Enhanced Tally Scheme for the DEMOS End-2-End Verifiable E-voting

Thomas Souliotis

Master of Science

School of Informatics

University of Edinburgh

2018

Abstract

E-voting can solve many problems that traditional conventional voting systems cannot, in a secure mathematically proven way. The DEMOS e-voting system is one of the many examples of modern state of the art e-voting systems. The DEMOS e-voting system has many desirable properties and uses many novel techniques. However, DEMOS has some problems that need to be solved, so as to extend its usability. In this dissertation, we explain how DEMOS works and what makes it so special. Moreover, we extend DEMOS usability by providing a new way of conducting enhanced voting schemes in the context of DEMOS. This is achieved by changing the way that the ballots are encoded and by providing a new zero knowledge proof approach. Furthermore, our work is supported by the necessary security proofs, while a first practical implementation is also provided. We also provide a novel theoretical proof of concept, for far more complex voting systems like STV, which might be the first of its kind for homomorphic systems. Finally, we provide a new zero knowledge proof of a shuffle argument, which can be used for applications outside of this dissertation purposes.

Keywords: E-voting, DEMOS, Standard Model, Shuffle Proofs, E2E Verifiability, Receipt-freeness, Borda count, Preferential Voting, STV, Homomorphic Encryption.

Acknowledgements

This project was supervised by professor Aggelos Kiayias. His background and expertise helped me a lot during our discussions for further understanding and developing complex protocols. Moreover, I would like to thank Dr Thomas Zacharias and Dr Bingsheng Zhang who have helped me a lot, by explaining anything required about the DEMOS protocol and E2E e-voting, in general. Finally, I would like to thank Marios Levogiannis for his help in developing the practical part of this work.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Thomas Souliotis*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we will introduce voting, in general, focusing on e-voting. Moreover, we will state the importance of cryptography in modern voting systems, and finally we will talk about the structure of this dissertation and give a brief overview.

1.1 A Voting Process

1.1.1 What is a voting process

A voting process can be defined as a way, that a group of two or more people express their opinion or make a collective choice about a subject. Voting processes have been around almost forever, since they have been used in some form from the first time that people gathered into communities, while they were established with the first democratic societies [1].

However, most voting processes should obey some rules. These rules are mostly related to the **privacy** of the voter and the **integrity/verifiability** of the voting process. The privacy requirement ensures that nobody else other than the voter learns what the voter voted, while the integrity/verifiability requirement ensures that the vote was **cast as intended, recorded as cast and counted as recorded** [2]. We note that many voting processes do not require both criteria (especially the privacy requirement is not necessary in many cases) and the level of fulfillment may vary depending on the importance of the voting process.

Moreover, we have stated what is a voting process and its basic properties, but we have not yet described how a voting process is really conducted. Generally, the voters should make a choice regarding a question. This choice is based on the voting system that was already pre-decided, and then he/she casts his/her vote. When the voting part is over, the result is counted (again based on the voting system chosen) and the 'winner(s)' is/are determined. This is a very simple way to describe a normal voting process. However, we have intentionally kept it simple, so as to emphasize further on the constituting parts of the process.

We stated that the voting process is generally defined, based on the corresponding voting system. The first and most simple type of voting is the **simple approval voting**, where the voter has to choose one choice among others (1 out of m). An evolution of that, is the generalized approval voting, where the voter can choose a subset of choices, instead of just one (k out of m). These voting systems, despite their practicality and their wide use, lack into completeness, as no relation between the choice is revealed. As a result, **more complex systems have been proposed to solve the voting problem**, where the voter submits the candidates/choices in an order of preference. Other than that, a **score voting** may be implemented, such that the choices are just graded based on the will of the voter and no specific order is required. Despite that, we should also mention that the the **final result is again computed based on the chosen system design**. Briefly, some of the most important voting systems proposed are among others [3]: Approval, Borda count, Copeland, IRV, Kemeny-Young, Majority judgment, Minimax, Plurality, Range voting, Ranked pairs, Runoff voting, Schulze, Sortition, arbitrary winner, etc. Of course we have to state that it has been proven from game theory, that there is no way to extract the best possible outcome no matter the voting system due to Arrow's impossibility theorem [4].

