validate the next mix server uses the sorting network and so on. On the other hand if the proofs of a mix server do not validate, then his output is ignored, that is M_{j+1} operates on the output of M_{j-1}. This process is repeated $t+1$ times where t is the number of honest mix servers. After the mixing process has concluded the mix servers cooperate to decrypt the output.
- **Decryption** The second mode of operation suggested in [92] assigns a subset of the gates to each mix server in a way that M_j controls a column. Each M_j validates the proofs of every previous one. If the proofs of a single server $M_{\hat{j}}$ are invalid then all outputs from $\hat{j}$ to $j-1$ is ignored, and M_j operates on the output of $M_{\hat{j}-1}$. To do so however the output of the intermediary mix servers needs to be decrypted. As a result $t+1$ mix servers must cooperate to decrypt it. After all these tasks M_j operates the column it is assigned and produces the required proofs. This mode of operation results in a decrypted output.

Both mixnets based on permutation networks exhibit complexity of $O(tnlog_2n)$. This might seem larger than the $O(\eta kn)$ complexity of the [84] mixnet, but for smaller populations it is actually more efficients due to the smaller constant used. A security issue in [92] was discovered by [2]. The problem regards permutation networks that can generate (due to the number of switching gates they contain) more than the $N!$ permutations of N items. Then some permutations have more representations then other. If the switching gates operate randomly and independently (e.g. by outputting (C_0, C_1) or (C_1, C_0) using a random control bit), it is proved that the network as a whole is biased and selects the permutations with more representation more often. A simple solution is to preselect a permutation, uniformly before the mixing process and set the switching gates accordingly.

4.3.3 Furukawa - Sako

During that time, Furukawa and Sako [59] proposed another mixing technique that except for verifiability was very fast ($18n$ exponentiations). Their paper focuses completely on verifying a shuffle without examining applications in voting or elsewhere. The main idea behind their work was to represent mixing with permutation π by using a permutation matrix, which is a matrix A with:

$$A_{ij} = \begin{cases} 1, \pi(i) = j \\ 0, \text{otherwise} \end{cases}.$$

For example the permutation $\pi(1,2,3) = (2,3,1)$ can be represented using:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

As a consequence shuffling and reencryption of a list of El Gamal ciphertexts $L_j = \{(g_i, m_i)\}_{i=1}^{N}$ to produce $L_{j+1} = \{(g'_i, m'_i)\}_{i=1}^{N}$ takes the form:

$$g'_i = g^{r_i} \cdot g_{\pi^{-1}(i)} \quad (mod\, p) = g^{r_i} \cdot \prod_{j=1}^{n} g_j^{A_{ji}} \quad (mod\, p)$$

$$m'_i = y^{r_i} \cdot m_{\pi^{-1}(i)} \quad (mod\, p) = y^{r_i} \cdot \prod_{j=1}^{n} m_j^{A_{ji}} \quad (mod\, p)$$

Note that if $\pi^{-1}(i) = j$, then $\pi(j) = i$ which means that $A_{ji} = 1 \pmod q$. Since A is a permutation matrix, for each i, a single A_{ji} in the products above is $1 \pmod q$.

With this representation, to verify the shuffle of L_j to L_{j+1} M_j must:

1. Prove that there exist randomness r_i and matrix $[A_{ij}]$ that satisfy the above relations for all (g_i, m_i) and (g'_i, m'_i).
2. Prove that to generate each output pair, the same r_i, $[A_{ij}]$ have been applied to the input pair.
3. Prove that the matrix $[A_{ij}]$ is a permutation matrix.

The first two items can be proved using a variation of the Chaum-Pedersen Protocol (2.1.7.2).

In order to prove that A is a permutation matrix [59] use the following observation:

Proposition 4.3. *A matrix $[A_{ij}]$ is a permutation matrix iff $\forall i, j, k$ the relations below hold:*

$$P_{ij} = \sum_{h=1}^{n} A_{hi} \cdot A_{hj} = \begin{cases} 1 & i = j \\ 0 & \textit{otherwise} \end{cases}$$

$$P_{ijk} = \sum_{h=1}^{n} A_{hi} \cdot A_{hj} \cdot A_{hjk} = \begin{cases} 1 & i = j = k \\ 0 & \textit{otherwise} \end{cases}$$

It is easy to see that the relations above hold in the case of a permutation matrix. The intuition, for the reverse, is that the equations, imply that exactly a single item x for each row and column is non zero. Moreover $x^2 = 1 \pmod q$ and $x^3 = 1 \pmod q$, which implies that $x = 1$.

As a result the protocol for proving a shuffle becomes a combination of the following proofs:

1. **P1** Prove that $\mathcal{P}$ knows r_i, A_{ji} such that $g'_i = g^{r_i} \cdot \prod_{j=1}^{n} g_j^{A_{ji}} \pmod p$ where $[A_{ij}]$ satisfies $P_{ij} = 1$ if $i = j$ and $P_{ij} = 0$ otherwise.
2. **P2** Prove that $\mathcal{P}$ knows r_i, A_{ji} such that $g'_i = g^{r_i} \cdot \prod_{j=1}^{n} g_j^{A_{ji}} \pmod p$ where $[A_{ij}]$ satisfies $P_{ijk} = 1$ if $i = j = k$ and $P_{ijk} = 0$ otherwise.
3. **P3** Prove that the exponents used are the same in both of the relations above
4. **P4** Prove that the same exponents are used for both parts of the input items (g_i, m_i)

As far as the first part of **P1** is concerned, that is, knowledge of r_i, A_{ij} and correctness of the transformation, the verifier will check a single relation. That is the proof will be a batch proof:

- **Offer**:
  - $\mathcal{P}$ selects random values α, $\{a_i\}_{i=1}^{n}$ to blind the randomization and matrix values respectively.
  - He calculates $g' = g^a \prod_j g_j^{a_j}$
  - He sends g'
- **Challenge**: $\mathcal{V}$ sends the n random challenges $\{c_i\}$
- **Response** $\mathcal{P}$ computes and submits $s = \sum_j r_j c_j$ and $\{s_i = \sum_j A_{ij} c_j + a_i\}$

- **Verification:** $\mathcal{V}$ checks if $g^s \prod_j g_j^{s_j} = g' \prod g_j^{c_j}$

  Completeness follows since:

$$g^s = g^{\sum_j r_j c_j + a} = g^a \prod_j g^{r_j c_j}$$

$$\prod_j g_j^{s_j} = \prod_j g_j^{\sum_k a_{jk} c_k + a_j} = \prod_j (g_j^{a_j} \prod_k g_j^{a_{jk} c_k})$$

  By combining and rearranging we get:

$$g^s \prod_j g_j^{s_j} = (g^a \prod_j g^{r_j c_j})(\prod_j (g_j^{a_j} \prod_k g_j^{a_{jk} c_k})) =$$

$$= (g^a \prod_j g_j^{a_j})(\prod_j (g^{r_j c_j} \prod_k g_j^{a_{jk} c_k})) = g' \prod_j (g^{r_j} \prod_k g_j^{a_{jk}})^{c_j} =$$

$$g' \prod_j g_j'^{c_j}$$

In order to prove the second part of **P1**, namely that the entries $[A_{ij}]$ indeed satisfy that $P_{ij} = 1 \Leftrightarrow i = j$ and $P_{ij} = 0 \Leftrightarrow i \neq j$ the following steps will be merged with the proof above.

- **Offer**: $\mathcal{P}$ generates a random value σ and utilises α, $\{a_i\}_{i=1}^n$
- **Challenge**: $\mathcal{V}$ has already sent the n random challenges $\{c_i\}$
- **Response**: $\mathcal{P}$ calculates:
  - $\dot{w}_i = g^{\sum_j 2a_j A_{ji} + \sigma r_i}$
  - $\dot{w} = g^{\sum_j a_j^2 + \sigma\alpha}$
- **Verification**: $\mathcal{V}$ checks if $w^s g^{\sum_j (s_j^2 - c_j^2)} = \dot{w} \prod_j \dot{w_j}^{c_j}$

  For completeness it must be noted that

  $s_j^2 = (\sum_k A_{jk} c_k + a_j)^2 = (\sum_k A_{jk} c_k)^2 + 2a_j \sum A_{jk} c_k + a_j^2$

  The first term reduces to c_j^2 if the matrix A has property we want to prove. As a result $s_j^2 = c_j^2 + 2a_j \sum A_{jk} c_k + a_j^2$

  As a result $w^s g^{\sum_j (s_j^2 - c_j^2)} = w^s g^{2a_j \sum_k A_{jk} c_k + a_j^2} = g^{\sigma(\sum_j r_j c_j + \alpha)} g^{\sum_j 2a_j \sum_k A_{jk} c_k + a_j^2}$

  By rearranging the terms we get: $w^s g^{\sum_j (s_j^2 - c_j^2)} = g^{\sigma\alpha + \sum_j a_j^2} g^{\sigma \sum_j r_j c_j} g^{\sum_j 2a_j \sum_k A_{jk} c_k}$ which can easily lead to the required relation.

The proof **P2** from the 4.3.3 is implemented in a similar manner. It is more complex however since the hiding of the permutation table that satisfies the second property creates a cubic polynomial. The final protocol is the combination of **P1**, **P2**, **P3**, **P4**. It is complete, sound upon the DLOG assumption and does not leak any information about the permutation. The number of exponentiations required for the protocol is $18n$ and the communication complexity is $n2^{11}$ bits. In a subsequent paper [58], the previous proof of a shuffle is combined with a decryption proof, to yield a complete mixnet and a voting protocol.

4.3.4 Neff Verifiable Mixnet

Around the same time as [59], Andrew Neff [98] proposed a new proof of a shuffle of ElGamal cipher texts that can be used in a verifiable mixnet. It is unconditionally sound and computationally zero knowledge. Its main idea resides on a key property of polynomials: *A polynomial is unaffected by permutation of the roots:* $\prod_{i=1}^{n}(m_i - x) = \prod_{i=1}^{n}(m_{\pi(i)} - x)$. To utilise this idea both inputs and outputs are represented as polynomials P_{in}, P_{out} with coefficients $\in Z_q$. The verifier chooses a random $t \in Z_q$ and evaluates both input and output polynomials. If mixing is correct the results match with very high probability. The scheme was improved in [99].

Specifically, the Neff proof consists of three subprotocols:

- Iterated Logarithmic Multiplication Proof Protocol (**ILMPP**): A generalisation of Chaum and Pedersen protocol to prove equality of the product of a sequence of discrete logarithms
- Simple n-shuffle Proof Protocol: It verifies the evaluation of a pair of polynomials at a common random point when the coefficients are known.
- General n-shuffle Proof Protocol: It verifies the evaluation of a pair of polynomials at a common random point when the coefficients are not known.

At first we shall describe the ILMPP subproof: Both the shuffler ($\mathcal{P}$) and the verifier ($\mathcal{V}$) know $\mathbb{Z}_p$ sequences $\{X_i\}_{i=1}^{n}, \{Y_i\}_{i=1}^{n}$. The prover also knows $\{x_i = log_g X_i, y_i = log_g Y_i\}_{i=1}^{n}$ and wants to convince the verifier that $\prod_{i=1}^{n} x_i = \prod_{i=1}^{n} y_i$ without exposing the logarithms. For $n = 2$ it is the Chaum Pedersen protocol with the two sequences being $\{h_1, g_2\}, \{h_2, g_1\}$ (as in 2.1.7.2) respectively. The proof takes place in 3 rounds:

1. $\mathcal{P}$ picks $n-1$ random offer values $\theta_i \in \mathbb{Z}_q$ and uses them to compute and send n values A_i:

$$\begin{aligned} A_1 &= Y_1^{\theta_1} \\ &\dots \\ A_i &= X_i^{\theta_{i-1}} \cdot Y_i^{\theta_i} \\ &\dots \\ A_n &= X_n^{\theta_{n-1}} \end{aligned}$$

2. $\mathcal{V}$ generates a random challenge $\gamma \in \mathbb{Z}_q$
3. $\mathcal{P}$ calculates $n-1$ random response values $r_i = \theta_i + \gamma(-1)^i \prod_{j=1}^{i} \frac{x_j}{y_j}$ and sends them to $\mathcal{V}$ [1]
4. The verifier accepts if all the above relations hold.

$$\begin{aligned} Y_1^{r_1} &= A_1 \cdot X_1^{-\gamma} \\ &\dots \\ X_i^{r_{i-1}} \cdot Y_i^{i} &= A_i \\ &\dots \\ X_n^{r_{n-1}} &= A_n \cdot Y_n^{(-1)^{n-1} \cdot r_{n-1}} \end{aligned}$$

[1]In order to keep the exposition simple we do not deal with edge cases which almost never occur. The full exposition can be found in [98].

It is proved that the subproof is complete, that is pairs $\{\theta_i, r_i\}$ can always be found if the prover indeed knows the logarithms $\{x_i, y_i\}$ and $\prod_{i=1}^{n} x_i = \prod_{i=1}^{n} y_i$. In addition it is sound with probability $\frac{1}{q}$ and honest verifier zero knowledge.

The second part of the Neff scheme is called the *simple n-shuffle proof* protocol. It allows a prover to commit to a permutation without revealing it, by using the ILMPP protocol. The public data of the protocol are again n $\mathbb{Z}_p$ sequences $\{X_i\}_{i=1}^{n}, \{Y_i\}_{i=1}^{n}$ and two commitments C, D. $\mathcal{P}$ has also private inputs the discrete logarithms of the sequences and the opening of the commitments. The objective is to convince the verifier that he knows the a permutation π such that $Y_i^d = X_{\pi(i)}^c$ without revealing any discrete logarithm or permutation. Note that the permutation is implicit in the ILMPP, however in this case it is blinded using the committed values. The simple n-shuffle proof protocol is a 4 round protocol. More spefically:

- Verifier challenges with random $t \in \mathbb{Z}_q$
- Both $\mathcal{P}$ and $\mathcal{V}$ compute C^t, D^t and run the ILMPP with the following $2n$ sequences: $(\frac{X_1}{D^t}, \cdots, \frac{X_n}{D^t}, C, \cdots, C)$ and $(\frac{Y_1}{C^t}, \cdots, \frac{Y_n}{C^t}, D, \cdots, D)$

The above protocol is complete because, successful execution of the ILMPP protocol will mean that: $c^k \prod_{i=1}^{n}(x_i - dt) = d^k \prod_{i=1}^{n}(y_i - ct)$. By dividing with $c^n d^n$ and reversing the signs, we get the equivalent relation: $\prod_{i=1}^{n}(t - \frac{x_i}{d}) = \prod_{i=1}^{n}(t - \frac{y_i}{c})$ which can be translated as equality of n degree polynomials in t. If $dy_i = cx_{\pi(i)} \Rightarrow \frac{y_i}{c} = \frac{x_i}{d}$ then the polynomials are obviously equal.

Soundness is probabilistic, since equality without knowing the logarithms and the permutation might mean that the challenge was a root. This can occur with probability $\frac{n-1}{q}$, since the n degree terms cancel out. Alternatively, it might mean that $\mathcal{P}$ tries to cheat. This happens with probability $\frac{n-1+q}{q^2}$.

In a typical shuffle of El-Gamal ciphertexts, however, the prover does not know the discrete logarithms of the input sequence. As a result the previous proof must incorporate this scenario. This gives us the General n-shuffle Proof Protocol. The public input of the protocol are again n $\mathbb{Z}_p$ sequences $\{X_i\}_{i=1}^{n}, \{Y_i\}_{i=1}^{n}$ and a single commitment C, D. In this case $\mathcal{P}$ can only open the commitments. The objective is to convince the verifier that he knows the a permutation π such that $Y_i^d = X_{\pi(i)}^c$ without revealing any discrete logarithm or permutation. The main idea of the proof is to have $\mathcal{P}$ verifiably commit to a permutation by executing the *simple n-shuffle proof* using random exponents exchanged with $\mathcal{V}$. Using the committed permutation then, $\mathcal{V}$ is convinced that $\mathcal{P}$ indeed knows the $d$such that$D = g^d$. In more details the protocol consists of the following steps:

- Commit to permutation
  - $\mathcal{P}$ generates randomly two non zero n-item sequences of exponents $\{e_{1i}\}, \{e_{2i}\}$ and sends $\{E_{1i} = g^{e_{1i}}\}, \{E_{2i} = g^{e_{2i}}\}$ and sends them to $\mathcal{V}$
  - $\mathcal{V}$ challenges with two different non zero n-item sequences $\{f_{1i}\}, \{f_{2i}\}$
  - $\mathcal{P}$ calculates $\{F_{1i} = g^{df_{1\pi^{-1}(i)}e_{1\pi^{-1}(i)}}\}, \{F_{2i} = g^{df_{2\pi^{-1}(i)}e_{2\pi^{-1}(i)}}\}$
  - $\mathcal{V}$ challenges with a single random value c
  - Both $\mathcal{P}$ and $\mathcal{V}$ compute the sequences U_i, V_i respectively as:
    * $\mathcal{P} : U_i = g^{f_{1i}e_{1i} + cf_{2i}e_{2i}}, \quad \mathcal{V} : U_i = E_{1i}^{f_{1i}} E_{2i}^{cf_{2i}}$
    * $\mathcal{P} : V_i = g^{df_{1\pi^{-1}(i)}e_{2\pi^{-1}(i)} + cdf_{2\pi^{-1}(i)}e_{2\pi^{-1}(i)}}, \quad \mathcal{V} : V_i = F_{1i}F_{2i}^{c}$
    * They execute the *simple n-shuffle proof* for $(U, V, g, D = g^d)$
  - If the proof succeeds the $\mathcal{V}$ is convinced that $\mathcal{P}$ knows the values of $u_i = logU_i, v_i = logV_i, d$ and the permutation π.

- Prove knowledge of the d used in U_i, V_i, X_i, Y_i
- This can be done using n executions of the Schnorr Protocol for knowledge of d, along with a Chaum Pedersen proof.

Groth's Application To Homomorphic Schemes In [65], Jens Groth applied the main idea of [98] to the mixing of homomorphic ciphertexts, obtaining efficiency improvements. In the particular setting, the input and output ciphertexts under π satisfy: $\{E'_i = E_{\pi^{-1}(i)}E(0, r_i)\}_{i=1}^{n}$.

The homomorphic property implies that if the mix server is honest, then the product of the outputs is a reencryption of the product of the inputs:

$\prod_{i=1}^{n} E'_i = \prod_{i=1}^{n} E_{\pi^{-1}(i)}E(0, r_i) = \prod_{i=1}^{n} E(m_{\pi^{-1}(i)}, r'_i), E(0, r_i) = E(\sum_{i=1}^{n} m_i, \sum_{i=1}^{n} r_i + r'_i) = \prod_{i=1}^{n} E_i E(0, r_i)$

This provides a way to check correctness. As a result the scheme works as follows:

- $\mathcal{P}$ and $\mathcal{V}$ exchange knowledge of a permutation π without revealing it.
  - To this end $\mathcal{P}$ commits to π by selecting n random elements $\{s_i\}$, applying π to them to create $\{s_{\pi(i)}\}$, and committing to the result $c_{si} = commit(s_{\pi(i)})$. For efficiency a homomorphic multicommitment scheme can be used, where k items are committed to simultaneously
  - $\mathcal{V}$ challenges with n items of its own $\{t_i\}$
  - $\mathcal{P}$ applies pi and returns them to $\mathcal{V}$ as commitments $c_{ti} = commit(s_{\pi(i)})$
  - $\mathcal{V}$ sends a single value λ
  - $\mathcal{P}$ calculates $\{s_i + \lambda t_i\}$ and proves that a shuffle of it is contained in $\{c_{si}c_{ti}^{\lambda}\}$, by using the idea of Neff's simple n-shuffle proof.
- $\mathcal{V}$ selects challenges which will be used to blind the inputs and the permuted outputs
- $\mathcal{P}$ proves the equality of the products of the blinded inputs and output

4.3.5 Sender Verifiability

As we described in 4.2.2, a decryption only mixnet cannot be achieved using the underlying El Gamal cryptosystem, since the first part of the ciphertext would remain unchanged. In [138] Douglas Wikström overcomes this difficulty by proposing a two component key, so as to enable modifying both parts of each ciphertext. In more detail:

- The secret key of each mix server is constructed from a pair of values generated by each of the participants $(w_j, x_j) \in \mathbb{Z}_{q^2}$, while the public keys are (g^{w_j}, g^{x_j})
- The public key is computed iteratively from the last to the first mix server:
  - The last mix server selects $(g, 1)$ and calculates $(z_k, y_k) = (g^{w_k}, g^{x_k})$
  - Each previous mix server computes $(z_j, y_j) = (z_{j+1}^{w_j}, y_{j+1}z_{j+1}^{x_j})$ that is, raises the first component of the public key created so far with the first part of his own share, and multiplies the second component of the public key so far with the first raised to the second component of his share.
  - As a result the first mix server has computed the key of the mixnet as

$$(Z_1, Y_1) = (g^{\prod_{t=1}^{k} w_t}, g^{\sum_{t=1}^{k}(x_t \prod_{s=j+1}^{k} w_s)})$$

  - Each intermediate key calculated with this process can be made public.

- Each voter i encrypts using the classical El Gamal process, by selecting a random value r_i and computing $(Z_1^{r_i}, v_i Y_1^{r_i})$
- Each mix server M_j receives as input a list of tuples and decrypts each item in it, by raising the first component to $\frac{1}{w_j}$ and the multiplying the second component to $\frac{-x_j}{w_j}$ and following the normal El Gamal decryption. More specifically for a single item $(a_{j-1}, b_{j-1}) = (Z_j^r, vY_j^r)$ we get $a_j = a_{j-1}^{\frac{1}{w_j}} = g^{r \prod_{t=j+1}^{k} w_t} = Z_j^r$ and $b_j = b_{j-1} a_{j-1}^{\frac{-x_j}{w_j}} = mY_j^r$
- After the decryption the mix server sorts the partially decrypted items, in effect applying the permutation

This construction has a desirable side effect, that is called *sender verifiability.* The voter can compute the partially decrypted item by retrieving (Z_j, Y_j) apply the encryption producing (Z_j^r, vY_j^r) and search for it in the input list.

4.3.6 Performance Analysis

In order to compare the above schemes, we try to calculate their efficiency in terms of computation and communication cost. The computation cost is is measured in the total number of exponentiations for reencryption, proof and decryption. The authors however use precomputed values, and we could not exactly compare their schemes. As a result, we refer to [22] as a source for the total number of exponentiations of n items by k servers. The performance of the mixnets described above is:

- [84] $2n + 642nk + (2 + 4k)n$
- [91] $2n + 7nlogn(2k - 1) + (2 + 4k)n$
- [59] $2n + 18n(2k - 1) + (2 + 4k)n$
- [98] $2n + 8n(2k - 1) + (2 + 4k)n$

According to [22] the Neff mixnet is the fastest one, while according to [65] the Furukawa-Sako is the fastest.Everybody agrees though that verification via such proofs is computationally expensive. As a result the next step in the mix net research was to construct verifiable mix nets without using zero knowledge proofs.

4.4 Almost Verifiable Mixnets

The protocols in this section provide a performance enhancement over verifiable mix nets by sacrificing either privacy or correctness. This, in simple terms, means that the mix net will unveil the correlation between some of the inputs and the output or provide a less 'convincing' proof of correctness or both. The specifics of each protocol aim to minimise the probability the association between a voter and vote is discovered and that a mix server cheated, by injecting fake votes or changing a subset undetected or aided in privacy breaking. However, as we shall see, this is not always the case. Various attacks have been demonstrated that utilise the tagging concept first presented by Pfitzman and described earlier the alleged security of most of these constructs.