1.1.2 Typical Voting and Transition to e-voting

We previously defined what is a voting process and its basic properties. However, we did not present at all, how this process will take place, and most importantly, how the necessary requirements are satisfied. Initially, the very first voting processes were voice votes, where the privacy requirement does not exist. The first voting systems that tried to obscure the voter's identity can be located in ancient Greece, where a form of ballot was created and the voters were anonymized by throwing seemingly identical

'votes' (e.g. rocks or other artifacts) into these ballots. This process was evolved as time went by and eventually we have paper -almost identical- ballots, but the general idea remains the same. Finally, all the ballots are opened and counted at the end of the process, and the winner(s) are elected.

The fundamental idea of current typical voting systems, is that there will be some **trusted authorities**, responsible for the soundness of the whole process. These authorities, normally, consist of some trusted parties, (e.g. for governmental elections judges, policemen, politicians and everyday people are required to cooperate under some regulation) that are supposed to ensure the procedure. As it is apparent, big problems may arise in the current systems, when the trusted authorities are malicious, and/or do not do their jobs correctly. In a secure, democratic environment these risks are considered to be minimal, although still existent. With the developments in technology, e-voting was also an idea that could save us from unnecessary costs and would highly increase efficiency. However, in order to ensure privacy and integrity in an online system, the authorities should be now defined differently while the whole process should also be re-considered. Therefore, e-voting was not really an option due to the insufficient security precautions.

All these changed when the first cryptographically secure voting systems were proposed [5, 6]. In these systems, the trusted authority 'loses' most of its power, while the privacy and the integrity of the process is purely mathematically proven and independent of how some people will do their jobs. These systems, are called End-to-End (E2E) voting systems, and satisfy an E2E encryption of the vote,solving the problem of the privacy and integrity. Many such systems have been proposed and we will refer to them in the next chapter. What needs to be stated for now is that these systems rely on the security of some mathematical and cryptographic protocols for fulfilling the election requirements.

1.2 Objective, Motivation and Results of this Dissertation

We explained what is a voting process, its importance and the new era of e-voting systems. In this dissertation we initially, intend to explain the basic cryptographic notions which are used in modern state of the art e-voting systems, for a better understanding

of our complex work. Furthermore, we focus on DEMOS e-voting system, and explain it further, stating its novel techniques. Based on the DEMOS e-voting system, we then present our new enhanced DEMOS protocol which comes to solve some unresolved issues of the current system. Providing a new enhanced system based on DEMOS protocol that will include more options than the current is actually, the main **objective** of this dissertation.

The **motivation** for this work is that modern voting systems, are not just approval as we stated before. They can be a lot of things and a simple approval system is not enough for many elections, nowadays. Therefore, considering that DEMOS offers some very beneficial properties, we decided to transform DEMOS so as to offer these additional options.

The **results** of this dissertation are many and spread beyond the initial objectives. We managed to deeply understand and explain all the protocols used in DEMOS and other modern e-voting systems. Based on those and by developing some new ones, we provided a new tally scheme for DEMOS. This new tally scheme is implemented, initially, by a theoretical description of the new system. In this theoretical research, we also provide a new efficient zero knowledge shuffle proof argument, which we use in our system, but its use can be expanded further than the scope of this dissertation. Finally, the dissertation had a practical part too, since we provided a preliminary practical implementation for our new enhanced scheme, which can be used as a back-end to the already implemented 'simple' DEMOS protocol.

1.3 Summary - Outline

After this introductory chapter, we present some necessary background in the next chapter, so as anyone to try and understand, how and why everything works. Then, we present the current DEMOS implementation, where we explain further how it was implemented and what are the key and novel techniques it uses. In chapter four, we present our new enhanced protocol in full detail, alongside with the necessary proofs and explanations, while the next chapter is about the practical implementation of what we provided theoretically. Finally, in the last chapter we analyze what we managed in this dissertation, while we talk about unresolved issues and potential future work.

Chapter 2

Preliminaries And Background

In this chapter we present all the necessary background, anyone would require to comprehend our work. Moreover, a brief description of the most significant protocols will be given, while we will explain further some core ideas, when required.