4.4.1 Almost Entirely Correct Mixing

The first such case to describe here was proposed in [22]. It is based on the idea, that we might not need a 100% correct proof of mixing, especially if the margin of victory is wide. An almost correct proof of mixing might do. This approach yields a huge speed up over previous techniques. For instance and for 10^6 votes it provides instantly a 99% correct proof of mixing. The method employed consists of the following steps:

- Verifier selects a random subset S of mix server inputs
- Calculates the product π_s
- Reveals S to the mix server.
- Ask to produce a set of outputs S' st. $\pi_s = \pi_{s'}$
- This challenge is easy for an honest mix server, as it can simply apply the permutation
- But for a cheating mix server it might not be impossible to find such S'.

More specifically:

- Let the input ciphertexts be El Gamal encryptions $C_i = (g^{r_i}, m_i \cdot y^{r_i})$
- Let the output ciphertexts be $C_{\pi(i)} = (g^{r'_i}, m'_i \cdot y^{r'_i})$
- Each server M_j generates and commits to random r_j.
- All mix servers calculate $r = \oplus_j r_j$ and agree on a security parameter α.
- Each Mix-Server M_j proves that $\prod_{i=1}^{n} m_i = \prod_{i=1}^{n} m'_i$ using 2.1.7.2
- Generate α sets $S_1, \cdots, S_\alpha, S_i \subset S$ by including each input item with probability $\frac{1}{2}$ where randomness stems from r.
- The sets $S_1, \cdots, S_\alpha$ are given to M_j.
- M_j must produce $S'_1, \cdots, S'_\alpha$ st. $\forall i \in \{1, \cdots \alpha\} \|S_i\| = \|S'_i\|$ and $\prod_{k \in S_i} m_i = \prod_{k \in S'_i} m'_i$. Proof is given using using 2.1.7.2
- If a server fails in the previous test it is considered dishonest and is excluded and the protocol is restarted.
- If the protocol succeeds then the ballots are decrypted.

Now for k mix servers and n inputs the following performance evaluation apply:

- Proof Cost = $2\alpha(2k - 1)$ exponentiations
- Decryption Cost = $(2 + 4k)n$
- Privacy Every input is hidden among $n/2^\alpha$.
- Cheating will be detected with probability $1 - \frac{5}{8}^\alpha$ or the DLOG problem can be solved in polynomial time.

The authors in [22] calculated that in an election with 160K voters and $\alpha = 6$ subsets to check per server every vote is hidden in 2500 votes and cheating will be detected with probability 94%.

4.4.2 Optimistic Mixing

A best of both worlds approach was given in [64] and is called Optimistic or Exit - Poll Mixing. It assumes an El Gamal re-encryption mixnet. It produces a fast proof when all mix servers are honest. If a cheating mix server is found the no output is produced, and all the relevant data are forwarded to a verifiable mixnet, in a way that privacy is not compromised. As it is designed to execute several mix sessions with the same set of keys all the proofs of knowledge are bound to a session id.

The technique in this approach borrows form [22] as it uses a proof of product again, albeit with a checksum. The mix net is given input the encryption of plaintexts with cryptographic checksum. In order to prove correct operation the mix net the relationship $\prod_{input} plaintexts = \prod_{output} plaintexts$ must be satisfied without any knowledge of the plaintexts (for privacy reasons). However product preservation does not imply absence of cheating, and for this reason the check sum is employed. If cheating occurs some plaintexts will have been replaced so that the product is not altered, but the checksums will still be invalid. An invalid check sum however does not only point to a cheating mix server but to a cheating voter as well (the checksum was invalid from the onset). The protocol requires extra investigation to discriminate between the two cases.

In order to apply the checksum solution, the input plaintexts v_i (*and* the output plaintexts v_i') are restricted to a particular format so that it is infeasible $\{v_i\} \neq \{v_i'\}$ and $\prod v_i = \prod v_i'$. The checksums are computed by applying a hash function to the input. A triple $(E(m, r), h(m), E(h(m), r'))$ is created in this form and this triplet is given to the mix net. The authors of [64] cite a theorem by Wagner that states that it is impossible to find two different lists of ciphertexts of equal length $\{(u_i, v_i)\}_{i=1}^{N} \neq \{(u_i', v_i')\}_{i=1}^{N}$ such that $\prod_{i=1}^{N} h(u_i, v_i) = \prod_{i=1}^{N} h(u_i', v_i')$ in a group where the discrete logarithm is hard.

Because there will be two mixing sessions extra care must be given to privacy. As cheating will be detected **after the fact** not much can be done if privacy has been compromised or the votes have been decrypted. For example:

- User i submits $t_i = (\, E(m_i, r_i), E(h(m_i), r_i')\,)$
- Cheating mix server replaces u_1 with $t_1 \cdot t_2$ and t_2 with $(1, 1, 1, 1)$.
- Cheating will be discovered after the decryption
- The cheating server will be disqualified but
- If mixing is restarted the cheating server will be able to distinguish the output of the first two users from the outputs of the rest by comparing the output of the second run with the output of the first run by checking for the product of the two plaintexts

In order to avoid this situation double enveloping is proposed:

- User input is encrypted twice
- Each user submits the triple: $t_i = (\, E(G_i, r_i), E(M_i, r_i'), E(h(M, G), r_i'')\,)$ where $(G_i, M_i) = (g^{r_i}, m \cdot y^{r_i}) = E(m_i, r_i)$
- Verification Step: Decrypt Once
- If verification succeeds (both product and checksum) then decrypt twice and tally
- If cheating is discovered then encrypted inputs the cause of cheating is sought:
  - Cheating senders: Solution: Ignore them, and proceed with the rest

  - Cheating mix servers: The messages will be forwarded to the slower mixnet with the mix server replaced

- Cheating will not expose privacy, since the cheating server will cheat on (single) ciphertexts

However extra investigation must be give for the invalid triples, the ones where $w_i \neq h(u_i, v_i)$. Their path through the mix netback is back-traced by the following process:

- Last server reveals: (input, randomisation)
- Validate that the triple can be obtained from the input
- Repeat until the first server
- If the first server is reached then classify as user cheating
- If a server fails validation the classify as server cheating

4.4.2.1 Attacks

In [137] a variety of attacks are described against optimistic mixing. The main idea is based on the attack of Pfitzmann [111], adapted to the double enveloping setting. Moreover, some weaknesses that apply to the particular protocol are used. In this section we detail these attacks:

Break privacy when all servers are honest This attack exploits the fact, that the protocol employs proof of ciphertext knowledge and not of message and that it uses the same keys for both layers of encryption. An attacker must be able to send 2 messages to the mixnet (e.g. by corrupting two voters) in order to mount this attack. The attacker targets user Alice, who submits message m to mixnet. The message is 'double enveloped': $m \rightarrow_{E_1}\rightarrow (\mu_1, \mu_2) \rightarrow_H\rightarrow (\mu_1, \mu_2, w) \rightarrow_{E_2}$. The attacker randomly select γ, δ and calculates $\mu_1^\delta, \mu_2^\delta, w_\delta = H(\mu_1^\delta, \mu_2^\delta)$ and $\mu_1^\gamma, \mu_2^\gamma, w_\gamma = H(\mu_1^\gamma, \mu_2^\gamma)$ and submits $E(\mu_1^\delta), E(\mu_2^\delta), E(w_\delta)$ and $E(\mu_1^\gamma), E(\mu_2^\gamma), E(w_\gamma)$, by corrupting two voters. The mixnet operates as usual and since all mix servers are honest removes both layers of encryption and produces output L_k. Then the attacker raises outputs to $\frac{\delta}{\gamma}$ producing output $L_k^{'}$. This is more clear in the following sketch:

$$
output \rightarrow_{D_1} \begin{bmatrix} (u_1,v_1,w_1) \\ \cdots \\ \mu_1^\delta,\mu_2^\delta,w_\delta \\ \cdots \\ \mu_1,\mu_2,w \\ \cdots \\ \mu_1^\gamma,\mu_2^\gamma,w_\gamma \\ \cdots \\ (u_N,v_N,w_N) \end{bmatrix} \rightarrow_{D_2}\text{to produce} L_k \begin{bmatrix} m_1 \\ \cdots \\ m_x = m^\delta \\ \cdots \\ m \\ \cdots \\ m_y = m^\gamma \\ \cdots \\ m_N \end{bmatrix} \rightarrow \text{raise to } \tfrac{\delta}{\gamma} \text{to produce } L_k' \begin{bmatrix} m_1^{\frac{\delta}{\gamma}} \\ \cdots \\ m_x^{\frac{\delta}{\gamma}} \\ \cdots \\ m^{\frac{\delta}{\gamma}} \\ \cdots \\ m_y^{\frac{\delta}{\gamma}} = m_x \\ \cdots \\ m_N^{\frac{\delta}{\gamma}} \end{bmatrix}
$$

FIGURE 4.4: Pfitzmann attack in optimistic mixing

Between L_k and $L_k^{'}$ there will be 2 identical outputs $m_x = m^\delta$. Now the attacker can calculate $m_y^{\frac{1}{\gamma}} = m$ in L_k, where m_y is indicated by the duplicate in $L_k^{'}$. As a result the message m will be revealed. The attacker will search the output list for duplicates and find index l of message m. Since the attacker would like to correlate the vote with the voter, she will search the initial encryption for ciphertext corresponding to l. As a result alice is revealed. Of course this attack can be generalised to break the privacy of s senders, by corrupting $2s$ voters.

Different Keys and Corrupt Mix Server Now, if we apply some countermeasures and strengthen the protocol by requiring that double enveloping uses different keys, we can still exploit the fact that the voters prove knowledge of ciphertext and not of message. This time we would require the aid of one corrupt mix server to mount the attack. Let's denote the initial (inner) encryption as E_{in} and the outer encryption (where we encrypt each triple) E_{out}. In [137] it is assumed that the protocol allows 2 mix sessions with the same keys, which is unclear in [64]. The corrupted mix server is the first and this is identified by the mix net after the first mix session. The attack works as follows: During the first mix session that attacker targets user Alice with message m and user Bob with message m'. The corrupted mix server swaps encryption of hashes $m \rightarrow E_{in} \rightarrow (u,v) \rightarrow H \rightarrow w \rightarrow E_{out} \rightarrow [E_{out}(u), E_{out}(v), E_{out}(w')] = \alpha$ and $m' \rightarrow E_{in} \rightarrow (u',v') \rightarrow H \rightarrow w' \rightarrow E_{out} \rightarrow [E_{out}(u'), E_{out}(v'), E_{out}(w)] = \alpha'$. The protocol proceeds as usual. By the swapping the output contains two invalid tuples (u,v,w') and (u',v',w). Back-Tracing reveals first mix server as cheater, but the attacker has linked Alice to (u,v) and Bob to (u',v'). The second mix session will be used as a decryption oracle, by a relation attack on Alice's ciphertext (u,v), similar to the one described in the previous paragraph. The attacker will randomly select γ, δ and calculate $u^{\delta}, v^{\delta}, w_{\delta} = H(u^{\delta}, v^{\delta})$ and $u^{\gamma}, v^{\gamma}, w_{\gamma} = H(u^{\gamma}, v^{\gamma})$. Then he calculates and submits $E_{out}(u^{\delta}), E_{out}(v^{\delta}), E_{out}(w_{\delta})$ and $E_{out}(u^{\gamma}), E_{out}(v^{\gamma}), E_{out}(w_{\gamma})$. The mix net operates as usual and since all mix servers are honest (now) removes both layers of encryption. The attacker raises outputs to $\frac{\delta}{\gamma}$ and correlates messages as before.

Attack On Privacy without corrupted users A different attack in [137] assumes no corrupted voters (all senders are honest) and a single corrupted mix server, the last one M_k. This mix server is extremely powerful, which results in breaking the privacy of all users. The actions of M_k are:

1. Generate the tuple $(a', b', \cdots, f') = (\frac{a_{k-1}}{a_0}, \frac{b_{k-1}}{b_0}, \cdots, \frac{f_{k-1}}{f_0})$ where $a_j = \prod_{i=1}^{N} a_{ji}$ the product of the first components of the N inputs of M_j
2. Generate the tuple $\alpha_1 = (a' \cdot a_{01}, \cdots, f' \cdot f_{01})$
3. Generate the list L'_{k-1} by retrieving L_0, the input list to the mixnet, and replacing the first message with α_1
4. Instead of processing the normal input list L_{k-1} process the list L'_{k-1}.
5. Mixnet output is a permutation and reencryption of L'_{k-1}, essentially of L_0

If M_k doesn't get caught then after decryption, privacy is broken. It turns out that this is the case and the corrupted server passes verification. It can prove correct reencryption:

- The modification to the input list is *invisible* $a'_{k-1} = a' a_{01} \cdot \prod_{i=2}^{N} a_{0i} = a' \cdot \prod_{i=1}^{N} a_{0i} = a' \cdot a_0 = \frac{a_{k-1}}{a_0} \cdot a_0 = a_{k-1}$
- Subsequently M_k behaves honestly

During a potential investigation of invalid triples M_k cannot correlate the real L_{k-1} with L_k but it is not necessary as:

- All users are honest
- All mix servers except M_k are honest
- M_k does not corrupt inner triples

As a result, there are no invalid triples, M_k doesn't get caught and the privacy of all users is breached.

Attack with two corrupted mix servers Using number theory and 2 corrupted mix servers (M_1,M_k) the privacy of a particular user can be broken, assuming all senders are honest.

Let P, Q the primes in the initialisation of the El Gamal Cryptosystem. All ciphertexts should be in G_Q. If no check is made, we can use elements in $\mathbb{Z}_P^* - G_Q$ to 'tag' messages and break privacy. More specifically:

- Corrupt mix server M_1 and select $\gamma \in_R \mathbb{Z}_P^* - G_Q$
- Compute $\alpha_1' = (\gamma \cdot a_{01}, \cdots, f_{01})$ where α_1 is the message of Alice (target) and $\alpha_2' = (\gamma^{-1} \cdot a_{02}, \cdots, f_{02})$ where α_2 is the message of another user Bob.
- Replace α_1 with α_1' and α_2 with α_2' and proceed as usual
- Corrupt mix server M_{k-1} receives input L_{k-1} and processes it as usual
- Output would be $L_k = \{\beta_i\}_{i=1}^N = \{(\cdot a_{ki}, \cdots, f_{ki})\}_{i=1}^N$
- Search for l, l' st: $a_{kl}^Q = \gamma^Q$, $a_{kl'}^Q = \gamma^{-Q}$
- Replace β_l with $(\gamma^{-1} a_{kl}, \cdots, f_{kl})$ and $\beta_{l'}$ with $(\gamma_{kl'}^a, \cdots, f_{kl'})$
- Output the result. Decryption takes place. Alice's message is m_l

This attack works because: $\forall b \in G_Q : b^Q = 1$. As a result $a_{kl}^Q = \gamma^Q (g^r a_{01})^Q = \gamma^Q$. The attack passes validations since only the a-component of the tuple is changed and the products are left unchanged: $a_1 = \gamma a_{01} \gamma^{-1} a_{02} \prod_{i=3}^N a_{0i} = \prod_{i=1}^N a_{0i} = a_1$ and $a_k = \gamma a_{kl'} \gamma^{-1} a_{kl} \prod_{i \neq l,l'}^N a_{0i} = \prod_{i=1}^N a_{ki} = a_k$

Delayed Effect Attack In a different kind of attack, the adversary can cancel out the cheating based on an event after the vote casting phase. The first mix server must be corrupted as must be the case for an even number of senders. This scenario is applicable in situations such as double elections. The adversary might inject votes in the second election, but depending on the first outcome he might cancel them out.In more details let $x_i = (a_i, b_i, c_i, d_i, e_i, f_i) = (E(u_i), E(v_i), E(w_i)), i = 1, \cdots, N$ be the casted votes. The attacker can select $z \in G_Q$ and two voters k, l and compute $x_k' = (a_i, b_i, c_i, zd_i, e_i, f_i)$ and $x_l' = (a_i, b_i, c_i, z^{-1} d_i, e_i, f_i)$. Since the product is maintained an attacker might replace x_k, x_l and M_1 Might choose to correct them or not

4.4.3 Randomised Partial Checking

The most widely used mix net of this category was proposed in [74] and it is named after the technique of random partial checking that it uses to check on the correct operation of the mix net. Henceforth we shall refer to the mix net proposed in [74] as RPC. More specifically, the RPC gives up the expensive notion of proof and provides strong evidence that the mix server has operated correctly, that is the output is indeed a permutation of the input. Strong evidence here means that the verification here is probabilistic. The idea here is simple: The mix server are 'asked' to partially reveal the input output correspondence. For n items, $\frac{n}{2}$ are randomly chosen and the input/output relation is revealed. Of course, the mix server has no control of which items are revealed. The idea is that a cheater cannot get away with altering too many votes. Of course, there is a privacy trade off, as a mix server is not completely anonymizing the votes. This is circumvented, as each mix server reveals different portions of its input and output. As a result, no vote can be traced end to end, and privacy is attained albeit at the mix net as a whole. It must be noted, that one of the most important benefits of the system is that it is crypto system agnostic, as it can be used with any public key crypto system. More specifically, after system setup, ballot B_i preparation and encryption $C_{i,0} = E_{PK}(B_i)$, ballot validity checking takes place. Then each mix server commits to a permutation by the following scheme:

- A commitment to integer i using witness w, $\zeta_w[i]$ is selected. For speed: $\zeta_w[i] = h(w||i)$ where h is a hash function.
- The actual commitment of M_j to the permutation is denoted γ_{ij}.
- There are two options:
  - Commit to the mappings of input elements to output elements $\Gamma^{IN} = \{\zeta_{w_{ij}}[\pi_j(i)]\}_{i=1}^{n}$
  - Commit to the mappings of output elements to input elements $\Gamma^{OUT} = \{\zeta_{w_{ij}}[\pi_j^{-1}(i)]\}_{i=1}^{n}$

After permutation commitment the mix net processing takes place: $C_{ij} = X_j(C_{ij-1})$, that is the mix server reencrypts or partially decrypts its input items or both. Then the committed permutation is applied. The complete operation can be summarised as:

$$C_{ij} = X_j(C_{\pi_j^{-1}(i),j-1})$$

Then verification proceeds. M_j reveals a fraction $p > 0$ of the input - output correspondence:

- Let $\pi_j(k) = i$ and $C_{ij} = X_j(C_{kj-1})$
- Reveal (k, i, R_{ijk}) where R_{ijk} is the necessary information to reconstruct the processing (padding,randomness etc.)
- Reveal the in-commitment γ_k or the out-commitment γ_i.

The specifics of what to reveal are:

- Each M_j commits to random value r_j
- Let $R = \oplus_j r_j$
- Let $Q = h(R, BB)$ where BB are the contents of the bulletin board
- $Q_j = h(Q, j)$ determines which values must be revealed.
- $P_{IN}(Q_j, k) = true \Rightarrow$ reveal the pair with k as input
- $P_{OUT}(Q_j, i) = true \Rightarrow$ reveal the pair with i as output

Since revelations take place extra care must given to preserve privacy. To this end RPC, pairs the servers so that we never reveal the same pair twice. The requirement for one honest mix server, now turns into a requirement for one honest pair of mix servers. Let j odd and (M_j, M_{j+1}) a server pair . Then:

- $P_{IN}(Q_j, k) = false$
- $P_{OUT}(Q_j, i) = true$ with probability $\frac{1}{2}$
- $P_{IN}(Q_{j+1}, i) = \bar{P_{OUT}}(Q_j, i)$
- $P_{IN}(Q_{j+1}, m) = false$

For the security analysis, let k minimum number of votes to alter election result. The probability Pr_{und} that someone switches k votes without detection, ff $p = \frac{1}{2}n$ permutations are revealed is $Pr_{und} = \frac{1}{2^k}$. This can be explained as testing is independent and since mix servers are paired, we check only predecessors or successors of intermediate items. If a mix server is not honest, we will find it out by checking predecessor or successor.

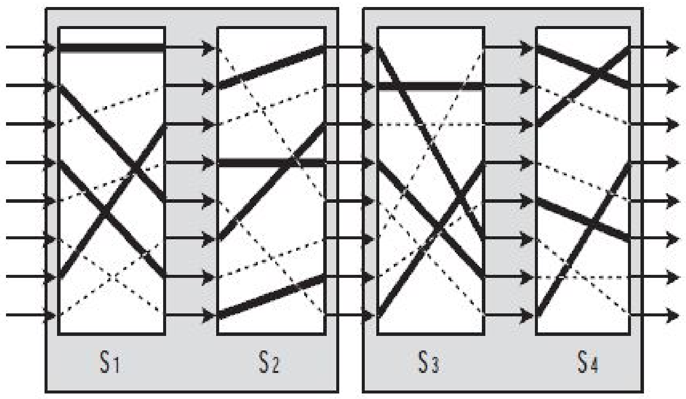


FIGURE 4.5: Mixnet with Randomised Partial Checking [74]

4.4.3.1 Attacks

The RPC protocol withstood 8 years of cryptanalysis and was used in many commercial products such as Prêt à Voter and in many real elections. Fairly recently, however some serious attacks were reported [80], both on El Gamal mix nets and Chaumian mix nets.

Tagging attacks First of all, one must note that the Pfitzmann attacks work with probability $\frac{1}{2}$. If the first mix server is corrupted along with one voter then a single message can be traced. The submitted ciphertext is c. The corrupted mix server chooses an integer δ and replaces an output by c^{δ}. This is not detected during verification with probability $\frac{1}{2}$. The output can be searched for messages m, m^* st $m^* = m^{\delta}$. If such a message is found then c was encryption of m. A variation of this attack, allows the tracing of multiple messages:

- Submitted ciphertexts are $\{c_i\}_{i=1}^{s}$
- Corrupted mix server chooses integers$\{\delta_i\}_{i=1}^{s}$
- Replace an output by $\prod_{i=1}^{s} c_i^{\delta_i}$
- This is not detected during verification with probability $\frac{1}{2}$
- Search the output for messages $\{m_i\}_{i=1}^{s}, m^*$ st $m^* = \prod_{i=1}^{s} m_i^{\delta_i}$
- Correspondence for m_i is revealed

Duplicates are dangerous A chaumian RPC mix net that does not check for duplicates can be successfully attacked with 2 corrupt mix server (M_1, M_k) and $\frac{s(s+1)}{2}$ corrupted voters. The first mix server targets ciphertexts $\{c_i\}_{i=1}^{s}$. According to the protocol it removes the first layer of encryption. The corruption takes place as for c_i it makes i independent encryptions using his public key totalling $\frac{s(s+1)}{2}$ ciphertexts replaced with duplicates. The last mix server is given input that contains $i+1$ duplicate for ciphetext i. It decrypt using public key as always. But it can identify correspondence based on the number of duplicates. While $s + \frac{s(s+1)}{2} \leq N$, the privacy of $O(\sqrt{N})$ voters can be broken. Of course, a simple countermeasure to this attack is to remove duplicates everywhere.

Commitments must be valid permutations The most serious attack is based on the fact that nowhere in [74] the commitments are validated to be correct permutations. This omission was reflected in many implementations. The specifics assume a reencryption mix net, 2 corrupted mix servers that due to the nature of RPC, appear as one and are operated by the same entity and 2 corrupted voters. The result is impressive as privacy can be broken without detection.

During initialisation the submitted ciphertexts are $\{c_i\}_{i=1}^{s}$ and the attacker chooses integers$\{\delta_i\}_{i=1}^{s}$ calculating $c = \prod_{i=1}^{s} c_i^{\delta_i}$. The corrupted voters post 2 messages that are reencryption of each other $c_{i_1,0}, c_{i_2,0}$.

M_1 is passive and only keeps track of corrupted messages permutations and reencryption randomness. M_2 commits c_{i_1}, c_{i_2} to same output $\pi_2(i_1)$ and replaces output $\pi_2(i_2)$ with c as depicted in the figure below:

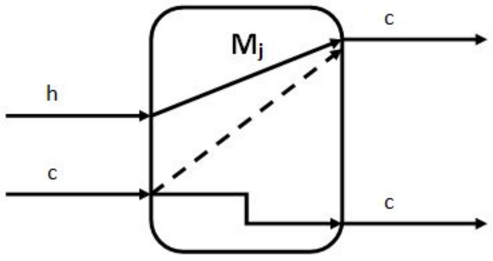


FIGURE 4.6: Mix Server with invalid permutation

After mixing the attacker can search the output for messages $\{m_i\}_{i=1}^{s}, m^*$ st $m^* = \prod_{i=1}^{s} m_i^{\delta_i}$. The correspondence for m_i is revealed without detection, as both commitments verify since c_{i_1}, c_{i_2} are reencryptions of the same message and M_2 can provide the randomness to prove it.

More generally, if the attacker can provide $r + 1$ reencryptions of a message and $r + 1$ false commitments then she can track $r \cdot s$ messages.

In a extreme scenario this allows to rig an election undetected:

- Corrupt the first mix server and the first sender
- First sender sends m as plaintext $c_{10} = Enc(m)$
- Replace all the ciphertexts with c_{10}
- Corrupted mix server commits all outputs to 1
- Corrupted Mix Server can pass verification

Of course, it would be more realistic to replace the actual outputs with the distribution of votes that she likes.

The above apply for unopened commitments as well. If unopened commitments are not be consistent with a permutation, an attacker might replace a message sent by an honest sender with a cheating message and commit to the same output. The detection probability is equal to the probability of opening both commitments namely $\frac{1}{4}$. For t messages the success probability equals $(\frac{3}{4})^t$.

4.4.4 The Puigally Mix Net

A more recent implementation, [10] was proposed by Puiggali and Castello in 2010 and was part of the Norwegian 2011 municipal elections. It combines the principles of Randomised Partial Checking,

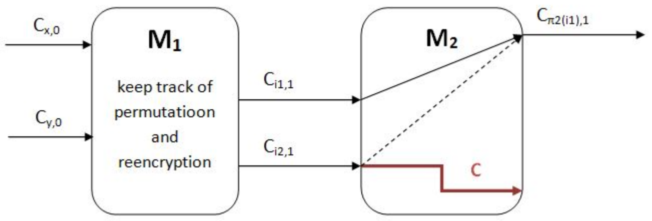


FIGURE 4.7: Mix server replaces unopened commitments

Almost Entirely Correct Mixing and Optimised Mixing into a reencryption based mixnet. Verification occurs after operation and before decryption. Inputs and outputs are split into groups with constant group size l and group membership is revealed for all votes, by comparing $\prod inputs$ with $\prod outputs$ for each group in zero-knowledge. More specifically the verifier groups encrypted input. For each output the prover reveals the input group and then calculates:

- **Input Integrity Proof:** Products per group of input votes
- **Output Integrity Proof:** Products per input group of output proofs

Afterwards it provides a Chaum-Pedersen Proof that the output integrity proof is a reencryption of the input integrity proof. The first grouping is random and for each M_j Regrouping takes place:

- Sort outputs of M_{j-1} by group index
- Input group #1 Receive *first* item of each output group
- Input group #2 Receive *second* item of each output group ...

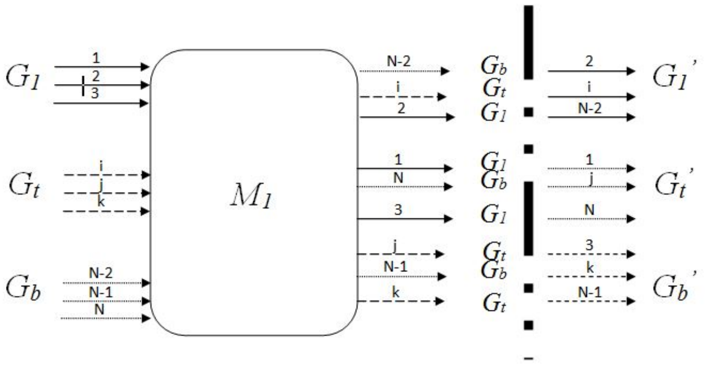


FIGURE 4.8: Puigally Mix Net

The randomness of the first grouping can be created using an application of the Fiat Shamir heuristic [49], without any need for an external verifier. The first mix server concatenates the ciphertexts that it must process, and supplies them to a hash function. The output is then used as a seed in a random number generator, whose output is split into $log_2 l$ sized groups which indicate group membership for each input ciphertext. Subsequent groups can be formed using the rule described above.

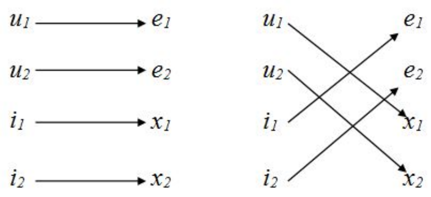
4.4.4.1 Attacks

The pattern of attacks must be evident by now. Each almost verifiable mix net, accept some form of the tagging attack, that succeeds with some probability or that uses more messages to trace its target. The same apply for the Puigally mix net [79]. Before we begin the description of the attacks, let's remind ourselves of the *birthday paradox*: If we want to pick s elements from a multiset with b elements the collision probability can be bounded: $P(s,b) \geq 1 - e^{\frac{-s^2}{2\cdot b}}$. Now lets formulate the basic Pfitzmann Attack:

The attacker is after the privacy of s voters with inputs $c_1, ..., c_s$. He corrupts 2 voters with inputs c_{01}, c_{02} and the first mix server/ Then he select s random exponents $\delta_1, ..., \delta_s$ and calculate $u_1 = \prod_{i=1}^{s} c_i^{\delta_i}$ and $u_2 = \frac{c_{01}\cdot c_{02}}{u_1}$ to replace c_{01}, c_{02} respectively. Note that the product is preserved: $c_{01} \cdot c_{02} = u_1 \cdot u_2$. In order to pass verifcation u_1, u_2 must belong to the same block, which happens with probability: $\frac{1}{b}$. If this is the case the original message might be recovered after decryption, by searching $s+1$ messages to find the ones with $m = \prod_{i=1}^{s} m_i^{\delta_i}$.

A variation of this attack can be made undetected: Again we target privacy of s voters and we have at our disposal a single corrupted mix server and B corrupted voters that post votes that are reencryptions of each other for the corrupted mix server can handle them without affecting the verification process We can get a very high probability of success with few corrupted voters. For instance with $B = 3\cdot\sqrt{b}$ corrupted voters we can get 98% chance of success, by the following reasoning:

- Build u_1, u_2 'malicious messages', where $u_1 = \prod_{i=1}^{s} c_i^{\delta_i}$ and $u_2 = \frac{c_{01}\cdot c_{02}}{u_1}$
- u_1, u_2 must end up in the same input block
- Say that u_1, u_2 exit the mix server at positions e_1, e_2
- Let i_1, i_2 two corrupted messages that are indeed in the same input group
- Say that i_1, i_2 exit at positions x_1, x_2
- The corrupted mix server can swap *u's* with the *i's* to put them in the same block



Swapping can go undetected with very hight probability due to the birthday paradox, as the corrupted mix server must prove that $x_1 \cdot x_2 = ReEnc(i_1 \cdot i_2)$. But $ReEnc(i_1 \cdot i_2) = ReEnc(c_{01} \cdot c_{02}) = ReEnc(u_1 \cdot u_2)$. Of course by hushing up the replacement the other 'processing' can be proved as well $e_1 = ReEnc(u_1) = ReEnc(i_1)$ and $e_2 = ReEnc(u_2) = ReEnc(i_2)$. Then *Probability of success = Probability 2 corrupted messages end up in the same block*

Attack On Correctness Using 1 corrupted mix server and if $l \geq b$ it can replace $R = \frac{1}{3}\sqrt{b} - 1$ votes without detection. The corrupted mix server M_j replaces thefirst R+1 votes:

- The first corrupted votes $c_{j-1,1} \cdots c_{j-1,R}$ get replaced by 'malicious' messages $u_1, \cdots u_R$
- In order to maintain the product the vote $c_{j-1,R+1}$ gets replaced by $\frac{\prod_{i=1}^{R+1} c_{j-1,i}}{\prod_{i=1}^{R} u_j}$

The replaced list L'_{j-1} gets reencrypted and shuffled. The probability that 2 integers in $\{1, R+1\}$are in the same block is 0.05 With very high probability all such integers are in different blocks. By way of output partitioning all end up in the same output, with very high probability and the proof is valid both for L_{j-1} and L'_{j-1}.

4.5 Other Approaches

4.5.1 Online-Offline Mixing

So far, we have built our description of mixnets on the premise that they operate *online*, that is, during the election. As a result the mixing and verification should happen with reasonable duration, which justifies all the research we presented for more efficient proofs of shuffling. However, if we lift this assumption and allow part of the mixnet operation to happen before the election, aka *offline*, interesting side effects might come about. Firstly the performance requirements can be relaxed as far the offline phase is concerned, since it typically lasts for days and it is not as heated as the actual elections, because of the need for immediate results. As a consequence, the complex operations can be moved there, leaving an efficient online phase. In addition if the online functions are public, the scheme gets instant verifiability. This approach has its roots on the simple precomputation of values that aid in the encryption, decryption and the various proofs used in mixnets, but has evolved in the preexecution of entire operations.

This idea first appeared in [7] as *mixnet obfuscation*. Obfuscation is a term originating from software engineering, and refers to a functionality-preserving transformation of a program's source code, that yields no more information about it than black-box access. The actual shuffling takes place online and in public. However the process follows a specification which can be obfuscated and proved offline, before the cipherexts are even given. In order to be applicable a homomorphic cryptosystem should be used. The insight of this method is that shuffling can be represented as multiplication between the ciphertexts and a permutation matrix like the one we saw earlier in 4.3.3. This idea has the following implications:

- Shuffling becomes *deterministic* as each mix server is not selecting a random permutation each time it operates, but uses the permutation matrix that is fixed
- In order to implement homomorphic matrix multiplication, a cryptosystem is required that is both additive and multiplicative homomorphic
- The permutation matrix should be generated in a distributed fashion, as its knowledge breaks anonymity
- The security of the system is characterised by the inability of an attacker $\mathcal{A}$ to distinguish between two permutations resulting from the application of the mixnet

In more detail the scheme operates as follows:

- **Distributed generation of permutation matrix**
  - An identity matrix is encrypted using deterministic encryption (aka randomisation 0)
  - The stakeholders form a (traditional) mixnet, which receives as input the rows of the identity matrix. Each mix server permutes and reencrypts the entries in the permutation matrix
  - In the end, an encrypted permutation matrix is generated. A single honest shareholder guarantees the security of the scheme
- **Proof of correctness**
  - It must be proved that the output of the mixnet is the encryption EP of some permutation matrix P
  - To this end, the verifier can simply generate encryptions of random vectors

  - These vectors are mixed with the permutation matrix
  - Since they are known the verifier can check if the results were a correct shuffle

- The obfuscated mixnet is simply the encryption of the permutation matrix, which is made public

- The mixing is performed by multiplying the encryptions of the inputs with the obfuscated matrix, utilising the homomorphic property of the scheme: $\oplus_{i=1}^{N} Enc(v_i) \otimes Enc(EP_{ij})$ where $\oplus, \otimes$ correspond to addition and multiplication of the plaintexts

Initially the scheme was generic and applied to every homomorphic cryptosystem that was both additively and multiplicatively homomorphic. In [8] a clever modification was made in order to use the Paillier cryptosystem (2.1.1.3), based on the properties we described in 2.1.2 combined with the fact that a permutation matrix contains only the elements 1 and 0.The problem with both schemes is that the element - wise encryption of the permutation matrix has complexity $O(N^2)$ which is impractical even for medium populations.

A further refinement and simplification was given in [9], which uses an additively homomorphic scheme. The offline phase generates encryptions of $|v|$ multiples of powers of 2, where $|v|$ is the size of the vote. That is $V_i = 2^{(i-1)|v|}$. It is well known that a multiplication with a power of 2, causes a left shift in position dictated by the exponent in the binary representation of the number. These encryptions are rerandomised and shuffled and then assigned to each voter $V'_{\pi(i)}$.

In the voting phase each voter computes $({V'}_{\pi(i)})^{v_{\pi(i)}} Enc(0, r_i) = Enc(v_{\pi(i)} 2^{(\pi(i)-1)|v|}, r_i)$. In effect the voter has calculated the encryption of his vote, offset by $(\pi(i) - 1)|v|$ bits. It is clear that the $|v|$ guarantees that the shifts will not overlap each other.

In the online phase all the ciphertexts are multiplied. As a result the shifted votes are added together. In binary notation, all the permutted and ofset votes are concatenated. The result is decrypted, and each vote is detached from the plaintext. More concretely:

$$\prod_{i=1}^{N} Enc(v_{\pi(i)} 2^{(\pi(i)-1)|v|}, r_i) = Enc(\sum_{i=1}^{N} v_{\pi(i)} 2^{(\pi(i)-1)|v|}, \sum_{i=1}^{N} r_i) = Enc(v_1 || \cdots || v_N, \sum_{i=1}^{N} r_i)$$

It useful to note, that the approach in [9] is a combination of a mixnet (offline phase), with a homomorphic cryptosystem (online phase). In order for the scheme to work all the votes should fit in a ciphertext, that is $N|v| \leq |c|$, where $|c|$ is the length of the ciphertext of the cryptosystem used. This limits the application of the scheme, to small populations.

Chapter 5

Enhancing Privacy

In this chapter, we shall review some approaches to electronic voting that do not fit into any of the approaches discussed in the previous chapters. These approaches share the common characteristic that they defend against stronger adversaries targeting the privacy of the votes.

5.1 Voting with Blind Signatures

Exploiting the anonymity feature of blind signatures for electronic voting was one of the goals of the original Chaum paper [29] that put them forth. The scheme proposed there, provides ballot secrecy and individual verifiability:

- The voter submits a blinded version of the ballot along with his registration information.
- The voting authority validates the voter data, and if the voter has the right to vote, signs the blinded ballot and returns it to the voter.
- The voter validates the signature of the authority and posts the signed ballot *anonymously*.
- The authority receives the signed ballots, validates its signature and posts them to a bulletin board for verification. Verification is done through a random pattern that the voter has embedded into the ballot, known only to him.

The problem with [29] is twofold. Firstly, it lacks fairness as the authority knows the intermediate voting results. In addition in the case of dispute the voter must show its vote, thus defeating privacy. A solution was proposed in [57] that is based on separation of functions between 2 entities:

- The election authority, that knows the voter's identity but not the actual vote. As a result it can provide authorisation and authentication checking voter eligibility.
- The tallying authority, that knows the vote but not the identity. As a result it can provide the counting.

In greater detail, the scheme assumes an authority with a public and private key pair (e_A, d_A), and voters each with their own key pairs (e_I, d_I). It proceeds in phases:

1. **Preparation**

   (a) The i-th voter prepares her vote v_i

(b) Based on the vote and using randomness rc_i, she creates a ballot $b_i = commit(v_i, rc_i) = g^{rc_i}h^{v_i}$. The bit commitment scheme ensures that the voter cannot behave differently in the preparation and generation phases. Moreover it substitutes the random pattern that the user must embed into her vote in order to verify it in the bulletin board.

(c) Using randomness rb_i and the public key of the authority she creates the blinded ballot $bb_i = blind(b_i, rb_i) = b_i rb_i^{e_A}$

(d) Using her private key she signs the blinded ballot: $sbb_i^I = sign_{d_I}(bb_i)$

(e) She submits the message (i, bb_i, sbb_i^I) to the election authority, where the index i denotes identity information for the voter.

2. **Authorisation**

(a) Upon receipt, the authority validates voter's signature, voter's eligibility from the identity information and checks for double requests so as to defend against double voting.

(b) If all checks turn out ok, it signs the blinded ballot $sbb_i^A = sign_{d_A}(bb_i) = b_i^{d_A} rb_i$

(c) The authority sends sbb_i^A to voter i and announces the total number of voters

3. **Voting**

(a) Upon receipt, the voter unblinds the signed blind ballot, obtaining the signature of the authority to the ballot. $sb_i^A = unblind(sbb_A^i) = b_i^{d_A}$

(b) The signature is validated with the public key of the authority. Authority cheating can be proved by showing b_i, sb_i^A, without revealing the vote, so that everybody can verify that the signature is invalid.

(c) If all checks turn out ok, then she **anonymously** sends b_i, sb_i^A to the counter. The use of an anonymous channel is required, to hide the network identity of the voter from the counter, who should not to be able to trace it back to the real identity.

4. **Collecting**

(a) The counter validates the sb_i^A with the public key of the authority

(b) If all checks turn out ok, it publishes an indexed list of all the ballots along with the signature $\{idx, b_i, sb_i^A\}$

5. **Opening**

(a) Each voter validates that the voters published by the authority equal the number of ballots and that her ballot is included on the list, with index idx. A corrupt counter, that has not included or has altered the ballot, can be uncovered by presenting the valid ballot and the valid signature by the authority.

(b) If all checks turn out ok, she sends anonymously the decommitment value idx, rc_i.

(c) A corrupt voter might send an invalid opening value in order to cancel the vote. This is a weakness of the current scheme.

(d) A corrupt counter might claim that it received invalid values and cannot open the commitment. The offended voter might object, but since the opening values are sent anonymously the dispute cannot be resolved anonymously. A solution would be to use a verifiable mixnet in this stage, but this will hamper the performance.

6. **Counting**

(a) Now all the ballots can be made public, by opening the commitments. Since the voting phase has ended nobody can benefit from knowledge of a premature and partial result. This guarantees fairness. In addition the ballots to be opened are anonymous, since they were transported through an anonymous channel and lack any other identifying information.

(b) After opening, the votes are checked to conform to the voting scheme. Then they are counted and the result is announced.

(c) Everybody can verify the result, by computing it on their own.

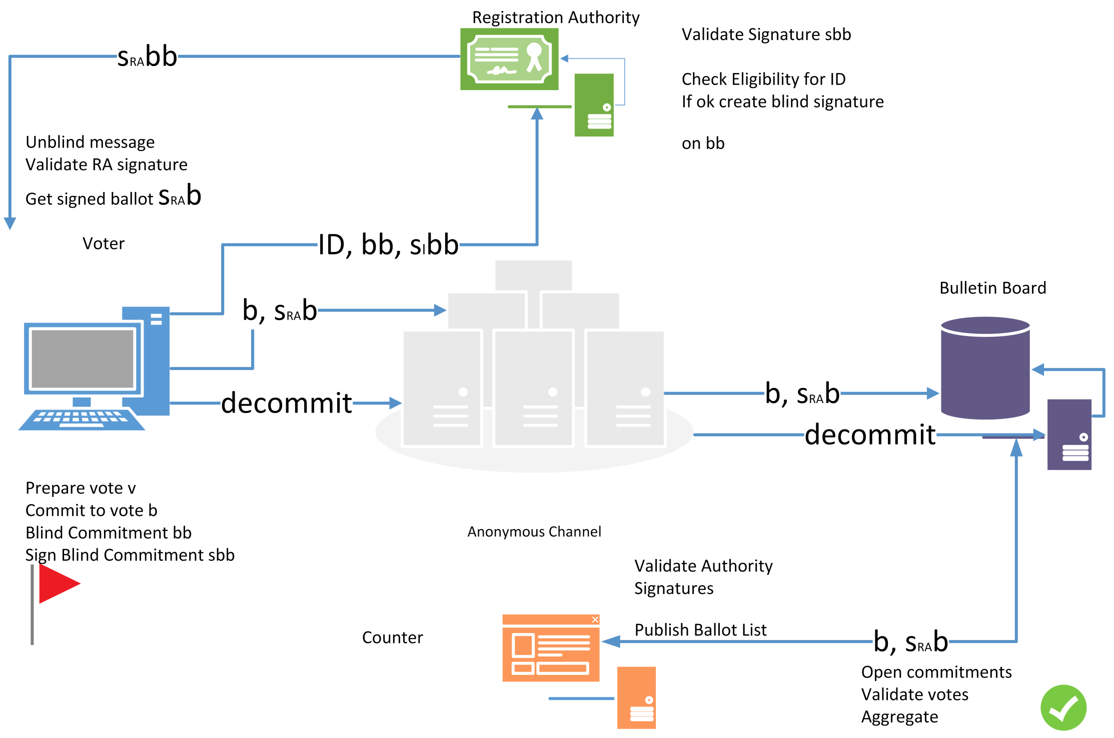


FIGURE 5.1: Voting with blind signatures [57]

The scheme offers privacy in the form of anonymity, fairness, individual verifiability. Its actual efficiency depends on the actual form of the anonymous channel used in the opening and voting phases.

On the other hand, universal verifiability is problematic in our opinion. It might be the fact that everybody can check that the published decommitments indeed open the commitments and that the result corresponds to the opened values. However as we saw dispute resolution is problematic, as a corrupt voter or a corrupt counter can nullify votes without being able to prove invalidity.

Another drawback is that the scheme requires voter interaction in at least three stages, namely during authorization, voting and opening. The last of these is the most problematic, as it requires from a voter to wait until everybody else has cast their votes. This provides inefficiency especially if we contrast it to the *vote and go* approach of [44](3.3.2).

Moreover it does not provide the stronger forms of privacy, namely receipt freeness and coercion resistance. The commitment serves as a receipt of voting. As a result, the voter can sell his vote, and prove his offer by providing a v_i, rc_i that open the commitments. Alternatively a coercer might provide the openings himself and check the bulletin board for compliance.

Variations Despite its drawbacks, a blind signature based scheme is conceptually simple, efficient and supports many social choice functions. As a result many improvements have been proposed.

In [101], the authors reduce the voter interaction by one step. They achieve this by replacing the commitment scheme with a threshold encryption scheme. More specifically the counter receives a (public, private) key pair. To maintain confidentiality the private key is split into m parts that require a subset of minimum size t to reconstruct. Instead of committing to her choice, during step 2 of the preparation phase, the voter encrypts with the public key of the counter. The encrypted vote is then blind-signed by the registration authority, and sent with the signature to the $\mathcal{BB}$ during the voting phase. This is the last step of the voter interaction, assuming nothing goes wrong. After everybody has voted, the voter can simply check the$\mathcal{BB}$ for her encrypted vote and object if it is not found by revealing the authority signature (*Note:* This step does not count towards the total number since it is not mandatory). Instead of the opening phase, the counters collectively decrypt each vote and write the result to the bulleting board. An omission of [101] is that they do not require a proof of correct decryption, but this is purportedly dealt by the existence of the threshold scheme. Subsequently the votes are aggregated.

Another variation from He and Su [69] to the [57] scheme gives the voter the option and capability to change her vote if she wishes. This is an interesting problem that has not been studied in the literature and can potentially have important consequences. Each voter is required to have a public, private key pair. Instead of signing an encrypted/committed ballot, the authority blindly signs the voter public key. Consequently, instead of submitting a ballot, the voter submits their public key. The actual ballot sent during the voting phase the is encrypted using a symmetric cipher. To decrypt it the voter must 'open'it, by sending the symmetric key. It will, of course, be encrypted with the certified public key, which acts as a sort of *blind identifier* and allow to group the votes. As a result, if a voter changes his mind, she can omit to send the symmetric key, and/or resubmit the new in a similar manner. To prevent double voting, the counter can check that there is only one decrypted vote per public key or retrieve the last vote per public key.

The scheme of [57], can provide an idea on how to build a simple anonymous (one-time) login system that can be used for voting. Such a system can be used where there is a requirement for voting, but the requirements for integrity and secrecy are of little importance and only anonymity is required. Such a system was discussed in [67].

- The participants register with a registration authority using their real names and passwords.
- When voting is required the participants create a random credential m and commit to it using a hash function $h(m)$
- Then the credential is blinded, using for example the technique described in chapter **??** creating $b_m = h(m)k^e mod n$
- To validate that they are indeed authorised to vote, they login with their real credentials and submit b_m
- The registration authority checks that the login credentials are valid and that the participant is authorised to vote. In that case, it signs the blinded credentials $bs_A = b_m^d = h(m)^d k^{ed} = h(m)^d k \pmod n$ and presents them to the participant
- The participants unblinds the signature and retrieves the signature on the committed credential: $bs_A k^{-1} = h(m)^d = s_A$
- As a result the participant has an anonymous username m signed by an authority that can validate his identity.
- Now when voting is due, the participant provides m and s_A to a login form. The voting system retrieves the public key of the authority and validates whether $s_A^e = h(m)$. If the signature is valid then the participant can vote.