2.1 Mathematical Background

Initially, we will refer to the mathematical concepts that most cryptographic protocols are based on. We will also provide cites and links to anything not elaborated here, as these are considered general knowledge and are not really part of our dissertation.

2.1.1 Algebra, Number and Group Theory

In this section some basic algebraic protocols are stated. Any interested party shall be familiar with those, so as to understand cryptography related projects. We also present some number theory significant notions that are also needed.

First of all, the definition of a group [7] is very important.
Definition 2.1.1. Group: A **Group** $(G,*)$ is a set G, alongside with an operation $*$, which satisfies the following requirements.

- **Closure:** $\forall g,h \in G \Rightarrow g*h \in G$
- **Associativity:** $\forall g,h,l \in G : (g*h)*l = g*(h*l)$.

- **Identity element :** $\exists e \in G \ : \forall a \in G \ \ e * a = a * e = a$
- **Inverse element :** $\forall g \in G \ \exists h \in G : \ \ g * h = h * g = e$, where e is the identity element and h is denoted as g^{-1}.

Other important definitions here are the **order** of a group, the **order of an element of a group**, the **subgroup**, the **Cyclic group**, **Abelian groups**, the **ring** R, the **field** F, the **Chinese Remainder Theorem**, etc. We are not going to define everything here, as they are already defined in multiple books and for most we just refer them and provide a citation [7, 8] for anyone interested.

Definition 2.1.2. Order of a Group: We define the number of elements of (a finite) group G, as order of a group G, and we denote that as $|G|$.

Definition 2.1.3. Order of an Element of a Group: The order of an element g of a group G is a number x s.t. $g^x = e$, and we denote $x = ord(g)$

Definition 2.1.4. Cyclic Group: A group G will be called cyclic, if

$$\exists g \in G : \forall a \in G \ \exists x \ : g^x = a$$

2.1.2 Statistical Analysis

A statistical analysis is crucial for any cryptographic protocol, as this is the measure of success for the protocol. Since, it is impossible to examine a protocol for any possible set of inputs and outputs, a statistical analysis solves our problems by providing ways for defining vague concepts as security in a mathematical way. Moreover, the statistical analysis contains concepts like **Discrete and Conditional Probabilities**, **Random Variables** and many others. We need to highlight the importance of **Statistical Distance** [9] here, by providing a simple very useful definition. The notion of Statistical Distance is utilized by many cryptographic protocols for the security part.

Definition 2.1.5. Statistical Distance: Let X, Y be random variables. We define the statistical distance Δ as:

$$\Delta[X,Y] = \frac{1}{2} \sum_{u \in V} |Pr[X = u] - Pr[Y = u]|$$

with V being the union of supports of X and Y.

The above metric helps us measure how much variables following specific distributions, differ one from another, which is very useful to security analysis. As a complementary to the above, we also define statistical tests.

Definition 2.1.6. Statistical Test: A statistical test A for an ensemble $D = \{D_n\}, n \in N$ is an algorithm that takes input elements from D_n and outputs values in $\{0,1\}$ for each $n \in N$.

It is proven (chapter 8.8 of [10]) that the statistical distance with respect to a test A, is:

$$\Delta[X,Y] \geq \Delta_A[X,Y] = |Prob[A(X) = 1] - Prob[A(Y) = 1]|, \forall A$$

while there is always an A s.t. the equality holds. What the above really means is that the statistical tests will provide always a less precise correlation between two distributions, while there is always a statistical test (the one that actually checks all possible scenarios) that will provide the same results as the true statistical distance. So, if we consider the A to be PPT (probabilistic polynomial time) bounded algorithms we can then perform a statistical proof of security.

2.2 Cryptographic Notions

Now, we are ready to state the most important cryptographic notions and explain them further for our dissertation. We present almost all the important cryptographic primitives that have been developed and are widely used in e-voting, while we explain in more details the ones that we actually need in DEMOS and our improved system.

2.2.1 Modern Cryptographic Systems

People use cryptography for many reasons, but mainly so as to communicate in a secure and verifiable way, obstructing any third (possibly) malicious party being able to intercept and/or tamper the messages they exchange. Thus, some requirements must be followed. There are many ways that someone may handle this problem, such as creating a secure channel between the participants, meet in person with each other, or send an encrypted message. It becomes apparent that all other ways, other than an encrypted message, will fail in the general case, or will eventually need some kind of encryption.

The first attempts for secret and secure message transmission are located in ancient times. There were many systems that were developed over the course of time e.g. Caesar cipher, Substitution and transposition cipher, Vigenre cipher, Playfair cipher, XOR

cipher, etc. Eventually, almost all the initial cryptographic systems shared a common core protocol, the two (or more) participants shared one (or more) common encryption/decryption keys, and they encrypted/decrypted the message they sent/received. All these systems, with the above form are called **symmetric cryptosystems** [11]. Their main problem is the need of an a priori secret and secure communication. This changed, when **asymetric or public key cryptography** [12] was proposed.

2.2.1.1 Symmetric Cryptosystems

Although symmetric cryptosystems are not used in DEMOS context, we just simply refer to them, due to their importance. A symmetric cryptosystem consists of a plaintext $m \in M$, an encryption key $k \in K$, a ciphertext $c \in C$, an encryption algorithm $Enc(k,m) = c$ and a decryption algorithm $Dec(k,c) = m$. As explained previously, symmetric cryptosystems require a secure and secret pre-communication so as the key k, to be exchanged. The first symmetric cryptosystems were the classical ciphers e.g. Substitution cipher, Vigenere cipher, Vernam cipher, One-Time Pad, transposition ciphers etc. Most of these ciphers belong to history while more recent, widely used, ones are the Data Encryption Standard (DES) [13] which is based on Feistel cipher. DES was later replaced by Advanced Encryption Standard (AES) [14], due to the security issues it faced [15]. Finally, symmetric systems can be also distinguished whether they are based on Block Ciphers or on Stream Ciphers [16].

2.2.1.2 Asymmetric Cryptosystems

Asymmetric encryption is based on the principle, that each party that wants to send a message creates a public key pk which is known to everyone, and a secret key sk, which is private and nobody else knows its value. Again, we have a plaintext $m \in M$, a ciphertext $c \in C$, an encryption algorithm $Enc()$, a decryption algorithm $Dec()$, and of course a key generation algorithm $Gen()$. The main difference now, is that anyone who wants to send an encrypted message, say m to a party A, will encrypt the message with A's public key, s.t. $c = ENC(pk_A, m)$, and send it to A. Now, anyone may eavesdrop the ciphertext c, but only A will be able to decrypt it by calculating:

$$Dec(sk_A, c) = Dec(sk_A, ENC(pk_A, m)) = m$$

and receive m.

Many public key cryptography systems have been created, with some of the most important being **Paillier** [17], **RSA** [18], **ElGamal** [19] and many others. We care more about the ElGamal system, as this is the one we actually use in our implementation of DEMOS. ElGamal is based on the Diffie-Hellman key exchange (we will later refer to the DH protocol), and is defined over a cyclic group G, with g as a group generator, a prime order of q and λ as a security parameter. More precisely, the system is the following:

- **Key generation:**

$$(pk, sk) \leftarrow Gen(1^{\lambda})$$
$$x \xleftarrow{r} \mathbb{Z}_q$$
$$h = g^x$$
$$pk = ((p,q,g),h)$$
$$sk = x$$

- **Encryption:**

$$m \rightarrow M \in G$$
$$r \xleftarrow{r} \mathbb{Z}_q$$
$$c = Enc(pk, M) = (c_1, c_2) = (g^r, h^r M)$$

- **Decryption:**

$$Dec(sk, c) = \frac{c_2}{c_1^{sk}} = \frac{h^r M}{(g^r)^x} = M$$

We will not get into more details for now, regarding the security analysis, as the next section refers to that. We just state that in our dissertation, we use a variation of ElGamal, the lifted ElGamal, where we encrypt the g^M and not just M.