A variation of this idea has been also proposed in [27]. Instead of a one-time login, the voter can create multiple identities, called *pseudo-voter identities (PVID)*. The registration authority blindly signs each one of them. Each PVID should be used with a single authority (either tallying, verifying etc.)

An interesting implementation of the main scheme presented in this paragraph was given in [95]. It avoids the malicious software vulnerability found in client computers by using smartcards that can execute programs written in the Java programming language. In the most recent version these programs can execute network calls and implement the HTTP protocol both as client and as servers. Furthermore, the Java Information Flow, controls the reading and writing of data between the processes that execute in the Java card or on the network. The interesting aspect of their work in the protocol level is the implementation of the anonymous channel. Instead of using a mixnet they execute the blind signature protocol twice. Once to get a blindly signed username for the voter and once to get a blind signature on the ballot.

Proposals that add receipt freeness to [57] will be reviewed in section 5.2.3.

5.2 Coercion Resistance

In chapter 1 we noticed that a non-technical threat to ballot secrecy is coercion, the involuntary or voluntary dictation of the vote contents. Coercion is very easily achieved if the voting system provides any kind of receipt as to how the voter actually voted. As a result, a desirable property of any voting system, is the inability of the voter to construct a voting receipt. This property is called receipt-freeness and was first defined in [18].

5.2.1 Receipt Freeness

The main difficulty for achieving receipt freeness is that any voting system based on probabilistic encryption cannot be receipt free. Since the voter uses a random value to construct his vote, this random value can serve as a receipt. The coercer simply retrieves the random value and compares the encryption of the alleged vote with the actual secret vote. As a result further modifications must be made to any such scheme, the most important one being that the voter *should not create the vote encryption on his own*.

The Benaloh-Tuinstra [18] protocol was the first to claim the receipt-freeness property. It utilises the concept of a voting booth which of course is inspired by the concept found in traditional elections, offering isolation to the voting agent.

Definition 5.1. A voting booth is an addon to the voter that provides information theoretically secure communication between two entities.

An interpretation of this definition is that, a voter in *booth mode* cannot initiate communication with external processes by sending neither public nor private messages. On the other hand the voter can receive messages from external processes. The voting booth provides the voter plausible deniability, allowing him to construct a receipt for all possible votes, thus making it useless. The rest of the protocol is the scheme presented in chapter 3 by Benaloh, Fischer and Yung in [40] and [19]. The difference is that the vote preparation protocol is executed by the authority instead of the voter, while he is inside the booth. The vote casting phase is executed in *normal mode*. In more details:

1. The authority generates $\eta + 1$ ballots as randomised encryptions of a $yes - vote$ and a $no - vote$ in random order.
2. Inside the voting booth, the voter receives the decryption of a specific component for each of the $\eta + 1$ ballots.

3. The authority proves publicly that the ballots are of the correct form as in [40].
4. The voter *indicates* the preferred index in the first ballot, to be his vote.

Every *'external'* observer and possible coercer will be convinced that the ballot is valid by the third step in the process. However, in step 2, the voter actually learns the decryption of the ballot and can correctly indicate the choice of preference.

Of course the receipt-free version of [40] suffers from the same flaw: The authority can progressively decrypt each subtally before and after each vote is cast, thus learning each individual vote. A receipt free version of [19] fixes this problem by having the voter distribute his vote to multiple authorities. Specifically:

- Each authority A_j selects a blinding factor f_j and supplies its encryption e_j to each voter
- Inside the voting booth, the voter is given f_j and convinced with overwhelming probability that the encrypted value given previously corresponds to the actual blinding factor without yielding a proof
- The voter selects the vote v_i and a polynomial P_i in order to share the votes to the authorities. $P_i(0) = v_i$
- Each share is blinded by calculating $y_j = P(j) - f_j$
- Each voter releases the pair (y_j, e_j) for all authorities. This tuple is the actual vote
- The authorities add the blinded shares and multiply the encryptions for all the voters. Since the encryption is homomorphic the product of the e_j can be decrypted and used to unblind the sum of the blinded shares
- Each authority now has a point for the combined voter polynomial $Q = \sum_{i=1}^{N} P_i$ where $Q(0) = \sum_{i=1}^{N} P_i(0)$ yields the election result
- At least t authorities can reconstruct the polynomial and retrieve the tally

The above scheme has a loose end. The voter must prove that his vote is valid. This can be done with η random bits. The voter constructs $\eta + 1$ ballots (in total) and uses the probabilistic technique of [40] to prove the master ballot's validity. This technique however can provide a receipt of the vote, as remarked in [71]. The coercer can force the voter to commit to a particular string of ballots and can watch the verifications. The coercion succeeds with the same probability that the ballot is valid, namely $1 - 2^{-\eta}$.

The second approach to receipt free voting schemes was presented in [84], which we reviewed in chapter 4 as the first verifiable mixnet. The receipt free component of the protocol rests again on the idea that the encrypted votes must not be prepared by the voter, but by the system. However the system must convince the voter of the validity of the encryption in a way that is not transferable to a coercer. For the first part [84] utilises the mixnet *backwards*. The last mix server prepares encryptions of a pair of *yes/no* votes in random internal order for each voter. He shuffles and reneencrypts the lists and proves the correctness of the action using the cut and choose protocol. He then sends the list to the next mix server, which is actually the previous one. Each mix server rearranges randomly the internal order of the votes and proceeds to his normal reencryption. When the mixnet terminates the voter must pinpoint the encryption destined for him in the shuffled list and choose internally his preferred option, either yes or no. To be able to do so, he must know the internal reorderings, the shuffles and the reencryptions. Each mix server must commit to these actions and send the commitments to the voter. The opening of the commitments will provide the necessary information to the voter. It must be done however in a manner that does not provide a receipt. As a result all the commitments given to the voter are *chameleon blobs*, which are bit commitments that can open to all possible values. The chameleon technique was due Chaum, Brassard and Crepeau. In order to vote, the voter selects the correct component and uses the mixnet in the normal forward fashion. The Sako-Killian scheme provides receipt freeness without a need for a voting booth. The only requirement is that the channels used for the transmission of the decommitments are *physically untappable*.

5.2.2 Adding Receipt Freeness to Homomorphic Schemes

This approach of [84] was pursued further by Hirt and Sako in [71], where a solution to the problem of [18] described above was presented, along with a framework to provide receipt freeness to any voting scheme based on homomorphic cryptosystems. The main idea is the one in [84] which frees the voter from the need to encrypt his choice. Again, the voting authorities prepare encryptions of the possible votes and the voter simply picks the choice of interest. Of course, ballot secrecy must be maintained. To achieve this, the voting authorities operate as a reencryption mixnet. The voter is notified of all the permutations and reenecryption factors used. As a result he can follow his vote of choice through the mixnet and pinpoint it in the end. The notification is tailor-made for each voter in a way that cannot be used by any coercer. This is achieved using designated verifier proofs. The work of [71] requires only *untappable one way communication channels* from the authorities to the voters.

Definition 5.2. An untappable channel is an addon to the voter that provides information theoretically secure one way communication to another party.

In more details their protocol operates in the following phases:

1. **Vote Generation**
   (a) Each valid choice $v_i \in \{v_1, v_2, \cdots v_C\}$ is deterministically encrypted by the public key encryption algorithm (using predetermined randomness) producing the list of the encryptions of the valid choices $L_0 = \{e_1^0, \cdots, e_C^0\}$. Everybody can validate these encryptions by encrypting each choice with the randomness agreed.
   (b) Each authority A_k reencrypts and permutes L_{k-1} thus producing L_k
   (c) Each authority *publicly* proves that each vote in L_{k-1} has a reencryption in L_k.
   (d) Each authority *privately* sends using the untappable channel the permutation π_k it applied and *privately* proves that it is valid.
2. **Vote Casting** Using the permutation information received from each authority the voter tracks down the output item that encrypts his choice. The vote cast is the index of the desired item.
3. **Vote Tallying** Since the scheme applies to homomorphic cryptosystems, the authorities compute the homomorphic function and decrypt the tally, while proving its correctness.

The correctness of the protocol relies on the public proof in the description above while the receipt freeness relies on the subsequent private proof. The actual proofs are cryptosystem dependent. We follow [71] and present them for the exponential El Gamal cryptosystem as used in [44].

In the first case the prover wants to prove that an encrypted item has a reenencryption in a list of items, without revealing which item is the reencryption. This can be accomplished by applying the framework of witness indistinguishable proofs of [42]. More specifically the prover wants to convince the verifier that there exists a reencryption for the encrypted vote (x, y) in a list of encrypted votes $\{(x_1, y_1), \cdots, (x_L, y_L)\}$ without revealing its position t or the reencryption factor ξ. Let $(x_t, y_t) = (g^\xi x, h^\xi y)$ the reencrypted item.

- The prover randomly selects $\{d_i\}_{i=1}^C, \{r_i\}_{i=1}^C$ and calculates $\{a_i = \frac{x_i}{x}^{d_i} g^{r_i}\}_{i=1}^C$ and $\{b_i = \frac{y_i}{y}^{d_i} h^{r_i}\}_{i=1}^C$. For the actual reencrypted value the prover calculates $a_t = g^{\xi d_t + r_t}$ and $b_t = h^{\xi d_t + r_t}$. The prover offers the calculated values to the verifier.

- The verifier randomly selects a challenge c and returns it to the prover.

- The prover recalculates d_t as $d'_t = c - \sum_{i=1, i \neq t}^C d_i$ and finds r'_t such that $\xi d_t + r_t = \xi d'_t + r'_t$. Then he updates the values of d_t, r_t to d'_t, r'_t. Finally he submits the values $\{d_i\}_{i=1}^C, \{r_i\}_{i=1}^C$ to the verifier.

- The verifier validates that:
  - $c = \sum_i d_i$
  - $\{a_i = \frac{x_i}{x}^{d_i} g^{r_i}\}_{i=1}^C$
  - $\{b_i = \frac{y_i}{y}^{d_i} h^{r_i}\}_{i=1}^C$.

In the second case the prover wants to convince the verifier that an item is a reencryption of another in a way that the verifier can fool a coercer. It is assumed that the verifier has a (public,secret) key pair (h_{ver}, z_{ver}). The main idea of the proof is that the verifier can use her private key in order to forge the proof for the coercer, while maintaining its validity. The item in question is that (x', y') is a reencryption of (x, y) with witness ξ or the relation $(x', y') = (g^\xi x, h^\xi y)$:

- The prover randomly select d, w, r and sends to the verifier the values $a = g^d, b = h^d, s = g^w h_{ver}^r = g^{w + z_{ver} r}$

- The verifier selects a random challenge c and sends it to the verifier.

- The prover calculates $u = d + \xi(c + w)$ and sends u, w, r to the verifier.

- The verifier validates that:
  - $s = g^w h_{ver}^r$
  - $g^u = \frac{x'}{x}^{c+w} a$
  - $h^u = \frac{y'}{y}^{c+w} b$

The verifier using his knowledge of the secret key can find w', r' such that $w + z_{ver} r = w' + z_{ver} r'$ and as a result can forge the proof for any (x'', y''), with the same offers. The values (x'', y'') which may play the role of other encrypted options.

5.2.3 Adding Receipt Freeness to Blind Signature based Schemes

As we mentioned in 5.1, the blind signature scheme of [57] is not receipt free. The commitment can serve as a proof of vote, which the voter can sell or the coercer can demand. The standard solution to this problem, proposed by [102], is to use a trapdoor bit commitment scheme (2.1.4.3) instead of the standard one.This means that the voter must create a new private and public key pair for use in this phase. Commitment will be made using the public key, so that opening will require the private key, allowing the voter to open it in multiple ways. The scheme proceeds as usual. The resulting protocol is receipt free since the voter can lie about the commitment, but it does not protect against a stronger attacher, who might require the voter to throw away the private key and vote using a public key provided by the coercer. A solution given in [103] requires a set of entities collectively referred to as the *parameter registration committee - PRC* with the assumption of an untappable channel or a voting booth. More specifically the scheme works as follows:

- Secret sharing of the commitment private key: Let a_i be the commitment key and $G_i = g^{a_i}$. It is split in n parts a_{ij} such that $a_i = \sum_{j=1}^{N} a_{ij} \quad (mod\, q)$ and a commitment $G_{ij} = g^{a_{ij}}$ is created for each one. The registration authority blinds G_i and eachG_{ij} along with the committed vote. During voting the parts a_{ij} are sent to each RPC member through the untappable channel along with G_i. They compute G_{ij} on their own and submit it to the bulletin board. The commitment is reconstructed during counting as $G_i = \prod_j G_{ij}$.

- Verifiable threshold secret sharing of the commitment private key. The first version has the downside that even one corrupted authority can block the scheme by refusing to submit their shares. As a result a t out of n scheme can be used.

- Voting booth: The voter proves using a voting booth that he knows the a_i

The main idea of this approach is that the voter can split the private key of the trapdoor bit commitment. As a result the public key is divided accordingly and the registration authority blinds each individual share of it. During the voting phase, the voter sends each share of the private key to each PRC member using an untappable channel. They reconstruct the public key shares and submit them to the bulletin board. During counting the tallier validates the public shares and reconstructs the public key. Afterwards the votes are counted. It must be noted, that in order to disallow malicious PRC members to disrupt the process, the secret key must be distributed using a verifiable scheme.

An alternative method (still from [103]) does not require the help of extra members but requires strengthened security in the form of an anonymous voting booth, which enables the voter to prove knowledge of the private key.

A more general treatment of the following situation will be given in the following paragraph.

5.2.4 Strengthening the attacker

Receipt freeness does not imply coercion resistance. An attacker can influence the outcome of an election without the need for a receipt by forcing a voter to:

- abstain from the voting process alltogether
- vote in a way that cancels out a particular vote
- vote randomly, in effect nullifying a vote and denying the right of a voter to express his opinion
- give away the secret keys (and be impersonated)

These extended capabilities of the attacker were firstly presented in [77]. For example here is how these attacks can be launched in a mixnet environment:

1. The adversary retrieves the private key for a voter
2. Then he selects its own preferred vote v
3. and encrypts it with the public key of the mixnet $u = E_(pk_{MN}, v)$
4. The vote is signed with the stolen private key $s = E(sk_i, E_(pk_{MN}, v))$
5. The pair (u, s) is appended to the bulletin board
6. Signature validates and so the vote is anonymised by the mixnet

The above attack is an impersonation of the voter. However it can be applied as a coercion attack, if the attacker skips the first three steps and simply presents the voter with the ready ciphertext to be cast. He can then check that the particular voter submitted to his coercion by checking the bulletin board for the pair (u, s).

A framework to counter such attack was proposed in [77], having in mind application to Internet voting. The main idea, is that a voting system has the property of coercion resistance, if the aspiring attacker *cannot tell whether the voter actually succumbed to the coercion attempt*. In order to achieve this the voter can cast **more than one** ballots with invalid credentials thus fooling the attacker. The vote to be counted will be the one containing the valid credential. Of course the scheme is dependent on some assumptions, which however, are more relaxed than a physically untappable communication channel *throughout* the election that we saw earlier:

- The attacker must be uncertain about the behaviour of at least some voters (apart from the victim). These means that the attacker cannot control all the voters.
- The vote casting channel must be anonymous. This is essential in order to thwart the forced abstention attack, because if the votes are not anonymised but are only secret than the coercer can check them to verify if the target voted indeed. As a result an anonymous channel is an essential prerequisite to achieve coercion resistance.
- The voting credentials must be anonymous and indiscernible ones must be able to be generated. In order for this assumption to hold, the credential acquiring phase must be conducted without the attacker watching - through an untappable channel. This requirement might seem strict, but it applies only to the registration phase which happens rarely, sometimes even only once, and can be reused for many elections.
- The voter has at least a moment of privacy, when he can cast his real vote.

If these requirements are met then the protocol of [77] can provide coercion resistance. The main idea is that during the registration phase voters are given anonymous credentials which are recorded in a voter credential list (referred to as the *voter roll*). These credentials can be faked. As a result, during a coercion attempt the voter uses the fake credential, which cannot be distinguished from the real one. This fake ballot is treated as a normal one by the system, except during the tallying phase where it is not taken into account. This can be achieved, because the tallying authority can compare the credentials embedded into the ballot, with the ones in the voter roll and count the votes corresponding to the real ones. These authorities should not be fully corrupted by the coercer, so that he cannot have access to the voter roll and validate the voter credentials. In more detail the protocol proceeds as follows:

1. **Setup Phase:** Key pairs are created for registration R and tallying authorities T, namely (pk_R, sk_R) and (pk_T, sk_T) . Corruption of a minority of registration authorities can be tolerated.
2. **Registration Phase:** During this phase, the voter identity is validated and the voter credentials are generated. The voter credentials are a public and private pair (pk_i, sk_i) derived from a random anonymous token σ_i which can be aa simple as a random number (or more elaborate as is the case in later refinements). The anonymous token is encrypted with the public key of the talliers and placed on the bulletin board $S_i = E_{pk_T}(\sigma_i)$. This is the public component, while the token itself is the private component, which is transmitted using the untappable channel. The voter roll is built from the public encryptions $VR = \{S_i\}_{i=1}^{N}$. In order to avoid corruption during this phase:
    - The credential is generated in a distributed fashion

- The voter must delete all the transcripts from the interaction with the registration authority *or*
- The coercer cannot corrupt any registration authority *or*
- The voters can distinguish the corrupted players.

3. **Voting Phase:**
   - Ballots are cast in the bulletin board as usual.
   - The ballot is a modified El Gamal encryption of both the candidate choice and the credential. More specifically:
     $B = (E(v_i), E(\sigma_i)) = ((g_1^{r_1}, g_2^{r_1}, v_i h^{r_1}), (g_1^{r_2}, g_2^{r_2}, \sigma_i h^{r_2}))$ where r_1, r_2 are random values
   - The ballot is completed by proofs of knowledge of vote and credential, proof of vote validity, namely that the candidate index is valid, and proof that the first two components of the encryptions use the same randomness. These proofs are essential to achieving coercion resistance, by the following rationale:
     - Since the voter roll is public, the attacker might spoof a credential by reencrypting a voter roll entry. A legitimate voter must prove that he knows the credential.
     - An invalid vote might indicate a forced abstention or a randomisation attack.
   - The voting phase takes place using an anonymous channel. This is very important as to thwart the forced abstention attack and to make the coercer unaware of the actual vote position in the bulletin board.

4. **Tallying Phase:** The authorities collect the ballots from the bulletin board and compute the election result. Because there are more ballots than voters, on account of coercion evasion, there must some preprocessing. This pre-tallying phase was aptly named *vote authorisation* by [120] and consists of the following subphases:
   - **Invalid Ballot Removal:** Before counting begins, the proofs of correctness are extracted and validated. Ballots with invalid proofs are discarded
   - **Duplicate Ballot Removal:** If multiple ballots correspond to the same credential, they are filtered and a single vote per credential is kept. To this end:
     - From ballots with valid proofs, two lists are created. The first one contains the encrypted votes and the second one the encrypted credentials.
     - The list of credentials is filtered and duplicate entries are deleted. The duplicates are found using the PET technique described in 2.1.7.6.
     - The removal is cascaded to the list with the encrypted votes. Given multiple duplicate entries the one to keep can be identified using a *first vote counts* or *last vote counts* rule.
     - Both lists that now contain distinct items are merged.
   - **Fake Ballot Removal:** Votes with fake credentials must be eliminated. To this end:
     - The merged list of the previous subphase is forwarded to a verifiable mixnet.
     - The voter roll is forwarded to the verifiable mixnet as well.
     - The mixed credential list is compared to the mixed voter roll leading to the discovery of the fake credentials using PET. Both the fake credentials and the corresponding votes are removed.

    - The reason for employing a mixnet is for the coercer to lose track of the credential index in the voter roll and the vote index in the bulletin board. If this particular vote corresponds to a fake credential it will be removed from the bulletin board before counting without raising any suspicion.
  - The resulting list of encrypted votes is decrypted and the votes are counted.

  During the tallying phase the adversary can only corrupt a minority of the authorities.

The above scheme inherits the property of universal verifiability from the mixnet used in the tallying phase. Other properties for which a security proof is provided include correctness and verifiability. However our focus in studying [77] is coercion resistance which is proved using the following game:

- The target voter flips a coin b
- If $b = 0$ then the voter evades coercion by using a fake credential
- If $b = 1$ the voter submits to coercion
- Coercion resistance is achieved when the adversary has a negligible advantage on guessing b

The proof will involve comparison of the output of the adversary in the real experiment with the corresponding output in an ideal experiment [77].

The real experiment proceeds as follows:

- The adversary $\mathcal{A}$ selects the controlled voters. It is assumed that except for the coercion target the attacker can control the behaviour of a subset of voters, such that the uncertainty of the total voter behaviour is maintained.
- Voters register and receive the credentials
- $\mathcal{A}$ sets the coercive target and its choice
- The voter flips a coin b
- If $b = 0$ then the voter makes a voting choice and returns a fake credential to the adversary to evade coercion
- If $b = 1$ then the voter votes, and returns the real credential to the adversary
- It must be remarked that in both cases the voter actually votes as the adversary suggested. Even in such a favourable case that attacker cannot deduce this from the credentials.
- The rest of the honest voters cast their ballots
- $\mathcal{A}$ posts to the bulletin board the votes of the controlled voters
- The tallying authorities count the votes
- $\mathcal{A}$ gets the result and tries to guess if the coercion attempt succeeded, by setting b'
- If $b' = b$ then output 1 else output 0

On the other hand in the ideal experiment the adversary $\mathcal{A}'$ is passive, and the only information at his disposal is the tally result computed in an ideal way:

- The adversary $\mathcal{A}'$ selects the controlled voters.
- Voters register and receive the credentials
- $\mathcal{A}$ sets coercive target and choice
- The voter flips a coin b
- If $b = 0$ then the voter makes a voting choice
- The voter *always* returns the real credential to the coercer since in the ideal experiment the attacker should not learn anything from the keys.
- $\mathcal{A}'$ posts to the bulletin board the votes of the controlled voters
- The tallying authorities count the votes in an *ideal* way, which means that they know all the private keys, can tell the controlled voters apart. More specifically:
  - Ballots by honest voters are counted normally
  - Double votes are not counted
  - Corrupted players' votes are not counted
  - If $b = 0$ (evade corruption) the target vote is not counted
  - If $b = 1$ (submit to corruption) the target vote is counted

  In short ideal tallying counts the votes of the honest voters and the target voters only if he submit to coercion
- $\mathcal{A}'$ gets the result and tries to guess if the coercion attempt succeeded, by setting b'
- If $b' = b$ then output 1 else output 0

If the powerful adversary $\mathcal{A}$ has no advantage in guessing if coercion succeeded than $\mathcal{A}'$ than the election protocol is coercion resistant.

Coercion resistance is proved in the following theorem:

Theorem 5.3. *If the Decisional Diffie-Hellman (DDH) Assumption holds, then the protocol provides coercion resistance*

Proof.

A simulator $\mathcal{S}$ capabe of simulationg all the components in all the phases of the election is constructed

$\mathcal{S}$ takes the honest votes and simulates $c - resist$

Instead of the votes $\mathcal{S}$ provides random ciphertexts for the honest voter's ciphertexts and the output of the mixnet

Due to the DDH the adversary cannot distinguish the simulated ciphertexts with the real ciphertexts that would come up in an election

As a result $\mathcal{A}$ in fact interacts with $c - resist - ideal$ and is no more powerful than $\mathcal{A}'$ $\square$

5.2.5 Implementation Efficiency, Security and Practical Improvements

The method in [77] requires comparison by '*PETting*' the ballots to check for duplicates and fakes. Each ballot is checked with every other ballot for duplicates, based on the credential. Subsequently each ballot is checked against the voter roll for fakes. If v is the total number of votes, then these operations requires $O(v^2) + O(vn)$ comparisons, a quadratic quantity *in the number of votes cast*, not in the number of actual voters. Due to the nature of coercion resistance, $v >> n$ since a voter might cast many votes with different credentials. This constitutes the method impractical even for medium populations. The next logical step, would be to improve the efficiency of the method with the goal of converting it to linear complexity. The standard method of converting such quadratic methods to linear involves hashtables. However it cannot be immediately applied to this setting, as both the ballot credentials and the voter roll credentials are encrypted using randomised algorithms. As a result, some intermediate steps are required.

A first line of relevant proposals are due to [130] and [134]. They proposed a way to remove the randomisation of the encrypted credential, without revealing it, and thus make hashtables relevant. More specifically, if $(u, v) = (g^r, \sigma h^r)$ is an El Gamal encryption of credential σ, where s is the shared private key and $h = g^s$ is the public key and r the randomness employed, then the randomness can be removed as follows:

- The authorities select a second private key z which should be generated in a distributed fashion.
- They blind the ciphertext by computing $(u^{sz}, v^z) = (g^{rsz}, \sigma^z g^{rsz})$
- Subsequently they divide the components which leave only the plaintext credential blinded σ^z
- These accomplishes the removal of the randomness.
- The hashtable approach can now be used

[130] proposes, with efficiency in mind, to split the blinded credential to two pieces and use the first half as a key to the hashtable. This approach is problematic since it will imply an increase in hashing collisions. This side-effect is particularly problematic during fake detection and removal, since a valid credential might collide with a fake one, and be wrongly removed, thus altering the election result and violating fairness. [134] skips this problem by using the complete blinded credential and appropriately adjusting the system parameters, such that σ^z is unique.

Both these approaches suffer from a variant of the tagging attack. To start with ([12], [38]), the coercer is able to retrieve the encryption of the voter's credential. Then he can utilize the malleability of the encryption scheme and construct a ciphertext that is somehow related to the original. For instance the coercer can compute the product of the encrypted credential and the encryption of a random value known to him. From this he computes as a tagged value, the encryption of the product of the credential and the random value. The decryption mixnet then decrypts all the votes. Using the knowledge of randomness and the tagged value the coercer pinpoints the position of the credential and the vote. Afterwards, he can find out if the particular vote passed the credential validity test by checking if it was removed from the final output and thus can know if the provided credential was valid or not. This particular attack can even succeed in the blinded versions of [130] and [134] if the blinding scheme is malleable and allows the tagging of credential by some operation [11].

A more concrete implementation was presented in [124]. In this particular scheme, only two credentials are needed, the real one and the fake one. At this point, it must be pointed out, that the fake and real credential are indistinguishable in form and content. The property that makes a credential valid is that it is included in the voter roll. The two credentials are included in an observer: a tamper resistant device like a smartcard.

Civitas In [38], the scheme of [77] is elaborated in details and an implementation is described. First of all, a concrete specification of the credential generation and distribution is given and the faking mechanism is discussed.

To generate a credential, a set of registration tellers cooperate:

- Each teller generates a share s_j of the credential, which is encrypted using standard ElGamal yielding S_j, which is placed in the bulletin board.
- The public part of the credential is calculated as $S = \prod_j S_j$
- Due to the multiplicative homomorphism of El Gamal $S = Enc(\prod_j s_j)$

The untappable channel is implemented using the concept of designated verifier proofs 2.1.4.3 through an untappable channel.

- Each teller gives its credential share to the voter and a mechanism to validate it.
- More specifically the teller presents $(s_j, r, S_j^{'} = Enc(s_j, r), D)$ where D is a DVP that $S_j^{'}$ is a reencryption of S_i
- The voter is convinced of the share validity, but cannot transfer the proof to the coercer
- The full private credential is constructed by multiplying received shares.

In order to fake the credential, a coerced voter can simply select a fake credential share $\hat{s_j}$, which is assumed belongs to a teller not controlled by $\mathcal{A}$. Its existence is an assumption of both [38] and [77].

To solve the scalability problem, Civitas employs parallelism. The voters are partitioned in *virtual precincts*, called blocks. Vote authorisation and tallying happens independently in each block, and the results are aggregated. As a result the complexity of the scheme is $O(BM^2) + O(BKM)$ where M is the maximum number of votes per block, B is the number of blocks and K is the minimum number of voters per block.

Extra Information and Anonymity Sets [131] noticed that the tagging attack is irrelevant in the duplicate removal subphase. As a result the blind hashtables can be used there thus achieving the goal of linearity in the number of votes. For linear fake removal something more is needed. During the *registration phase* the voter retrieves through the untappable channel the index where his credential is located in the bulletin board, along with the credential itself. During the vote casting phase, the ballot cast is a triple, containing the encrypted credential, the encrypted choice and the encrypted index accompanied with the corresponding proofs. During the tallying phase the index will be decrypted and the encrypted credential will be retrieved from the initial version of the voting roll. A single PET between the retrieved credential and the one contained in the ballot can reveal a fake vote. It must be noted here that there in this scheme a ballot can be excluded from counting not only by providing a σ value that does not appear in the bulletin board, but by specifying a different value for the index. If $1 \leq i \leq n$ the PET will fail. If $i > n$ the ballot will be removed during the first check of the vote authorisation phase. If the voter decides to cheat the coercer by supplying a fake index than the system must make sure that a uniform distribution of fake indices appears. To this end it is proposed that the tallying authorities themselves append fake votes. This has also the effect that a coercer that monitors the bulletin board *cannot tell whether a target voter made use of his moment of privacy*.

This approach is improved in [120] and the *Selections* scheme ([37]), based on the concept of *anonymity sets*, which in general is a set of identical items, only one of which is relevant and the rest are used as a decoy. During vote casting the voter presents the real credential accompanied by $\beta - 1$ credentials from the voter roll. More specifically in Selections:

- The voter encrypts the credential σ using exponential El Gamal. This is done to enable rerandomisation for use in multiple elections and more importantly to remove fake votes.
- During vote casting the voter commits to g^{σ} and rerandomises her entry from the voter roll. In addition she randomly picks $\beta - 1$ encrypted credentials from the voter roll and embeds them into the ballot. Along with the standard [77] proofs she proves that her credential is indeed a rerandomisation of one of the β credentials, without of course revealing which one (ala [42]).
- During the vote authorisation phase the deduplication process occurs in linear time as the same value g^{σ} is present in all of them. The last vote per g^{σ} is kept.
- For fake credential removal the commitment is treated as a deterministic encryption of the credential. As such it is randomised from the standard [77] mixnet and is 'PETted' with the rerandomised voter roll entry. A real vote is determined by a successful such test.

Selections borrows heavily from the self contained nature of [11], which we shall describe later. [120] lifts the processing from the voters by letting them specify the ballot anonymity set without any proof. During the vote authorisation stage the authorities create β ballots for every index in the anonymity set, by retrieving the credentials from the voter roll.

A practical aspect of the coercion resistance schemes that were presented above, relates to how does the voter actually creates the fake credentials when under coercion. This issue is quite vague in the literature reviewed. The [37] scheme tackles this problem using the notion of *panic passwords* [36]. During registration the voter selects a password to be used during vote casting. In reality the voter partitions the space of possible passwords into three categories:

- The actual password to authenticate the voter and submit the real credential
- The panic passwords which indicate that the user is under coercion and generate fake credentials. The interaction however is indistinguishable from the one that takes place with the actual password.
- The inadmissible passwords that indicate authentication failure and do not allow vote casting

Ideally panic passwords should be a sparse subset of inadmissible passwords and the actual password is a pre selected panic password. To illustrate the concept, better, we describe an example implementation from [36], *5-Dictionary*. The password space consists of any combination of five words. The valid passwords are any combination of five words from an agreed upon dictionary. A particular combination is selected as the actual password during registration. Any other combination from the dictionary is a panic password. Any other five word combination from the password space is invalid.

Structure to the credentials Another line of relevant proposals tries to avoid the pitfalls described above by adding structure to the credentials ([11], [12], [13]). This structure aims to make the voter roll irrelevant. To begin with, in [11], instead of σ being a random binary string, it is a quadruple $(r, a, b = a^{y}, c = a^{x+xry})$ where x, y are two secret keys generated by the registration authorities, a is a random number $a \neq 1$ and r is a random number in a group where the Decision Diffie Hellman problem is hard. This implies that an adversary cannot tell whether a quadruple is a valid one or it merely consists of randomly selected components. Another assumption cited in [11] is that an attacker that comes across quadruples (r_i, a_i, b_i, c_i) cannot use them to create a distinct valid one (the LRSW assumption). From the credential component the r part should be kept secret, while (a, b, c) could be made public. In the scheme, the modified El Gamal encryption of [77] is employed during the following phases:

- **Setup** The parameters of the election are created, namely:
  - Two generators g, o of a group of prime order where the DH assumption holds. g will be used for the standard ElGamal operations while o will be used for duplicate removal in linear time.
  - The private keys x, y of the registration authorities and the corresponding public keys R_x, R_y
  - The public key T of the tallying authorities and the corrsponding private key $\hat{T}$
- **Registration** As usual, the voter identity is validated and the credential is sent to the voter through an untappable channel with a DV proof (2.1.4.3) for well formedness. If the received credential (r, a, b, c) is valid, then every credential of the form (r, a^l, b^l, c^l) is valid. The voter can use this fact to vote in generate credentials for many elections
- **Voting** The ballot consists of encryptions with the tallier public key and proofs of knowledge. More specifically the voter computes the tuple $(E_T[\mathfrak{c}], E_T[a^r], E_T[b^r], E_T[c])$. He also provides the value a in plaintext and a blinded version o^r of o with the random credential r. In addition the voter proves that $\mathfrak{c}$ is a valid candidate selection, proofs of knowledge of each encrypted plaintext and proof that the blinding factor is the same in $E_T[a^r]$ and o^r. This is the reason why the value of a is initially unencrypted.
- **Tallying**
  - **Proof Validity Check** The proofs of the ballots are checked and the ones which are not verified are eliminated
  - **Duplicate Removal** The value o^r is passed through a hashtable and based on it the duplicate ballots are removed. Afterwards for all unique ballots the proofs as well as the value o^r are removed. The plaintext a is encrypted using the T.
  - **Anonymisation** The unique ballots are anonymised via a verifiable reencryption mixnet. The output of the mixnet is a quintuple
    $(t, u, v, w, z) = (Reenc(\mathfrak{c}), Reenc(a), Reenc(a^r), Reenc(b^r), Reenc(c))$
  - **Credential Validity Check** The identification of the invalid ballots is done with the cooperation of the registration authorities which is an active participant in the whole process. Instead of supplying the voter roll as in [77], they supply their private keys (x, y) in order to:
    * Compute $v^y = Reenc(a^r)^y = Reenc(a^{ry})$
    * Check if $PET(v^y, w) = 1$. If everything is OK the check should be successful since $v^y = Reenc(a^{ry})$ and $w = Reenc(b^r) = Reenc(a^{y^r})$, which means that both are encryptions of a^{ry}
    * The registration authorities compute a new shared key k and use it to blind the calculation $(\frac{z}{(uw)^x})^k = (\frac{Reenc(c)}{(Reenc(a)^x Reenc(b^r)^x)})^k$. If the credential is valid then the result of the computation should decrypt to 1. The decryption is carried out by the tallying authorities.
  - **Result Computation** The valid ballots are decrypted and aggregated.

The scheme works clearly in linear time, since deduplication is done using hashtables on o^r and each credential is validated 'internally'. A practical aspect of the scheme is that the credentials cannot be revoked since the registration authorities' keys are global and every change would affect all voters. A solution to this problem splits the credential in two parts: a private one which consists of r and a public one which consists of (a, b, c). Only the private portion should be communicated in an untappable way. This separation allows the authorities to recompute their registration keys in each election and forward them to the eligible voters each time. The main flaw of this line of work, is that the authorities can easily generate fake credentials that are identical to valid ones.

Finally, the approach of [77] has potentially another problem. The bulletin board is by design filled with fake votes to fool the coercion attempts. However this fact yields another attack vector. An entity wishing to disrupt an election can cause a denial of service attack, by deliberately injecting fake votes, causing a quadratic effect in the vote authorisation phase. A solution to this problem might be considered out of scope as far cryptographic election research is considered, as in all the literature reviewed the bulleting board is an always available structure. A solution however might affect the credentials. [85], who first described the problem, proposes the use of *dummy* credentials. These are anonymous tokens that are given to the voters during registration and aim to bound the total number of ballots. The approach relies on a 'smart' bulletin board that immediatelly rejects duplicate and fake votes using blind hashtables (note: that the tagging attack does not apply since the respective ballots dissapear and therefore cannot be tracked). As a result the only votes that are present in the bulletin board are the ones with valid and dummy credentials. Since both types are issued by the registration authorities, they present a bound to flooding of votes. Consideration must be given to the number of dummy credentials d per voter. If d is fixed, then the system *is not* coercion resistant as the attacker can force the voter to vote $d + 1$ times and rest assured that the voter has used his real credential. As a result a variable number of credentials per voter d_i is required. Unfortunately a percentage of voters are still vulnerable to coercion: the ones that receive the minimum and the maximum number of credentials. Their numbers must be kept to a minimum. In addition practical aspects of the credential selection remain unspecified.

5.3 Everlasting Privacy

The vast majority of the voting schemes presented so far in this thesis base their privacy on various computational assumptions like the difficulty of solving the DLOG problem. This difficulty is only temporary, however. In the worst case, future cryptographic advances might render the underlying problems easy. Even if such a development does not occur, advances in computing power might sometime render the schemes vulnerable to cracking (while in the theory the assumption holds). Adi Shamir is quoted in [96] saying that in 30 years' time all currently used keys will be insecure. This situation has very important consequences in voting. For example, a future totalitarian regime, might want to extract the political convictions of its subjects by examining their choices (or extrapolate based on the choices of relatives) in current democratic elections. This situation is made more difficult by the following facts: First of all, since votes might be considered public record, even in encrypted form, there might be legal obligation to hold them for some unforseeable amount of time. To make matters worse, bulletin boards for Internet voting schemes, cannot be permanently due to the nature of the internet. Finally, electronic storage nowadays is cheap and abundant. As a result the threats list above, are easy to implement. As a result voting systems require a stronger form of privacy than the one offered by computational assumption. They require *information theoretic* or *everlasting privacy*. On the other hand it is worth noticing that the integrity requirements of an election are ephemeral, required until the loser is convinced.

According to [48] the first one to consider voting with everlasting privacy was Bos, a student of Chaum. The first complete protocol to provide it explicitly is the [43] which we covered as a homomorphic scheme in 3.3.1. Its major drawback is that the work required by the voter depends on the number of authorities. Alternatively, everlasting privacy can be achieved as a byproduct of using a blind signature based scheme with anonymous channel. There, the bulletin board of the anonymous channel contains only anonymous information. Recent work in this area includes [96],[48] and [25]. The main theme permeating all these works is the use of the Pedersen commitment scheme 2.1.4.2 to unconditionally hide the vote. Recall that this scheme is computationally binding which does not matter, since there is no use in changing a vote after the elections are concluded (and no way to do it, for that matter). However there is a different and more important problem to be dealt with: how does one provide verifiability while maintaining (everlasting) privacy. More concretely, in order to verify the operations of the authorities ie. mixing and/or homomorhpic tallying one must open the commitments. But then the scheme does not provide privacy at all, let alone everlasting privacy. To fix this situation a compromise is made: The protocols provide everlasting privacy to the public but computational privacy to the authorities.

In more detail [48] incorporates everlasting privacy to [44] as follows:

- The voter commits to a candidate using the Pedersen scheme, namely by computing $commit(\mathfrak{c}, r) = g^r h^{\mathfrak{c}}$
- The voter encrypts $\mathfrak{c}, r$ using a homomorphic public key scheme such as Exponential El Gamal or Paillier by computing $Enc(\mathfrak{c}), Enc(r)$
- The ballot is a triple containing $commit(\mathfrak{c}, r), Enc(\mathfrak{c}), Enc(r))$ along with the required proofs:
  - The commitment and the encryption of the candidate hide the same $\mathfrak{c}$
  - The randomness used by the commitment and the second encryption is the same
  - $\mathfrak{c}$ is a valid candidate index (either 0 or 1)
  - There is a single selected candidate
- However only the commitment is placed on the public bulletin board
- The tallier multiplies the commitments and produces $commit(\sum \mathfrak{c}, \sum r)$
- The tallier receives the encryptions through a private channel. By multiplication he computes $Enc(\sum \mathfrak{c})$, and $Enc(\sum r)$, which are then decrypted in a threshold manner
- The decrypted values open the accumulated commitment thus realising universal verifiability

Everlasting privacy can be useful to mixnet based schemes as well [25]. In this case, the lifting of the computational assumption means that the processed messages will no longer remain anonymous as the input - output link will be revealed - the attacker will break the reencryption. The main idea is to replace the public key encryption scheme of the mixnet with a commitment scheme. That is, the voter does not merely encrypt the selection but instead she commits to it. If the commitment scheme is homomorphic like in the case of 2.1.4.2 the mixnet operates without any significant changes. The same problem arises however: In order to count the votes, the commitments must be opened. This requires that they are somehow sent to the tallying authorities. In order to enable privacy a private mixnet is proposed for the decommitment values. Again, everlasting privacy can be guaranteed only against the general public. More specifically:

- The voter against inputs a triple to the mixnet, $commit(\mathfrak{c}, r), Enc(\mathfrak{c}), Enc(r))$ along with the required proofs, as described previously. The only part of the input that is published is the commitment.
- Each mix server rerandomises the commitments and reencrypts the opening values and yields a random permutation of the triples.

- The new commitments are published to the bulletin board, while the rest of the values are sent to the next mix server using a private channel, while at the same time using techniques from 4
- After the last mix server has finished, the opening values are decrypted and the commitments are verified.
- When the elections finish, the authorities destroy all the private information that has passed through them.

The private channels between the mix servers can be implemented using the chaumian paradigm (decryption mixnet). Thus the system as a whole combines the work of [28] and [108]. A problem that arises in this scheme is that the first mix server gets to see the encryption of the voter choice $\mathfrak{c}$. If that is the case, then the private parts of each triple, can be split in two, mixed in parallel and recombined before decryption.

5.4 Coercion Resistance in a practical secret voting scheme for large scale elections

In this section, we propose a way to combine the voting scheme in [57] with the coercion resistance paradigm of [77], in order to provide coercion resistance to schemes based on blind signatures. As far as we know, there hasn't been any relevant attempts in the bibliography.

More specifically, we modify the system of [101] that provides the *vote and go* property to [57]. Our approach takes advantage of the function splitting architecture between the registration authority and the counter to reduce the quadratic complexity of the JCJ scheme. Consequently, we check for the validity of the voter credential during the blind signing of the ballot. This can be done in constant time as the voter identity is known. In our scheme the RA has access to the voter roll. As a result, it can PET the credential sent to it, with the one stored in the voter roll and find out if it is fake, meaning that the voter is under coercion.

After the check the authority will apply the blind signature scheme with a twist. It will use a different signature on a ballot with valid credential and a different one on a ballot with a fake credential. Of course, to satisfy coercion resistance both signatures must be indistinguishable to the coercer, that is verify with the public key of the authority. However the counter must be able to tell, whether the signature corresponds to a fake credential or not, in order to include the ballot in the tallying phase. To implement this functionality we turn to group signature schemes proposed by Heyst and Chaum in [32], with the counter playing the role of the manager. While many group signatures schemes have been proposed, we will built upon the simplicity of the work of [110]. Apart from these adjustments the scheme will follow the basic principles of [57] and [101].

Each of the participants in our scheme has a public and private key pair: The voters (V, v), the registration authority (RA, ra) and the counter (C, c). In more detail our scheme proceeds as follows:

- **Pre-Election Phase**
  - Each voter interacts with the registration authority using an untappable channel (for instance in person)
  - This is not prohibitive as the registration phase can be reused in multiple elections and as a result would occur rarely or only once.
  - The RA creates the voter credential σ and transfers it to the voter, following the JCJ restrictions.

- The RA creates and publishes the voter roll, that is a list of encrypted voter credentials $VR = \{E_{RA}(\sigma)\}$. In our version there is no use in announcing total number of votes, as there can by multiple votes per voter.

- **Authorisation Phase**
  - The voter prepares her vote and commits to it using the public key of the counter. That results in the creation of $E_C(vote)$ which will be called the ballot henceforth.
  - The ballot is blinded as $Bl(E_C(vote))$
  - The tuple $(ID, E_{RA}(\sigma), Bl(E_C(vote)))$ is sent to the authority signed with the voter public key. If the voter is under coercion, the voter will create a new fake anonymous credential, while in her moment of privacy the voter will supply the credential given during registration. This will be the only difference between a coerced and a non coerced vote.
  - Upon receipt the registration authority, validates the voters signature using the corresponding public key.
  - The RA checks using the identity information to check whether the voter has a right to vote, or if she has voted again.
  - If the previous check turns out ok, then the authority will sign the blinded ballot.
  - Before, however, it uses the identity of the voter to retrieve the encrypted credential from the voter roll.
  - Using the plaintext equivalence technique it compares the supplied credential with the retrieved one.
  - If the check is successful, that is both ciphertexts refer to the same plaintext, then the RA encrypts a validity token using the public key of the tallier, $EV = Enc_c(valid)$ else it creates an encryption of an invalidity token $EV = Enc_c(invalid)$
  - Subsequently, it signs the blinded ballot and submits the tuple $(sign_{RA}(Bl(E_C(vote))), EV, sign_{RA}(EV))$ to the voter

- **Voting Phase** The system continues as in 5.1 with the difference that EV is carried around as well.
  - That is the voter, unblinds the signed ballot to receive $sign_{RA}(E_C(vote))$
  - Both signatures are validated and then the voter proceeds to vote by submitting through an anonymous channel the tuple:
    $(E_C(vote), sign_{RA}(E_C(vote)), EV, sign_RA(EV), E_{RA}(\sigma))$

- **Tallying Phase** After the voting phase ends, counting can begin.
  - For each vote both signatures are validated.
  - Then the actual ballot is threshold decrypted, as well as the validity token. Only valid votes are counted.

Discussion One thing we did not deal with in the protocol above, is the handling of the duplicate votes, that are implied by the JCJ scheme. In our case, we must note that duplicates are dealt with **after** the validation procedure, instead of before as in the JCJ scheme. As a result they apply only to the valid credentials. One way to deal with such duplicates, is to have the registration authority implant a credential specific token of validity. As a result, valid votes by the same voter will bear the same credential. In this way the counter can identify if a valid vote is duplicate using a *O(1)* technique and then apply a *first vote* or *last vote counts* policy.

At first glance our protocol, seems to lack one important property: universal verifiability. To analyze this issue, we must note that the following actions must be verifiable:

1. The RA should only mark as valid the votes that correspond to valid credentials.
2. The counter should only use the validity tokens as provided by the RA.
3. The counter should only count the votes that correspond to valid tokens.

Of course universal verifiability should not make the scheme any less coercion resistant and vice versa. Firstly, since our scheme inherits from [101] it bases its verifiability on the threshold nature of the ballot decryption. If a vote is not correctly decrypted then one of the key shareholders might complain. A single counter cannot be trusted, but several talliers with conflicting interests might seem more trustworthy. The same applies for the registration authority.

Despite the above arguments, the verifiability of the protocol deserves further investigation.

Chapter 6

Case Studies and Implementations

6.1 Helios

6.1.1 Introduction

Helios is an open source voting system, with a web interface that enables remote voting. It was proposed and implemented by Ben Adida in [5]. Being a remote voting system, it cannot do away with coercion. As a result it is targeted to small elections, that can tolerate some levels (dubbed *low coercion elections*). It can also serve as a teaching tool for the general public to get to know electronic voting. It is currently in the third version. Its main characteristics are:

- Integrity. Helios supports unconditional integrity, which means that even if *everybody* is corrupted the tally will reflect the way users really voted.
- Voter Privacy. Ballot secrecy is conditional as it depends on the Helios server or on a set of trusted parties.
- Individual Verifiability. Each voter can check that her vote was correctly recorded.
- Universal Verifiability. Each voter can verify if the election result was calculated correctly.