2.2.2 Security in Cryptography

We previously explained that security in cryptography is a subject that can have multiple definitions. Firstly, we provide the problems that most modern asymmetric cryptosystems are based on, and then state some models of security that help us define the

security of a system. Generally, stating that a system is secure is shown by proving that it is computationally hard for an adversary to break the security design of this problem. Computational hardness, however, is another very important aspect that someone might have to investigate. In our case, we just state that modern cryptography is closely related to the NP-hardness and to the fact that $NP \neq P$, as if equality holds then most current systems are useless. Nevertheless, even if $NP \neq P$, that does not mean that we are undoubtedly secure, it just means that we are not wrong yet. Finally, reductions are also a very important aspect of our analysis.

2.2.2.1 Discrete Logarithm

The Discrete Logarithm (DLOG) [20] problem is one of the most significant in modern cryptography. Its formal definition is:

Definition 2.2.1. Discrete Logarithm: Suppose that G is a finite cyclic group, with group generator g. The discrete logarithm of an element $a \in G$ is denoted as: $log_g a$, and is an integer x s.t. $a = g^x,\ 0 \leq x < |G|$

The DLOG assumption, states that there is not an efficient way to classically calculate the DLOG, in the general case. This assumption is a very important one, since some of the most widely used cryptosystems are based on this. The DLOG problem is considered 'difficult' because it is an NP-intermediate problem [21], so its computation is thought to be computationally hard.

2.2.2.2 Diffie-Hellman

Now, we present one of the most important protocols in cryptography, the Diffie-Hellman key exchange protocol [22]. In this protocol two parties A and B, receive a common input p, q, g, where p is a prime, g is a group generator of $G = \mathbb{Z}_p$ and the group G of order q. The protocol is the following:

- A calculates: $x_A \xleftarrow{r} \mathbb{Z}_q$ and $y_A \leftarrow g^{x_A}\ mod\ p$ and sends y_A to B
- B calculates: $x_B \xleftarrow{r} \mathbb{Z}_q$ and $y_B \leftarrow g^{x_B}\ mod\ p$ and sends y_B to A
- Both A, B calculate the common key as A computes $key = y_B^{x_A}\ mod\ p$ and B computes $key = y_A^{x_B}\ mod\ p$

The above protocol is the core idea behind the computational Diffie-Hellman (CDH) and the decisional Diffie-Hellman (DDH) [23], where it is proven by reduction that $DDH \leq CDH \leq DLOG$. So, from the DDH assumption, we 'expect' that DDH problem is a 'hard' problem (from the statistical analysis techniques we provided before). Therefore, most modern crypto-systems are reduced to the DDH problem assumption, and thus have a security proof.

2.2.2.3 IND-CPA, IND-CCA and IND-CCA2 Security

The **IND-CPA** [24] security stands for the indistinguishability under chosen plaintext attack, and is a probabilistic game where an adversary A should win with a specific probability so as to be secure. More precisely it is defined as:
Definition 2.2.2. IND-CPA security A public-key cryptosystem with security parameter λ is IND-CPA secure if all PPT A can win the IND-CPA game with probability p where $p \leq \frac{1}{2} + negl(\lambda)$

Similarly, we define **IND-CCA** [25] security stands for the indistinguishability under chosen ciphertext attack, and **IND-CCA2** which is **IND-CCA** but with some additional queries to the oracle. The only difference in these securities is that adversaries have access to an oracle so as to query ciphertexts and get back plaintexts. We will not analyze further the securities we stated before, since this is not the goal of this dissertation. However, we just claim that ElGamal is IND-CPA secure, but not IND-CCA2 and IND-CCA secure.

2.2.3 Commitment Schemes

A commitment scheme [26] is a protocol, that enables a party to commit a specific value (or state) while keeping this value secret from others. Finally, after a specific course of actions, or whenever the party that knows the value wants, the value is revealed, and anyone can check and verify that this value is actually the committed one. From the previous high level description, we deduce that a commitment scheme consists of two phases the **commit** and the **open** phase.

More formally, the algorithms that a commitment scheme consists of, are the $Param(\lambda)$ which is the parameter generation algorithm with λ as a security parameter, the $Commit()$

algorithm and the $Verify()$ algorithm. A commitment scheme has two important properties, that should be obeyed so as to be considered secure. The **binding** property and the **hiding** property. These properties are again modeled with an adversarial game where each PPT A should win the binding game with probability $p \leq negl(\lambda)$ and the hiding game with probability $p \leq \frac{1}{2} + negl(\lambda)$.