As open source software, Helios can be downloaded and installed on local machines. Elections however, can be created and carried out in the cloud from `www.heliosvoting.org`. Helios includes a publicly accessible bulletin board where user initially post their identities and their encrypted ballots. The initial version uses a reencryption mixnet based on [84] implemented with El Gamal. Subsequent versions use homomorphic encryption ala 3.3.2.

The properties described above can be reinforced with the following audits from the elections shareholders :

- Verify that the encrypted ballot, reflects their actual choice, before casting.
- In addition, verify that the posted ballot, reflects their casted choice.
- Verify that the tallying procedure has been run correctly, ie. that the result corresponds to the contents of the bulletin board.
- Verify that there is no ballot stuffing, ie. that all the ballots in the bulletin board correspond to actual voters.

6.1.2 Operation

An administrator creates the election through a web interface. He sets the duration of the election and creates a voter list by providing email addresses. Each voter receives an email that provides per-election-credentials and a link to the web bulletin board. Voting can now begin.

An important characteristic of Helios is that ballot preparation and casting are distinct actions. Each user can construct a ballot as often as she wishes but is authenticated only when it is cast. This idea was first proposed by Benaloh [15]. As a result, everybody can construct a ballot irregardless of whether she is a voter, which makes individual verifiability accessible to everyone. Ballot preparation is concluded with ballot encryption and a commitment to the encrypted ballot in the form of a hash. After ballot preparation the user can choose to either audit or cast the ballot. In the audit phase Helios, proves that the hash corresponds to a correct encryption of the vote. In the casting phase, the voter is authenticated and the encrypted ballot is added to the bulletin board. Auditing and casting are mutually excluded, not allowing a voter to cast the audited ballot. In Helios 3, however following [51], audited ballots can be post the bulletin board, for verifiability purposes.

Voting is done inside a Web browser and implemented in Javascript using technologies and APIs such as jQuery, AJAX and LiveConnect to utilise the rich Java libraries for cryptography and number theory. The data exchange format is JSON. During vote preparation, no network connections are established. All processing takes place in the browser, implementing the concept of a voting booth. The voters can check this by setting their browsers to offline mode.

After ballot casting, the votes are sealed, meaning there can be no further changes. When the voting period has expired the election administrator initiates tallying, which is done either using a mixnet or homomorphically. In Helios v1.0, shuffling occurs, using 80 [84] secondary mixes. Thus the cheating probability is 2^{-80}. In the end, all the votes are decrypted and a tally is performed. The ballot processing occurs in the server side and is implement in Python. In the initial version no threshold decryption occurs, and as a result the server must be trusted for decryption. In Helios 2.0 [6] shuffling has been replaced by homomorphic tallying, in order to gain the efficiency that is required for larger scale elections. As a consequence the underlying cryptosystem has been changed to exponential El Gamal as described in chapter 2 in order to gain additive homomorphism. The ballot consists of:

- encryption of yes or no vote.
- disjunctive proof of knowledge of the contests of the vote like the one in 3.3.2
- proof of correct sum of all answers to a ballot's question

In order to increase the privacy of the system, multiple parties are required for decryption.

Helios shares a lot of similarities with a previous system named *ADDER* from [82]. Both systems are open source and implement essentially [44]. One difference is that the cryptographic operations in Helios are performed in Javascript which is embedded in all modern web browsers, while ADDER requires client-side software or Java applets. The most important difference however is that Helios allows users to audit the ballots, while in adder trust should be given to the voting software.

Attacks As a remote voting system, the security of Helios depends on the security of the client computer. Consequently, it is vulnerable to malicious software and exploits, such as buffer overflows, not only regarding its own client code, but that of other extensions. In [51] an attack is mounted that utilises buffer overflow vulnerabilities in Acrobat Reader. In the attack, the Helios Javascript client code is replaced by malicious code that *alters* the voter's choice to a candidate of the attacker's choice. Since the voter can audit the ballot, the malicious code interferes with the auditing and affirms the initial choice in spite of the actual vote being different. Of course, there is nothing to prevent the attacker, since he controls the client, to retrieve the user id of the voter, thus breaking privacy as well. The integrity attack can be revealed by auditing the bulletin board that contains the altered vote, however itis quite probable that it can go unnoticed.

Helios with mixnets The departure of Helios from the mix net model with version 2, was mainly for efficiency and for the support of votes with different weights according to voter type [6]. However the mix net model has some desirable properties, for instance the support of many social choice function such as STV voting, since the votes are decrypted. As a result there have been some efforts to reinstate mix net back to the Helios system. For example the *Zeus* system, [61] a variation of Helios with 128 [84] mix nets, has been implemented and tested in the elections for the Greek Universities' Governing Councils. After voting, a mixing phase occurs by the Zeus server and optionally external agents. Decryption is partially done by trusted parties and combined to produce the final unencrypted and anonymous ballots.

Everlasting privacy In section 5.3 we stressed an approach from [48] to provide everlasting privacy to homomorphic based system. The idea is to replace vote encryptions with perfectly hiding commitments. These commitments will be placed in the bulletin board. The opening values will be sent to the tallier using a compatible homomorphic encryption scheme. Both values will be combined and the aggregate commitments will open the aggregate result. Everlasting privacy is provided towards the bulletin board data, but not towards the opening data. This approach applied to Helios (with shuffling or homomorphic tallying), provides everlasting privacy towards outside parties, but not towards the Helios server.

6.2 Prêt à Voter

6.2.1 Introduction

Prêt à Voter is an end-to-end verifiable voting system and a design of ballots for electronic voting. It was created by Peter Ryan in 2004 ([34],[118]), however a similar idea can be found earlier in [45]. Its aim was to improve a previous cryptographic scheme by Chaum [31]. As it was successful on this task, it also influenced a subsequent voting scheme by Chaum. It is still in active development. Its characteristics are:

- Integrity
- Voter Privacy
- Receipt Freeness
- Voter Verifiability
- Voter Friendly
- Modular: Can implement many cryptographic voting protocols (encryption, zero knowledge etc.)
- Versatile: Can be used with many social choice functions

In order to be deployed certain assumption must be met:

- A voting infrastructure must be present, which means Voter registration and authentication is outside the scope of the system
- The ballot forms are properly managed (remain confidential, cannot be forged) between their creation and use
- Voting occurs in a voting booth with the traditional privacy characteristics
- There exists a tamper - free public bulletin board that can be used for verification

The most important concept in Prêt à Voter is the ballot, which is key to the operation. The main idea of the system is that: **The vote is recorded on a different frame of reference for each ballot**.As a result the system **encrypts the ballot not the vote**.

Prêt à Voter is an effort to achieve verifiable reflections with receipts. As mentioned before, cryptographic voting with receipts assures voters that their vote is being counted, while maintaining ballot secrecy and avoiding coercion. It seems as an uttainable goal but it has clear advantages as it

- Mathematically ensures election integrity
- Allow software independence [115] with the known advantages:
  - Avoid trust on proprietary hardware
  - Detect errors early (at the polling station)
  - Rerun the elections without software

The process of voting with receipts in general but in Prêt à Voter as well can be described by the following steps:

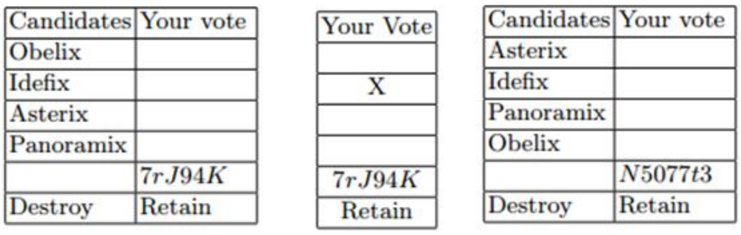
1. The voter enters the choices on the touch screen of the voting machine
2. A printer prints the receipt and issues a challenge
3. A voter checks the printout
4. Accept or Change Vote and answers the challenge
5. A mechanism for ballot secrecy is used
6. The receipt is printed
7. A mechanism for receipt secrecy comes into play in order to avoid coercion attacks.
8. A vote anonymisation technique might be applied.
9. Tallying takes place.
10. For universal and individual verification a web bulletin board exists. The voter enter some information about the receipt (e.g. receipt SN) and verifies that it is included in the tally.

One mechanism to apply to create the voter receipts is *visual cryptography*, as described in chapter 2. The system shown [31] might be considered the father of Prêt à Voter .

6.2.2 Ballot Form and Processing

As we mentioned before, the main idea of Prêt à Voter is that the vote is recorded on a different frame of reference for each ballot. For secrecy the system encrypts the ballot not the vote. The ballot consists of two columns: one for the candidate list and one for the voter selection. The candidate list is randomly ordered and the order is different for each ballot. The voter votes by marking or ranking the candidates on his/her ballot. It is very important that by exiting the voting booth the candidate list **be** disposed. The actual vote is the mark or rank that is recorded on the vote column. No encryption is needed, since the vote is useless without the candidate list and the vote column can be used as a receipt for voter verifiability. Another advantage of this ballot form is that since the vote order is different, a source of voter bias is removed but compliance might be affected. An extra bonus is that as we shall see side channel attacks are side stepped.

To be more specific, in the initial version of Prêt à Voter [34], there is a base ordering of a candidates generated by the election authority that creates the ballots. The actual ballots are produced by printing an ordering that is a transposition of the base by a particular offset. This offset is encrypted and printed on each ballot. This encrypted value is called onion, as in [34], it is processed by a classic Chaumian mixnet, as described in chapter 4. The voting equipment records the index/ranking and transmits it along with the encrypted onion. If the candidate list is destroyed only the voter knows the order of the candidates that he voted on. To sum up, no encryption takes place at the booth (the onion is created in advance) and the vote without the candidates is useless.



Candidates	Your vote
Obelix	
Idefix	
Asterix	
Panoramix	
	$7rJ94K$
Destroy	Retain

Your Vote
X
$7rJ94K$
Retain

Candidates	Your vote
Asterix	
Idefix	
Panoramix	
Obelix	
	$N5077t3$
Destroy	Retain

FIGURE 6.1: Prêt à Voter Ballot [34]

After this initial phase, the (vote,onion) pair is anonymised using a mixnet. Verification of the mixnet is done using RPC. After decryption, tallying takes place.

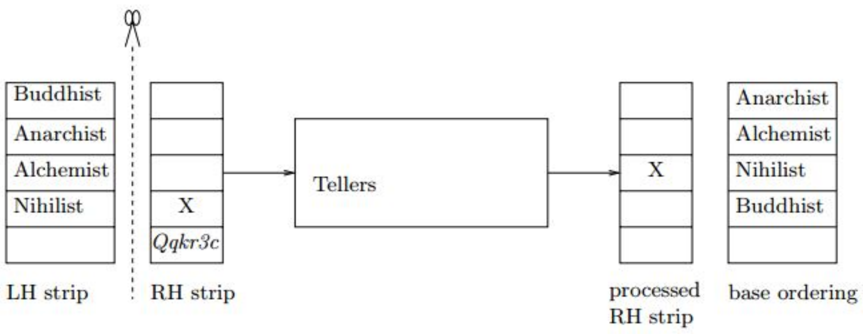


FIGURE 6.2: Prêt à Voter Vote Processing [34]

More specifically, the system parameters are n voters, m candidates, k mix servers (also called tellers in Prêt à Voter) and $2k$ key pairs as each each mix server performs 2 mixes (a different perspective on RPC than the one presented in chapter 3.

The ballot generation is layered:

- $2k$ random values $\{g_{i,j}\}_{0,0}^{n-1,2k-1}$ - germs - are generated for each voter.
- The germs will be used for the randomised encryption of the ballot.
- The onion for ballot i: $D_{i,2k} = Enc(g_{i,2k-1}, Enc(\cdots Enc(g_{i1}, Enc(g_{i0}, D_{i0})), \cdots))$ where D_{i0} random value for initialisation
- The **offset** is calculated as $\theta_i = \sum_{j=0}^{2k-1} d_j \quad mod \quad m$ where $d_j = hash(g_{ij})$

The vote i is a pair $(r_i, D_{i,2k})$ where r_i is the index of the marked candidate (in the simple voting case).

The vote is casted and the mixing takes place:

- Input $L_{2j} = \{r_i, D_{i,2j}\}_{i=0,j=k-1}^{n-1,0}$
- Decrypt the onion with the private key (g_ij, D_{ij})
- Apply the hash to the germ value of the onion $d_{ij} = hash(g_{ij})$
- Calculate new shift $r_{ij} = r_{i,j-1} - d_{ij}$
- Sort the list by the value of the onions
- Audit with RPC

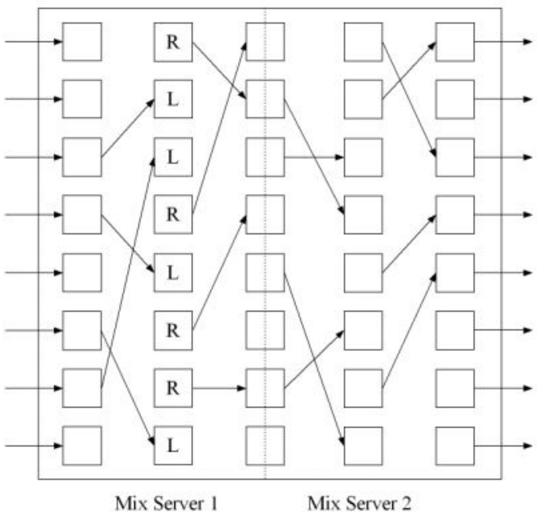


FIGURE 6.3: Auditing with RPC [34]

As mixing and auditing (RPC) is cryptosystem-agnostic the procedure above can be applied for reencryption based mix nets. [116].

Prêt à Voter has many advantages but is vulnerable to the following threats:

- Confidentiality of ballots, before elections. There is a need to trust the authority and the intermediates
- Chain Voting: Coercion by smuggling a ballot, pre-voting, and require a replacement ballot
- Kleptographic Channels: Choose special values of cryptographic variables in order to leak information to a colluding party, like the tagging attack in mixnets.

6.2.3 Distributed ballot generation

The processing as described above, places too much trust in a single authority. In order to surpass the need for trusting a single authority distributed generation of ballots. It is interesting that the same mechanism for used for vote processing can be used for this process [116].

In the distributed generation of ballots 2 encrypted *'onions'* will be generated. The first $onion_L$ will be used to produce the random candidate list and will be read (decrypted) by the voting machine in order to display the candidates to the voter. The second $onion_R$ will be used for voter verifiability and reconstruction of the candidate list during tallying. The ballot forms will be generated by l entities called *clerks*. Each clerk contributes to the generation of a common encrypted seed. The candidate list is constructed from the seed. To reveal the seed values all clerks must be corrupted.

Let p, q safe primes, α, γ generators of $\mathbb{Z}_q^*$, β_T the public key of the tellers and β_R the public key of the vote recording machines.

Clerk C_0 generates a batch of initial seeds s_i^0 - one for each ballot - and creates voting machine $E_R(\gamma^{-s_i^0}, x_i^0) = (\alpha^{x_i^0}, \beta_R^{x_i^0} \cdot \gamma^{-s_i^0})$ and teller $E_T(\gamma^{-s_i^0}, y_i^0) = (\alpha^{y_i^0}, \beta_T^{y_i^0} \cdot \gamma^{-s_i^0})$.

Clerk C_l receives batch of onions from previous clerk, generates fresh randomness and perform reencryption and shuffle. The final onions will look like:

- $onion_L = E_R(\gamma^{-\sum_i s_i}, \sum_i x_i)$
- $onion_R = E_T(\gamma^{-\sum_i s_i}, \sum_i y_i)$

If all the above operated correctly operation then the 'plaintexts' match.

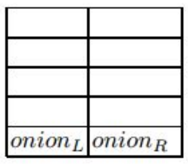


FIGURE 6.4: Distributed ballot generation in Prêt à Voter [116]

Voting with distributed ballots The onions are stored in encrypted form and are presented to the voter on the ballot form. The voting booth device reads and decrypts the left hand onion s is revealed. The candidate permutation is reconstructed as a function of s. The device should not see the right onion. The voter now faces a classical Prêt à Voter ballot. She marks the candidate index and embed the selected candidate index into seed by reencryption $vote = E_T(\gamma^{r-\sum_i s_i}, \sum_i y_i)$. The mixing proceeds as usual. After decryption,the plaintexts are of the form $\gamma^{r-\sum_i s_i}$. In order to reveal the tally a discrete logarithm must be solved. In order for this to be easy the seeds must be selected so that the exponent is not too large and the search space is bounded. Alternatively the Paillier Cryptosystem [117] can be used.

6.2.4 Alternate social choice functions

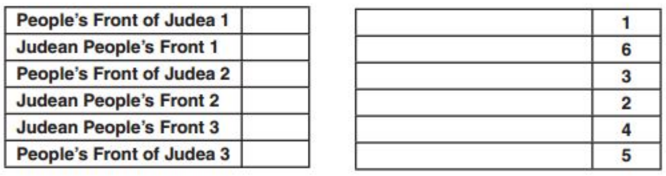
6.2.4.1 Singe Transferable Vote

In [70] an effort was presented to incorporating STV into Prêt à Voter . STV as described in chapter 2 has some useful characteristic that were helpful in the implementation:

- **Unused Preferences:** Some of the preferences might not be used, since the winner might get declared earlier.

- **Transfer history** is irrelevant. We are not interested for the previous candidates since they are eliminated
- **Lazy evaluation** can be applied. We can work on the candidates one by one.

However since tallying in Prêt à Voter uses a mix net based approach, the ballots will be decrypted. Thus the italian attack can be applied. More specifically, the cyclic shift proposed in [34] cannot be used to encode a partial ranking as it might leak too much information, because it preserves the relations found in the canonical ordering, as depicted in the figure below:



People's Front of Judea 1	
Judean People's Front 1	
People's Front of Judea 2	
Judean People's Front 2	
Judean People's Front 3	
People's Front of Judea 3	

	1
	6
	3
	2
	4
	5

FIGURE 6.5: A permutation that might leak information [70]

As a result a random but complete permutation, must be applied in each ballot. This cannot be done using El Gamal and reencryption. It can be done however using RSA onions, where each layer in the onion encodes a permutation that can be recovered and applied. However this approach has problems as well since:

- It is evident how many choices were made
- This can be used to partition ballots and leak information
- No random padding can be added, since it will affect the election results
- The Italian attack is still applicable.

The solution found by [70] is *multi valued reencryption mixes*. The vote is encoded as a sequence of cryptograms $\{c_1, \cdots, c_k\}$ that decrypt to a candidate index. The order of the cryptograms reflects the ranking of the candidates by the voter, ie c_1 is the encryption of the index of the favourite candidate. A dummy candidate 'STOP' id included that marks the end of the user ranking. A complete ranking is given by appending a random permutation of the remaining candidates.

The tallying is done in a *lazy* manner. First the heads of the candidate list are decrypted and the first preferences are counted. If a candidate is eliminated then the head is appended at the end, an anonymising mixnet is run and the heads are again decrypted. The process stops when a winner (a candidate with more than half the votes) is found.

6.2.4.2 Condorcet Voting

In [39] the Prêt à Voter ballots are used to implement Condorcet voting. For c candidates we have $\frac{c(c-1)}{2} \dashrightarrow O(c^2)$ ballots. Each ballot is a yes/no choice for candidates i, j. A yes choice indicates preference for candidate i over candidate j. Each yes/no ballot has its own onion $< i, j >$ which is contructed in a layered manner for use in decryption mixnet. Each voter submit $c \times c$ votes with the preferences and the onions. The vote is run through a decryption mixnet for anonymisation. It must be noted that there can be no correlation that all c^2 votes belong to the same voter. A summation matrix is used to declare the winner.

6.3 Proof of Concept Implementations

To get an actual feel of the protocols discussed in 3 and 4 we implemented two of them. More specifically, we built a homomorphic voting system based on the Damgard Jurik cryptosystem (2.1.1.3,3.4), that allows selecting for 1 out of C candidates. To be exact we implemented the system described in [46]. Henceforth, this implementation will be referred to as 'DJN'. In addition, we implemented voting with mixnets based on permutation networks (4.3.2), as a combination of [92] and [91]. Henceforth, this implementation will be referred to as 'MixPerm'. Before we delve into the details of each system, let's describe some common characteristics of both:

- **Dynamic and multiple elections**. Both systems allow for an administrator to create elections with multiple answers of his choice. The voter selects the desired election-question from a list and chooses the desired option.
- **Bulletin Board as a database**. Borrowing from Helios, we represent the bulletin board as a relational database. The database contains the election options, the cryptographic parameters as well as the encrypted votes and the proofs of shuffle and validity. For simplicity we chose the SQLLite (`http://sqlite.org/`) database engine.
- **Architecture** We follow a modular architecture for both implementations. The user interface is split from the database handling code and from the protocol implementation. The cryptographic functions and the voting protocols for both systems are implemented in a class library, named *CryptoLib* which is compiled as a Dynamic Link Library. It is the only one that does not have database access. Regarding the other components of the system, the *Election Creator* module initiates the election procedure by creating the voter choices and the cryptographic keys and stores everything in the database. Another component is the *Voting Client*, which allows the voter to select an election, retrieve the options and cast the vote. The third executable component differs per system and implements the main part of the protocol, which is the homomorphic tallying and mixing functions respectively, along with the proof validation.

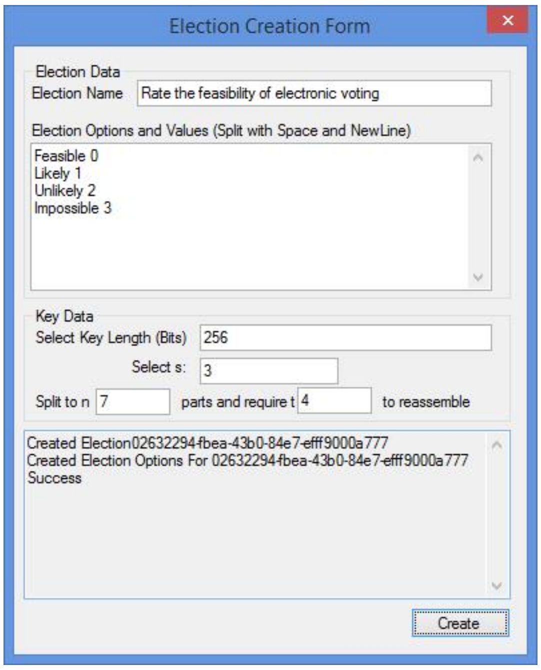
For the implementation we used a combination of .Net (`http://www.microsoft.com/net`) languages with Visual Studio 2013. This choice was made on grounds of familiarity. The user interface was implemented in C#, while the protocol as well as the cryptographic details were implemented in F#. We chose a functional programming language for the important part of the implementation, because of the expressiveness, simplicity and composability properties.

The code for both projects is available on Github.

6.3.1 The 'DJN' implementation

Functionality The election is created by an administrator by supplying the election issue and the election options. Each option is a label-value pair. The label can be any alphanumeric string, while the values must be continuous positive integers starting from zero (due to the nature of the cryptosystem and counters used). In addition the administrator selects the key size - which represents the bit length of each of the primes p, q (*not of the product* n) - and the generalisation factor of the 2.1.1.3 cryptosystem. Finally, the administrator selects the threshold of the cryptosystem, namely the total number of shares and the required number for reconstruction.

As a result, the election options (name, labels, values), the public key and the cryptosystem parameters are stored in the database. To illustrate the difference, the secret key shares are stored in text files. In a real election, these might be stored in smart cards or other devices.



Election Data
Election Name: Rate the feasibility of electronic voting

Election Options and Values (Split with Space and NewLine)

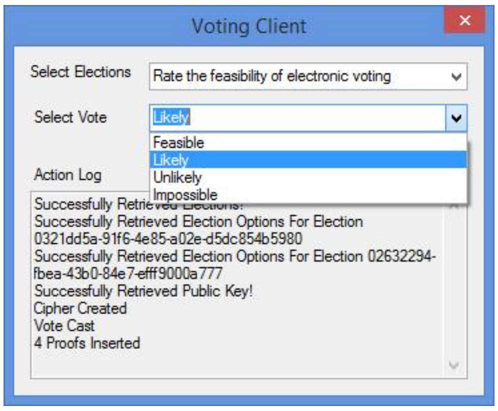
```
Feasible 0
Likely 1
Unlikely 2
Impossible 3
```

Key Data
Select Key Length (Bits): 256
Select s: 3
Split to n 7 parts and require t 4 to reassemble

```
Created Election02632294-fbea-43b0-84e7-efff9000a777
Created Election Options For 02632294-fbea-43b0-84e7-efff9000a777
Success
```

FIGURE 6.6: Election Creation Form in 'DJN' Project

Now voting can begin. To this end, the voting client will be used. The client initially retrieves all the available elections from the database and presents them to the voter. The voter selects the desired election, the program retrieves the corresponding choices and populates a dropdown list. The voter now can select the desired option and cast the vote. The system retrieves the public key from the database and encrypts M^{value}. In particular, the value *103* has been used as the basis M of the Baudron counter. Afterwards it generates proof of vote validity, namely that the ciphertext is an encryption of one of the available choices. Of course the actual proof depends on the number of choices. For example if there are 4 available choices, a *1 out of 4* proof is generated, and if there are five choices a *1 out of 5* proof is generated. The ciphertext and all the proofs are inserted in the database.



Select Elections: Rate the feasibility of electronic voting
Select Vote: Likely

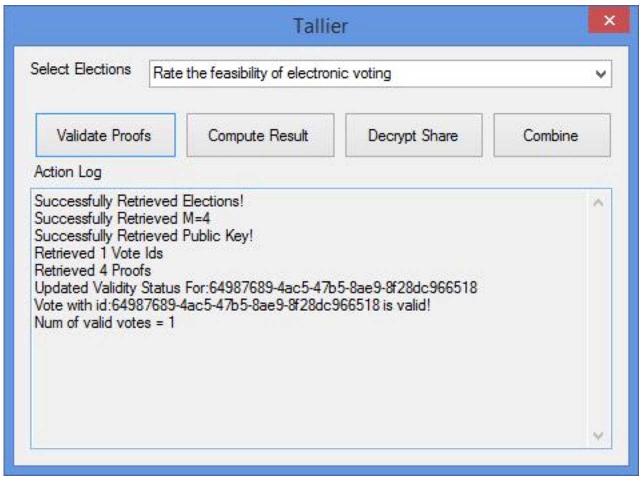
Action Log

```
Successfully Retrieved Elections!
Successfully Retrieved Election Options For Election
0321dd5a-91f6-4e85-a02e-d5dc854b5980
Successfully Retrieved Election Options For Election 02632294-
fbea-43b0-84e7-efff9000a777
Successfully Retrieved Public Key!
Cipher Created
Vote Cast
4 Proofs Inserted
```

FIGURE 6.7: Voting Client Form in 'DJN' Project

After everybody has voted, the result is computed. This is done, using the tallier program, which will be operated by an administrator. The user initially selects the election and then follows the workflow below:

- Validates the proofs for each vote.
- Aggregates the votes of valid proofs by using multiplication. The product is stored in the database.



```
Successfully Retrieved Elections!
Successfully Retrieved M=4
Successfully Retrieved Public Key!
Retrieved 1 Vote Ids
Retrieved 4 Proofs
Updated Validity Status For:64987689-4ac5-47b5-8ae9-8f28dc966518
Vote with id:64987689-4ac5-47b5-8ae9-8f28dc966518 is valid!
Num of valid votes = 1
```

FIGURE 6.8: Tallier Form in 'DJN' Project

- For the specified threshold value, selects each key share and 'decrypts' the result. Each decrypted share of the result is stored in the database.
- Combines the shares to create the result.

Architecture The general description of the previous section applies. Here we give more details. The database consists of the tables depicted in the following diagram:

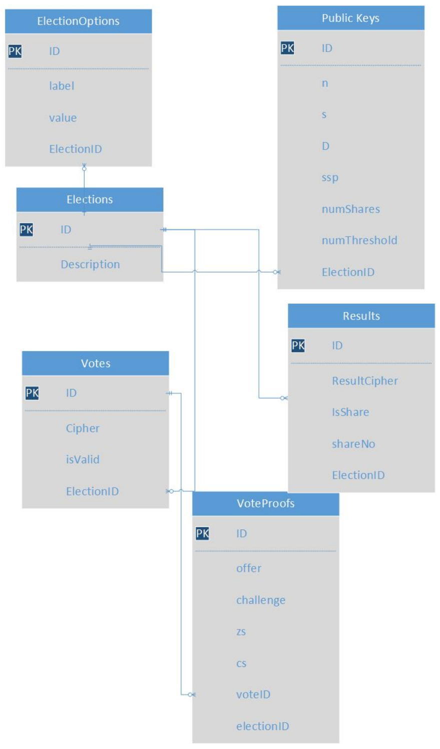


FIGURE 6.9: Database scheme for 'DJN' Project

Specifically, the table *Elections* contains the issue being voted for. Its primary key spans the complete database and allows for multiple elections. The options are stored in the table *ElectionOptions*, while the public key n, the public parameter s and the various values used for the threshold scheme namely $(D, ssp, numShares, numThreshold)$ are stored in the table *PublicKeys*. The actual votes goto *Votes*. Initially only the *Cipher* field is populated, containing the encryption of the vote. For each vote, the proofs of validity are entered in the table *VoteProofs* linked with the field *voteID*. When the votes are validated, the field *isValid* in the table *Votes* is completed. After aggregation the table *Result* is populated. The field *isShare* indicates whether the data corresponds to a decrypted share or to the result.

Code Due to space restrictions, we shall present as an example, the module that implements the essence of the election protocol.

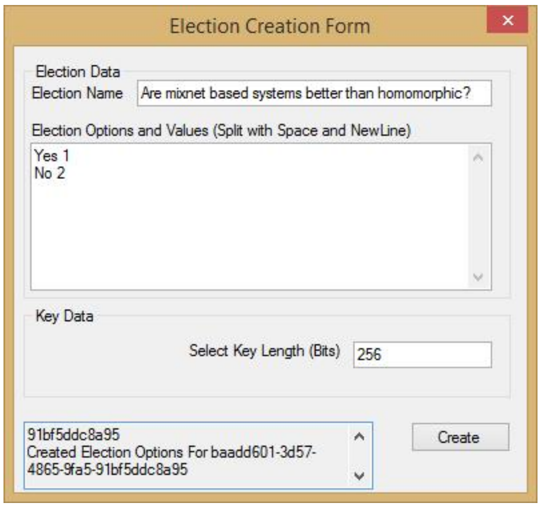
```
namespace CryptoLib

module DJN =

    open CryptoLib.Common
    open CryptoLib.Paillier

    type DJN(numCands) =
        let nc = numCands
        let M=103I

        let createWitnesses encVote n ns =
            let ns1 = ns * n
            let mutable arr = Array.init nc (fun _ -> 0I)
            for c = 0 to nc-1 do
                let b =  modulo (bigint.Pow(M,c)) ns
                arr.[c] <- modulo (encVote *invmod(powmod((n+1I),b
   ,ns1),ns1)) ns1
            arr

        let swap (arr:bigint[],a:int, b:int) =
            let mutable t=arr.[b]
            arr.[b]<-arr.[a]
            arr.[a]<-t
            let ls = arr|>Array.toList
            ls

        let extractRes res=
            let max = int(System.Math.Log(float res,float M))+1
            let mutable num = res
            let mutable votes = Array.zeroCreate max
            let mutable i = 0
            while num > 0I do
                let v = num%M
                votes.[i]<-v
                num<-num/M
                i<-i+1
            votes

        member d.vote choice pk =
            let enc = new Paillier()
            let n,s,_,_ = pk
            let encVote = enc.Encrypt(choice,pk,s)
            encVote

        member d.aggregate votes pk =
            let enc = new Paillier()
            let n,s,_,_ = pk
            let encRes = votes|>Seq.skip 1
                              |>Seq.fold (fun c1 c2 -> enc.Mult(c1
    ,c2,pk,s)) (votes|>Seq.head)
            encRes

        member d.mvote choice pk =
            let enc = new Paillier()
            let n,s,_,_ = pk
            let ns = bigint.Pow(n,s)
            let ns1 = ns*n

            let ballot =  modulo (bigint.Pow(M,choice)) ns
            let r = randomGCD(n)
            let encVote = enc.Encrypt(ballot,pk,s,r)
            let mutable arr = createWitnesses encVote n ns

            let ls = swap(arr,choice,0)
            let proof = enc.ProveNsPowerWID(ls,r,pk)

            let (offers, challenge, zs, cs)=proof
            let loffers = swap(offers,0,choice)
            let lzs = swap(zs,0,choice)
            let lcs = swap(cs,0,choice)

            encVote,(loffers|>List.toArray, challenge, lzs|>List.
    toArray, lcs|>List.toArray)

        member d.checkProof encVote proof pk =
            let enc = new Paillier()
            let n,s,_,_ = pk
            let ns = bigint.Pow(n,s)
            let (offers, challenge, zs, cs)=proof
            let ls = createWitnesses encVote n ns
            enc.ValidateNsPowerWID(ls,offers, challenge, zs, cs,
    pk)

        member d.decrypt cres pk sk =
            let dec = new Paillier()
            let res = dec.Decrypt(cres,sk,pk)
            extractRes res
        static member decryptShare cres pk sk =
            ThresPaillier.DecryptShare(cres,sk,pk)

        member d.combine dshares pk =
            let tp = new ThresPaillier()
            let x = tp.CombineShares(dshares,pk)
            extractRes x
```

Upon instantiation the code is fed with the number of candidates, which is an essential piece of information, as it affects the actual number of proofs that will be generated. The Baudron homomorphic counter is instantiated to 103 (a demo value). During voting, the function *mvote* is called with arguments the public key of the cryptosystem and the voter choice. The function encrypts M^{choice} and fills an array with random values for the rest of the choices. To create the witnessess, it divides each ciphertext (or random value) with the value that would be the choice (in function *createWitnessess*). Subsequently, the proof of validity is generated and returned for database insertion along with the ciphertext. In the proof generating function, it is assumed that the valid vote is contained in the first entry, and for the rest a random value is contained. To achieve this, the generated values will be swapped. Decryption of the result takes place using the functions *decryptShare*, *combine*, and *extractRes*. The first one decrypts the ciphertext using the share of the key *sk*. The decrypted values are then combined and the result is extracted using the algorithm of 2.1 implemented in the last of the above functions.

The complete and current code for this project can be obtained from the Github repository at: `https://github.com/pgrontas/Homomorphic-Voting-DJN`

6.3.2 The 'MixPerm' implementation

Functionality Again the election begins using the *Election Creator* module. This time the underlying cryptosystem is ElGamal. The election options are inserted into the database as well as the public key. Following [91] only yes/no voting is supported.



Election Data
Election Name: Are mixnet based systems better than homomorphic?

Election Options and Values (Split with Space and NewLine)

```
Yes 1
No 2
```

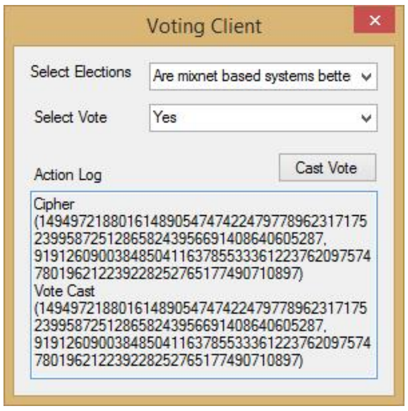
Key Data
Select Key Length (Bits): 256

```
91bf5ddc8a95
Created Election Options For baadd601-3d57-
4865-9fa5-91bf5ddc8a95
```

FIGURE 6.10: Election Creation Program For 'MixPerm' System

The voting client functionality is similar to the corresponding module in 'DJN'. The system retrieves the available elections, the voter selects one and it s parameters are fetched. The ballot cast is an ElGamal ciphertext. As the votes are to be fully decrypted, no proof of vote validity is needed.

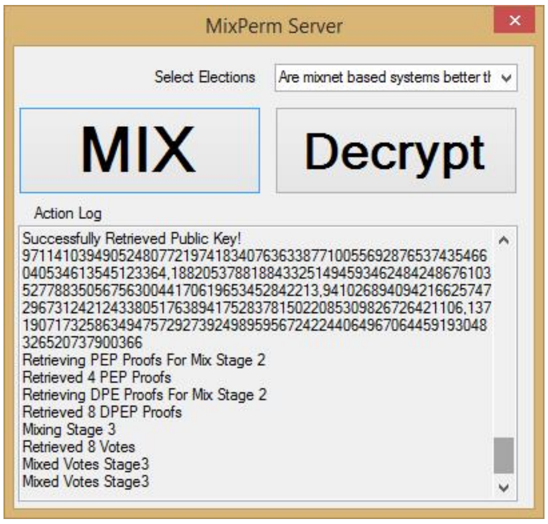
Mixing, decryption and counting is done using the MixServer module. Initially the votes are retrieved from the corresponding database table and placed in the table *Mix*, for processing. Since [91] supports



```
Cipher
(14949721880161489054747422479778962317175
239958725128658243956691408640605287,
919126090038485041163785533361223762097574
780196212239228252765177490710897)
Vote Cast
(14949721880161489054747422479778962317175
239958725128658243956691408640605287,
919126090038485041163785533361223762097574
780196212239228252765177490710897)
```

FIGURE 6.11: Voting Client For 'MixPerm' System

number of ballots that is a power two, to realise the functionality we add enough 'dummy votes' to reach the next such power. These votes are encryption of invalid choice and will not be counted during the final tally. During each shuffle, the votes are retrieved from there, get reencrypted and then shuffled using a recursive sorting network. Before this processing the proofs of the previous mix are checked and if they are found invalid the operation stops.



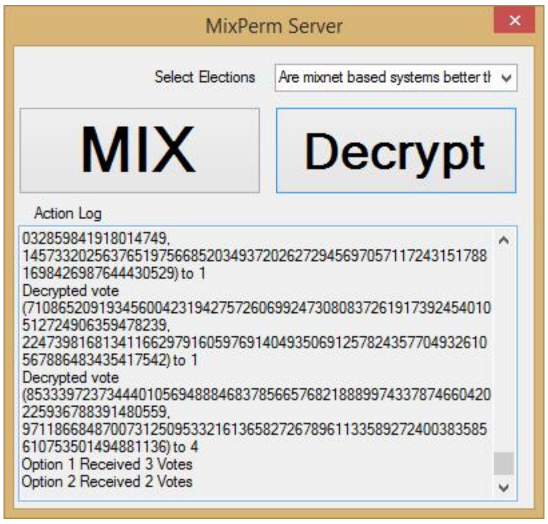
```
Successfully Retrieved Public Key!
9711410394905248077219741834076363387710055692876537435466
040534613545123364,1882053788188433251494593462484248676103
5277883505675630044170619653452842213,941026894094216625747
2967312421243380517638941752837815022085309826726421106,137
1907173258634947572927392498959567242244064967064459193048
326520737900366
Retrieving PEP Proofs For Mix Stage 2
Retrieved 4 PEP Proofs
Retrieving DPE Proofs For Mix Stage 2
Retrieved 8 DPEP Proofs
Mixing Stage 3
Retrieved 8 Votes
Mixed Votes Stage3
Mixed Votes Stage3
```

FIGURE 6.12: Vote Mixing in 'MixPerm' System

After each mix server has taken its turn, the votes are decrypted and the result is calculated.

Architecture The database of the system bares many similarities with that of 'DJN'. We shall focus here on the differences, omitting the ones that are caused by the use of the ElGamal cryptosystem. The *Mix* table is empty at first. When mixing begins all the encrypted votes are copied and retrieved by the first mix server. After operation, the mix server increments the *stage* fields and reinserts the results. The table *MixProofs* contains links to the proofs that were generated during mixing. For each switching gate in the permutation network 2 proofs are generated. They are stored in the table *SchnorrProofs*.

Code Due to space restrictions, we shall present as an example, the module that implements the mixing part of the election protocol. The entry point of this module is the *mix* function which takes a list of ciphertexts and an El Gamal public key and merely forwards them to the *permnet* function. The latter is a recursive function that splits the list in half, calls itself and then merges the result. In the base case, the switching gate of the permutation network is implemented, using a call to the *switchingGate*



```
032859841918014749,
1457332025637651975668520349372026272945697057117243151788
1698426987644430529) to 1
Decrypted vote
(7108652091934560042319427572606992473080837261917392454010
512724906359478239,
2247398168134116629791605976914049350691257824357704932610
567886483435417542) to 1
Decrypted vote
(8533397237344401056948884683785665768218889974337874660420
225936788391480559,
9711866848700731250953321613658272678961133589272400383585
610753501494881136) to 4
Option 1 Received 3 Votes
Option 2 Received 2 Votes
```

FIGURE 6.13: Vote Counting in 'MixPerm' System

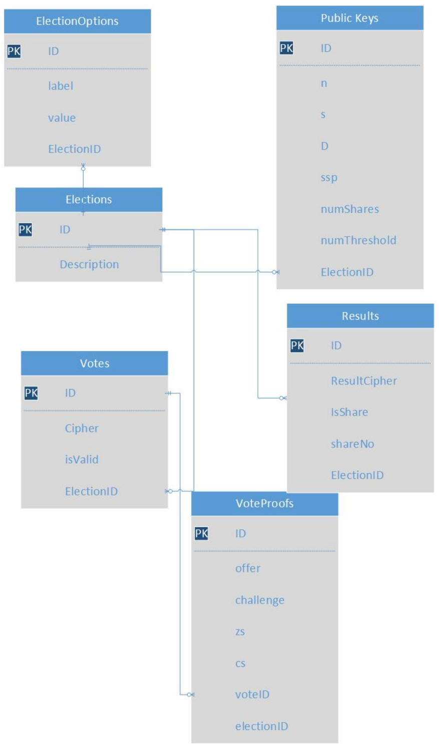


FIGURE 6.14: Database scheme for 'MixPerm' Project

function. It takes two ciphertexts and the public key. At first it reencrypts them using the public key and then using a random bit permutes them. It also generates a proof that each ciphertext is correctly reencrypted *DISPEProof* and a proof that the product of the outputs is a reencryption of the product of the inputs, according to [91]. The proofs are returned to the calling module and inserted to the database.

```
namespace CryptoLib

module PermNet =

    open CryptoLib.Common
    open CryptoLib.ElGamal
    open CryptoLib.Schnorr
```

```
 let splitList list = List.foldBack (fun x (l,r) -> x::r, l)
list ([],[])

 //Prove that ciphertexts c1 and c2 encrypt the same plaintext
under public key pk
 //The prover must know the reencryption factor r
 let genBlindedKey c1 c2 pk =
     let ((g,p,q),y) = pk
     let (g1,m1) = c1
     let (g2,m2) = c2
     let z1 = getRandom(q)
     let z2 = getRandom(q)
     let G = modulo (powmod(y,z2,p)*powmod(g,z1,p)) p
     let Y = modulo (powmod(g2*invmod(g1,p),z1,p)*powmod(m2*
invmod(m1,p),z2,p)) p
     (G,Y)

 let PEProof c1 c2 pk r =
     let ((g,p,q),y) = pk
     let (G,Y) = genBlindedKey c1 c2 pk
     let PK = ((G,p,q),Y)
     let SK = ((G,p,q),r)
     let sn = new Schnorr()
     sn.prove(SK,PK)

 //Disjunctive Plaintext Equivalence Proof
 //Prove that c is reencrypted either as rc1 or as rc2 with
reencryption factor r
 let DISPEProof c rc1 rc2 pk r =
     let ((g,p,q),y) = pk
     let snw = new SchnorrWID()
     let (G1,Y1) = genBlindedKey c rc1 pk
     let (G2,Y2) = genBlindedKey c rc2 pk
     let PK1 = ((G1,p,q),Y1)
     let PK2 = ((G2,p,q),Y2)
     let SK = ((G1,p,q),r)
     snw.prove(SK,[PK1;PK2])

 let switchingGate c1 c2 pk =
     let ((_,_,q),_) = pk
     let eg = new ElGamal()
     let r1 = getRandom(q)
     let r2 = getRandom(q)
     let rc1 = eg.ReEncrypt(c1,pk,r1)
     let rc2 = eg.ReEncrypt(c2,pk,r2)
     let dpep = DISPEProof c1 rc1 rc2 pk r1
```

```
        let c = eg.mult(c1,c2,pk)
        let rc = eg.mult(rc1,rc2,pk)
        let pep = PEProof c rc pk (r1+r2)

        let b = getRandom(bigint (System.Int32.MaxValue))
        if b%2I = 0I then
            (rc1,rc2), (pep,dpep)
        else
            (rc2,rc1), (pep,dpep)

    let rec permNet(ls, pk)=
        let n = ls|>List.length
        if n = 2 then
            let (c1,c2),(p1,p2) = switchingGate ls.[0] ls.[1] pk
            [c1;c2],[(p1,p2)]
        else
            let (ls1,ls2) = splitList ls
            let ls1', proofs1 = permNet(ls1,pk)
            let ls2', proofs2 = permNet(ls2,pk)
            ls1'@ls2', proofs1@proofs2

    let mix(ls,pk) =
       let xs = List.ofSeq(ls)
       let ls1,ls2 = permNet(xs, pk)
       Seq.ofList(ls1),Seq.ofList(ls2)

    let verify(xs) =
        let proofs = List.ofSeq(xs)
        let v1 = new Schnorr()
        let v2 = new SchnorrWID()
        proofs|>List.map (fun (p1,p2) -> v1.verify(p1) && v2.
   verify(p2))
              |>List.fold (fun a b -> a && b)true
```

The complete and current code for this project can be obtained from the Github repository at: `https://github.com/pgrontas/PermNetMix`

Chapter 7

Conclusions and Future Work

This thesis was built on the premise that voting as a generic decision making process will be eventually computerised. Electronic voting is voting by proxy and as such there are serious issues that need to be resolved, that are made worse by the volatile nature of software. However in our opinion, these difficulties are not show stoppers. Many voting technologies that have been used for many centuries (and are still around today), exhibit more important flaws, but are still preferred in various occasions. In our opinion, the reason for this is twofold. Firstly some of them, such as vote by acclamation or show of hands are very direct and intuitive despite their lack of privacy or integrity. Secondly the lack of trust is compensated by the involvement of multiple trustworthy institutions that have conflicting interests. This might seem as trading one problem for another, but trust in institutions is built in different ways and is more comprehensible by the general public. What is important, though is that the approaches described in this thesis provide another avenue for trust building. This avenue is cryptography. One could think that cryptography would be used only for keeping the votes secret, but as we saw, cryptography proves much more important in the verification of the process and as such is a trust builder.

The different approaches reviewed in this thesis fall into three categories. The first one aims to compute the result without ever decrypting the votes. To this end the homomorphic properties of the underlying cryptosystems are utilized to combine the secret votes. One problem with such systems is that the result needs to be decrypted. Whoever performs this operation is going to get the capability of decrypting individual votes for free. As a countermeasure we require the cooperation of many different parties for the decryption capability, relying on their conflicting interests. A variation of this approach splits, instead, the vote to the different parties. Now no authority possesses the complete vote in order to be decrypted, but the voter has to perform more work. It is argued, that the first approach is more applicable to large scale elections while the latter one is more suitable to smaller scale elections where the voter is the tallier as well. Another drawback of the homomorphic approach, is that a hidden vote provides an attack surface, as an attacker might use it to insert more votes to the system. As a result the voter must provide a proof of validity. Only votes that are verified will be counted. This verification however adds computational cost. The proof itself constrains the social choice functions that can be implemented by the system. A consequence of this fact is that implementations based on homomorphic cryptosystems, like Helios, do not implement the more complex voting methods such as STV, but stick essentially to variations of yes-no voting (*1 out of c*, *t out of c*).

The second approach downscales secrecy to anonymity. In practice, this means that the votes will be decrypted at some point, but they will be anonymous by then, meaning that the link between voter and vote will be broken. This anonymity services will be provided by a set of shuffles implemented by entities collectively referred to as a mixnet. Mixnets are a mature tool, that has been used beyond voting. In our case, the fact that the contents will be eventually revealed, would, one might reckon, free the protocol from expensive proofs and allow in theory for the incorporation of more advanced social choice functions. As we saw this is not the case. Attacks utilizing corrupt voters have proved that the voter needs to prove knowledge of the proof, or else his vote could be used for tagging in order to break the anonymity of the scheme. To make matters worse, the attack could originate from the mix server, which is a more powerful agent in the protocol, as it handles all the votes and as a result could do more harm than a mere voter. Consequently the mix server must also prove that it operated as specified by the protocol. In this thesis, we examined various ways to generate these proofs from quite inefficient to the most efficient ones. In addition, we reviewed approaches that trade anonymity and/or correctness for efficiency. While these approaches have been used in practice, they remain vulnerable to variations of the tagging attacks.

The third approach splits the registration and counting functions. With the use of the cryptographic primitive of blind signatures, it allows the registration authorities to validate the origin of the ballot without seeing its contents and the counting authorities to count the contents without seeing the origin. This scheme requires an anonymous channel, but it is lightweight and simple and it comes as no surprise that it is the one boasting the greatest number of implementations.

The systems reviewed so far defend against a basic adversary. Of course there are many different ways to cheat in voting. A voter, herself might want to sell her vote. A complete system should try to avert such attempts, providing a property that is referred to as receipt freeness. In this case the randomization that was useful in hiding the votes, becomes the enemy as it can serve as a receipt. To mend this, the encryption of votes is not prepared by the voter, but by the authorities and the information is transferred in a deniable fashion.

Of course a stronger adversary still exists. His capabilities include trying to impersonate a voter, trying to force the voter to abstain from voting or forcing the voter to nullify his and probably other votes. He can be found in the one of the more interesting applications of electronic voting, namely remote voting. A fairly recent success of the voting research is providing defense against such adversaries in the form of *coercion resistance*. This property is achieved when the stronger adversary cannot tell whether his attack succeeded or not and implemented by voting multiple times using fake and anonymous credentials that are indistinguishable from the real ones. It is based on some reasonable assumptions such as the provision of an untappable channel during registration and the existence of a moment of privacy for the voter.

The properties of receipt freeness and coercion resistance are provided not as standalone protocols, but are found as *addons* to the before mentioned paradigms. More over the blind signature scheme requires the existence of an anonymous channel. Can these combinations be extended? For example would it be reasonable to build a voting system that combines homomorphic systems and mix nets; Such a system would eliminate all tagging attacks as the votes would never be decrypted. In addition it could add the garbling feature of changing the number of ciphertexts in each mix stage by homomorphically aggregating a part of the votes. Bridging these worlds would not however result in combining the best, but also the worst of them. The resulting system would constrain the social choice functions and would be quite expensive computationally. However, as research advances and more efficient proofs are found such an approach could not be ruled out. Moreover, the application of general secure multi party computation techniques could bear more fruits, under the above premise.

While the subject of electronic voting is quite mature, as it spans 30 years of literature, there remain a lot to be done. We have touched on some of these issues that will be the subject of future research. To begin with, there is the problem of the actual level of privacy provided by cryptographic techniques. By this, we mean that if vote secrecy is based on computational assumptions, there might come a day that these assumptions do not apply. We do not refer only to the solving the underlying number theoretic problems, such as the discrete logarithm. Once an election ends, the votes cast remain as records. These might be permanent records and if published on a web bulletin board could be available to anybody for an indeterminate period of time. We repeat here the well known quote of Adi Shamir, that all cryptographic keys in use today will be useless in 30 years time. All the records captured will be able subject to cracking with dire consequences. Two efforts provide solutions to this problem and are based on commitment schemes and the idea that integrity is ephemeral (during the election and for a short time thereafter), but privacy should be everlasting. The first approach utilizes the various methods for secret sharing, while the second one is everlasting privacy with homomorphic commitments for the vote and homomorphic encryption for the public, that can be easily added to existing protocols. This method provides only everlasting privacy towards the vote, and not towards the opening values. It would be an interesting idea to combine the latter method with protocols for secure erasure. Alternatively, there could be entirely new avenues of research.

An open issue of great importance is the full integration of any social choice function in voting systems. As we saw, most voting protocols proposed, are designed for yes or no, 1 out of c, or t out of c voting. Our review, pointed the difficulties to embed such functions to homomorphic schemes. While on the surface it might seem easier to incorporate them to mixnets or blind signature based schemes, the reality is that as they encode more information, they broaden the attack surface to the systems. Recall the Italian attack that uses the lower-ranked preferences of ranked ballot as an identifier. In our opinion, how to make all social choice functions first class citizens of voting systems, remains an open problem.

Coercion resistance as the newest area added to the field of voting research, has, in our view the most potential for future work. For example there is a complete subfield of cryptography, dealing with anonymous credentials that could be combined with the results we reviewed. Interesting problems would be the speedup of the duplicate and fake credential detection process combined with efficient revocation. The latter is important as credentials are obtained during the untappable registration process and are to be used during many elections. Another possible avenue for research will be the combination of techniques reviewed in this thesis and the field of auctions. These two distinct fields share many common characteristics such as the need for secrecy and integrity and the efforts to avoid coercion and collusion. Of course we are not the first to notice this and many efforts are well underway.

Last but not least, one drawback of this thesis, brought by our trying to balance between theory and practice is that less importance was given to the efforts for proving the properties of the protocols described. We are not alone in this, however. It is a fact, that proposals without proofs dominate the voting literature. Notable exceptions, are the works proving the security of mixnets and homomorphic cryptosystems in the universal composability framework. This work could be extended to schemes based on blind signatures and almost verifiable shuffles.

Bibliography

[1] Masayuki Abe. Universally verifiable mix-net with verification work independent of the number of mix-servers. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 437--447. Springer Berlin Heidelberg, 1998.

[2] Masayuki Abe and Fumitaka Hoshino. Remarks on mix-network based on permutation networks. In *Public Key Cryptography*, pages 317--324. Springer, 2001.

[3] B. Adida. *Advances in cryptographic voting systems*. PhD thesis, Massachusetts Institute of Technology, 2006.

[4] Ben Adida. Verifying elections with cryptography, 2007.

[5] Ben Adida. Helios: web-based open-audit voting. In *Proceedings of the 17th conference on Security symposium*, SS'08, pages 335--348, Berkeley, CA, USA, 2008. USENIX Association.

[6] Ben Adida, Olivier Pereira, Olivier De Marneffe, and Jean jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of helios. In *In Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE*, 2009.

[7] Ben Adida and Douglas Wikström. Obfuscated ciphertext mixing. In *In IACR Eprint archive number 394*, 2005.

[8] Ben Adida and Douglas Wikström. How to shuffle in public. In *Theory of Cryptography*, pages 555--574. Springer, 2007.

[9] Ben Adida and Douglas Wikström. Offline/online mixing. In *Automata, Languages and Programming*, pages 484--495. Springer, 2007.

[10] Jordi Puiggalí Allepuz and Sandra Guasch Castelló. Universally verifiable efficient re-encryption mixnet. In *Electronic Voting*, pages 241--254, 2010.

[11] Roberto Araújo, Sébastien Foulle, and Jacques Traoré. A practical and secure coercion-resistant scheme for remote elections. In *Frontiers of Electronic Voting*, 2007.

[12] Roberto Araújo, Narjes Ben Rajeb, Riadh Robbana, Jacques Traoré, and Souheib Yousfi. Towards practical and secure coercion-resistant electronic elections. In *CANS*, pages 278--297, 2010.

[13] Roberto Araújo and Jacques Traoré. A practical coercion resistant voting scheme revisited. In *VOTE-ID*, pages 193--209, 2013.

[14] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, PODC '01, pages 274--283, New York, NY, USA, 2001. ACM.

[15] Josh Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, pages 14--14. USENIX Association, 2007.

[16] Josh Benaloh. Administrative and public verifiability: Can we have both? In David L. Dill and Tadayoshi Kohno, editors, *EVT*. USENIX Association, 2008.

[17] Josh Benaloh. Rethinking voter coercion. In *In Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE)*, 2013.

[18] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, pages 544--553, New York, NY, USA, 1994. ACM.

[19] Josh C Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. In *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, PODC '86, pages 52--62, New York, NY, USA, 1986. ACM.

[20] Josh Cohen Benaloh. Secret sharing homomorphisms: Keeping shares of a secret sharing. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 251--260. Springer, 1986.

[21] Dan Boneh. Cryptography i. Coursera Online Course, November 2012.

[22] Dan Boneh and Philippe Golle. Almost entirely correct mixing with applications to voting. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 68--77, New York, NY, USA, 2002. ACM.

[23] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, pages 431--444, 2000.

[24] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156--189, October 1988.

[25] Johannes Buchmann, Denise Demirel, and Jeroen van de Graaf. Towards a publicly-verifiable mix-net providing everlasting privacy. In *Financial Cryptography*, pages 197--204, 2013.

[26] Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. Efficient protocols for set membership and range proofs. In *ASIACRYPT*, pages 234--252, 2008.

[27] Orhan Cetinkaya and Ali Doganaksoy. Pseudo-voter identity (pvid) scheme for e-voting protocols. In *ARES*, pages 1190--1196, 2007.

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

[29] David Chaum. Blind signatures for untraceable payments. In D. Chaum, R.L. Rivest, and A.T. Sherman, editors, *Advances in Cryptology Proceedings of Crypto 82*, pages 199--203, 1983.

[30] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030--1044, October 1985.

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

[32] David Chaum and Eugene Heyst. Group signatures. In DonaldW. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 257--265. Springer Berlin Heidelberg, 1991.

[33] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 89--105, London, UK, UK, 1993. Springer-Verlag.

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

[35] Sansar Choinyambuu. Homomorphic tallying with paillier cryptosystem, 2009.

[36] Jeremy Clark and Urs Hengartner. Panic passwords authenticating under duress.

[37] Jeremy Clark and Urs Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. *IACR Cryptology ePrint Archive*, 2011:166, 2011.

[38] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354--368. IEEE Computer Society, 2008.

[39] Michael R. Clarkson and Andrew C. Myers. Coercion-resistant remote voting using decryption mixes. In *In Frontiers in Electronic Elections (FEE 2005*, 2005.

[40] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *FOCS*, pages 372--382, 1985.

[41] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[42] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '94, pages 174--187, London, UK, UK, 1994. Springer-Verlag.

[43] Ronald Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. pages 72--83. Springer-Verlag, 1996.

[44] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. pages 103--118. Springer-Verlag, 1997.

[45] Ivan Damgård and Mads Jurik. Client/server tradeoffs for online elections. In *Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems: Public Key Cryptography*, PKC '02, pages 125--140, London, UK, UK, 2002. Springer-Verlag.

[46] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *P Y A RYAN*, page 3, 2003.

[47] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, PKC '01, pages 119--136, London, UK, UK, 2001. Springer-Verlag.

[48] Denise Demirel, Jeroen Van De Graaf, and Roberto Araújo. Improving helios with everlasting privacy towards the public. In *Proceedings of the 2012 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, EVT/WOTE'12, pages 8--8, Berkeley, CA, USA, 2012. USENIX Association.

[49] Denise Demirel, Melanie Volkamer, and Hugo Jonker. Efficient mix-net veri cation by proofs of random blocks. *IACR Cryptology ePrint Archive*, 2012:16, 2012.

[50] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644--654, September 2006.

[51] Saghar Estehghari and Yvo Desmedt. Exploiting the client vulnerabilities in internet e-voting systems: hacking helios 2.0 as an example. In *Proceedings of the 2010 international conference on Electronic voting technology/workshop on trustworthy elections*, EVT/WOTE'10, pages 1--9, Berkeley, CA, USA, 2010. USENIX Association.

[52] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, STOC '90, pages 416--426, New York, NY, USA, 1990. ACM.

[53] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, pages 427--438, Washington, DC, USA, 1987. IEEE Computer Society.

[54] Ed Felten. E-voting:risk and opportunity - online symposium, 2012.

[55] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings on Advances in cryptology---CRYPTO '86*, pages 186--194, London, UK, UK, 1987. Springer-Verlag.

[56] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *Financial Cryptography*, pages 90--104. Springer, 2001.

[57] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, ASIACRYPT '92, pages 244--251, London, UK, UK, 1993. Springer-Verlag.

[58] Jun Furukawa, Hiroshi Miyauchi, Kengo Mori, Satoshi Obana, and Kazue Sako. In Matt Blaze, editor, *Financial Cryptography*, Lecture Notes in Computer Science, pages 16--30. Springer.

[59] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 368--387, London, UK, UK, 2001. Springer-Verlag.

[60] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10--18, 1984.

[61] Panos Louridas Panayiotis Tsanakas Georgios Tsoukalas, Kostas Papadimitriou. From helios to zeus. In *In Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE)*, 2013.

[62] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, STOC '85, pages 291--304, New York, NY, USA, 1985. ACM.

[63] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 365--377, New York, NY, USA, 1982. ACM Press.

[64] Philippe Golle, Sheng Zhong, Dan Boneh, Markus Jakobsson, and Ari Juels. Optimistic mixing for exit-polls. In *Asiacrypt 2002, LNCS 2501*, pages 451--465. Springer-Verlag, 2002.

[65] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. In *Public Key Cryptography*, pages 145--160, 2003.

[66] Jens Groth. Non-interactive zero-knowledge arguments for voting. In *ACNS*, pages 467--482, 2005.

[67] Corelab Crypto Group. Discussion on an anonymous login system, July.

[68] J. Alex Halderman. Securing digital democracy. Coursera Online Course, September 2012.

[69] Qi He and Zhongmin Su. A new practical secure e-voting scheme. In *IFIP SEC*, volume 98, pages 196--205, 1998.

[70] James Heather. Implementing stv securely in pret a voter. In *Computer Security Foundations Symposium, 2007. CSF'07. 20th IEEE*, pages 157--169. IEEE, 2007.

[71] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'00, pages 539--556, Berlin, Heidelberg, 2000. Springer-Verlag.

[72] Markus Jakobsson. Flash mixing. In *PODC '99: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 83--89, New York, NY, USA, 1999. ACM.

[73] Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. In *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, pages 162--177, London, UK, 2000. Springer-Verlag.

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

[75] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli M. Maurer, editor, *EUROCRYPT*, volume 1070 of *Lecture Notes in Computer Science*, pages 143--154. Springer, 1996.

[76] Hugo Jonker, Sjouke Mauw, and Jun Pang. Privacy and verifiability in voting systems: Methods, developments and trends. Cryptology ePrint Archive, Report 2013/615, 2013. http://eprint.iacr.org/.

[77] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 61--70. ACM, 2005.

[78] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[79] Shahram Khazaei, Björn Terelius, and Douglas Wikström. Cryptanalysis of a universally verifiable efficient re-encryption mixnet. In *Proceedings of the 2012 international conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, EVT/WOTE'12, pages 7--7, Berkeley, CA, USA, 2012. USENIX Association.

[80] Shahram Khazaei and Douglas Wikström. Randomized partial checking revisited. Technical report, Cryptology ePrint Archive, Report 2012/063, 2 012. http://eprint. iacr. org, 2012.

[81] Aggelos Kiayias. Cryptography primitives and protocols, 2011.

[82] Aggelos Kiayias, Michael Korman, and David Walluck. An internet voting system supporting user privacy. In *ACSAC*, pages 165--174. IEEE Computer Society, 2006.

[83] Aggelos Kiayias and Moti Yung. Self-tallying elections and perfect ballot secrecy. In *Public Key Cryptography*, pages 141--158. Springer, 2002.

[84] Joe Kilian and Kazue Sako. Receipt-free MIX-type voting scheme - a practical solution to the implementation of a voting booth. In *Proceedings of EUROCRYPT 1995*. Springer-Verlag, 1995.

[85] Reto E. Koenig, Rolf Haenni, and Stephan Fischli. Preventing board flooding attacks in coercion-resistant electronic voting schemes. In *SEC*, pages 116--127, 2011.

[86] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols, 2010.

[87] Helger Lipmaa. Statistical zero-knowledge proofs from diophantine equations. *IACR Cryptology ePrint Archive*, 2001:86, 2001.

[88] Helger Lipmaa, N. Asokan, and Valtteri Niemi. Secure vickrey auctions without threshold trust. *IACR Cryptology ePrint Archive*, 2001:95, 2001.

[89] Panos Louridas. e-voting from the ground up and zeus, 2013.

[90] Emmanouil Magkos, Panayiotis Kotzanikolaou, and Christos Douligeris. Towards secure online elections: models, primitives and open issues. *EG*, 4(3):249--268, 2007.

[91] Jakobsson Markus and Juels Ari. Millimix: Mixing in small batches. Technical report, 1999.

[92] ABE Masayuki. Mix-networks on permutation networks. In *ASIACRYPT'99*, page 258. Springer.

[93] Yoav Shoham Matthew O. Jackson, Kevin Leyton-Brown. Game theory ii: Advanced applications. Coursera Online Course, January 2014.

[94] M. Michels and P. Horster. Some remarks on a receipt-free and universally verifiable mix-type voting scheme. In *Advances in Cryptology—ASIACRYPT'96*, pages 125--132. Springer, 1996.

[95] Mostafa Mohammadpourfard, Mohammad Ali Doostari, Mohammad Bagher Ghaznavi Ghoushchi, and Nafiseh Shakiba. A new secure internet voting protocol using java card 3 technology and java information flow concept. *Security and Communication Networks*, 2014.

[96] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *Proceedings of the 26th Annual International Conference on Advances in Cryptology*, CRYPTO'06, pages 373--392, Berlin, Heidelberg, 2006. Springer-Verlag.

[97] Moni Naor and Adi Shamir. Visual cryptography. In *Advances in Cryptology—EUROCRYPT'94*, pages 1--12. Springer, 1995.

[98] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, CCS '01, pages 116--125, New York, NY, USA, 2001. ACM.

[99] C. Andrew Neff. Verifiable mixing (shuffling) of elgamal pairs. Technical report, In proceedings of PET '03, LNCS series, 2004.

[100] Ryan O'Donnell. Example of a good zero knowledge proof, 2012.

[101] Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka, and Tatsuaki Okamoto. An improvement on a practical secret voting scheme. In *Information Security*, volume 1729 of *Lecture Notes in Computer Science*, pages 225--234. Springer Berlin Heidelberg, 1999.

[102] Tatsuaki Okamoto. An electronic voting scheme. In Nobuyoshi Terashima and Edward Altman, editors, *Advanced IT Tools*, IFIP — The International Federation for Information Processing, pages 21--30. Springer US, 1996.

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

[104] Michael O'Keeffe. The paillier cryptosystem - a look into the cryptosystem and its potential application, 2008.

[105] Eric Pacuit. Voting methods. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 2012 edition, 2012.

[106] Eric Pacuit. Making better group decisions: Voting, judgement aggregation and fair division. Coursera Online Course, March 2014.

[107] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 223--238, Berlin, Heidelberg, 1999. Springer-Verlag.

[108] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *EUROCRYPT*, pages 248--259, 1993.

[109] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Proceedings of the 10th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'91, pages 522--526, Berlin, Heidelberg, 1991. Springer-Verlag.

[110] Holger Petersen. How to convert any digital signature scheme into a group signature scheme. In *Security Protocols Workshop*, pages 177--190, 1997.

[111] B. Pfitzmann. Breaking an efficient anonymous channel. In *Advances in Cryptology—EUROCRYPT'94*, pages 332--340. Springer, 1995.

[112] B. Pfitzmann and A. Pfitzmann. How to break the direct rsa-implementation of mixes. In *Advances in Cryptology—EUROCRYPT'89*, pages 373--381. Springer, 1990.

[113] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96--99, January 1983.

[114] Ronald Rivest. 6.897 - selected topics in cryptography, 2004.

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

[116] Peter Y. A. Ryan and Steve A. Schneider. Prêt à voter with re-encryption mixes. In *ESORICS*, pages 313--326, 2006.

[117] Peter YA Ryan. Prêt à voter with paillier encryption. *Mathematical and Computer Modelling*, 48(9):1646--1662, 2008.

[118] Peter YA Ryan, David Bismark, JA Heather, Steve A Schneider, and Zhe Xia. The prêt à voter verifiable election system. *IEEE transactions on information forensics and security*, 4(4):662--673, 2009.

[119] Krishna Sampigethaya and Radha Poovendran. A framework and taxonomy for comparison of electronic voting schemes. *Computers And Security*, 25(2):137--153, 2006.

[120] Michael Schlapfer, Rolf Haenni, Reto E. Koenig, and Oliver Spycher. Efficient vote authorization in coercion-resistant internet voting. In Aggelos Kiayias and Helger Lipmaa, editors, *VOTE-ID*, volume 7187 of *Lecture Notes in Computer Science*, pages 71--88. Springer, 2011.

[121] Bruce Schneier. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[122] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *In CRYPTO*, pages 148--164. Springer-Verlag, 1999.

[123] Berry Schoenmakers. Cryptographic protocols, 2013.

[124] Jorn Schweisgut. Coercion-resistant electronic elections with observer. In Robert Krimmer, editor, *Electronic Voting*, volume 86 of *LNI*, pages 171--177. GI, 2006.

[125] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612--613, November 1979.

[126] Victor Shoup. Practical threshold signatures. In *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'00, pages 207--220, Berlin, Heidelberg, 2000. Springer-Verlag.

[127] Victor Shoup and Victor Shoup. Why chosen ciphertext security matters, 1998.

[128] Gerardo I. Simari. A primer on zero knowledge protocols, 2002.

[129] Russell Michaels Simon Ardizzone. Hacking democracy, 2006.

[130] Warren D. Smith. New cryptographic voting scheme with best-known theoretical properties, June 2005.

[131] Oliver Spycher, Reto Koenig, Rolf Haenni, and Michael Schlapfer. A new approach towards coercion-resistant remote e-voting in linear time. In *Proceedings of the 15th international conference on Financial Cryptography and Data Security*, FC'11, pages 182--189, Berlin, Heidelberg, 2012. Springer-Verlag.

[132] Tobias Volkhausen. Paillier cryptosystem: A mathematical introduction, 2006.

[133] Changjie Wang and Ho fung Leung. A secure and fully private borda voting protocol with universal verifiability. In *COMPSAC*, pages 224--229. IEEE Computer Society, 2004.

[134] Stefan G. Weber, Roberto Araujo, and Johannes Buchmann. On coercion-resistant electronic elections with linear work. In *ARES*, pages 908--916. IEEE Computer Society, 2007.

[135] Wikipedia. Benaloh cryptosystem, 2012. [Online; accessed 2-September-2013].

[136] Wikipedia. Ciphertext indistinguishability, 2012. [Online; accessed 28-December-2013].

[137] Douglas Wikström. Five practical attacks for "optimistic mixing for exit-polls". In *Selected Areas in Cryptography*, pages 160--174. Springer, 2004.

[138] Douglas Wikström. A sender verifiable mix-net and a new proof of a shuffle. In *ASIACRYPT*, pages 273--292, 2005.

[139] Douglas Winkstom. Mixnets for voting. Secvote 2010, 2010.

[140] Brecht Wyseur and Mina Deng. Re-trust: Remote entrusting by run-time software authentication, deliverable: D3.3, 2009. project deliverable RE-TRUST.

[141] Zhe Xia, Chris Culnane, James Heather, Hugo Jonker, Peter YA Ryan, Steve Schneider, and Sriramkrishnan Srinivasan. Versatile prêt à voter: Handling multiple election methods with a unified interface. In *Progress in Cryptology-INDOCRYPT 2010*, pages 98--114. Springer, 2010.