A commitment scheme may be modeled with the use of **hash functions**. A hash function, is a function that gets a message m of any length and maps it to a constant size response. Cryptographic hash functions are a very important aspect of modern cryptography, and are used for multiple reasons, due to each properties. Generally, these functions are easy to compute, but hard to invert, while collision resistance is also another important property. The most well known hash functions are SHA1, SHA2, MD5, etc.

Other than that, **digital signatures** are also a very important way for a receiver to prove that the message he received is authentic, and was not tampered in any way. There are multiple ways, that this can be done, and we will not get into more details, as it is out of the scope of this dissertation.

2.2.4 Zero Knowledge Proofs

In cryptography, as well as in real life, we often need to prove we know a claim (witness), regarding a statement. Normally, this is proven by just presenting the statement and what we know and the other party can easily check the validity. However, many times, especially in cryptography, it is required that we do not disclose what we know, either that being a specific value/number or something more complicated.

Therefore, zero-knowledge proofs (ZKP) were devised [27] to solve this problem. A ZKP provides sufficient proof that a party P (prover) can convince another party V (verifier), that he knows a specific value w (witness) regarding a statement x. In order to do so, there is also a predicate R that can certify that a specific w is a valid witness to a statement x, if $R(x,w) = 1$, while the proof of knowledge that is created by P for V is denoted as π. Moreover, the predicate is considered to be a polynomial time function, while P and V are also bounded to be PPT parties, for most protocols.

Some important aspects of ZKP in the context we are going to study are that, finding a w such that $R(x,w) = 1$ for an x should be (computationally) 'hard', so as all the

procedure to be meaningful(if it is trivial to find a w no zkp is required), while the validity of the proof π can be checked efficiently and not (computationally) 'hard'. Furthermore, a formal definition of ZKP can be found here [28], and is actually just a formal way of expressing what we have defined above. We provide the basic properties a ZKP, where additionally to the above notation, z is a string and $out^{P}_{P,V}(), out^{V}_{P,V}()$ is the output of the protocol of P and V respectively:

- **Completeness:** If the statement is true, then an honest P convinces an honest V with very high probability:

$$Pr[out^{V}_{P,V}(x,w,z) = 1] \geq 1 - negl(\lambda)$$

- **Soundness:** If the statement is false, a malicious P convinces an honest V with negligible probability. Another, stronger definition of the above, is that if a V is convinced, then an extractor K an be constructed -which has more power than a simple V as it is probabilistic Turing machine and can rewind the protocol-, which extracts a witness for x.

- **Zero Knowledge:** A verifier does not learn anything other than the validity of the statement. This is formalized with **statistical** zero knowledge, where V is given access to a PPT simulator S which can simulate the whole transcript, and the two transcripts, the real and the simulated have negligible statistical distance. More formally:

$$\Delta_A[S(x,y), out^{V}_{P,V}(x,w,z)] < \varepsilon$$

  We are going to deal with a weaker notion of zero knowledge, the **honest verifier zero knowledge (HZVK)**, where we suppose that while V can normally perform polynomially time many computations, it is assumed that he is not malicious and will follow the protocol as prescribed.

Next, we present some of the most important zero knowledge protocols that are used in almost every e-voting system.

2.2.4.1 Schnorr Protocol

We will now present one of the main zero knowledge protocols, which might be considered the beginning of all, the Schnorr protocol [29]. This protocol, is based on the

DLOG assumption we previously explained. The protocol, is constructed over a cyclic group G of order q, with a group generator g. The common input on P,V is $((p,q,g),y)$ and the goal of the protocol is for P to prove that he knows w such that, $w = log_g y$. The protocol is the following:

- P choses a random $r \in \mathbb{Z}_q$ and commits to that by sending to V the $t = g^r$.
- V responds to v with a random challenge $c \in \mathbb{Z}_q$.
- P receives, c, computes and sends his response $s = r + wc$. V verifies s and accepts the proof iff $g^s = ty^c$

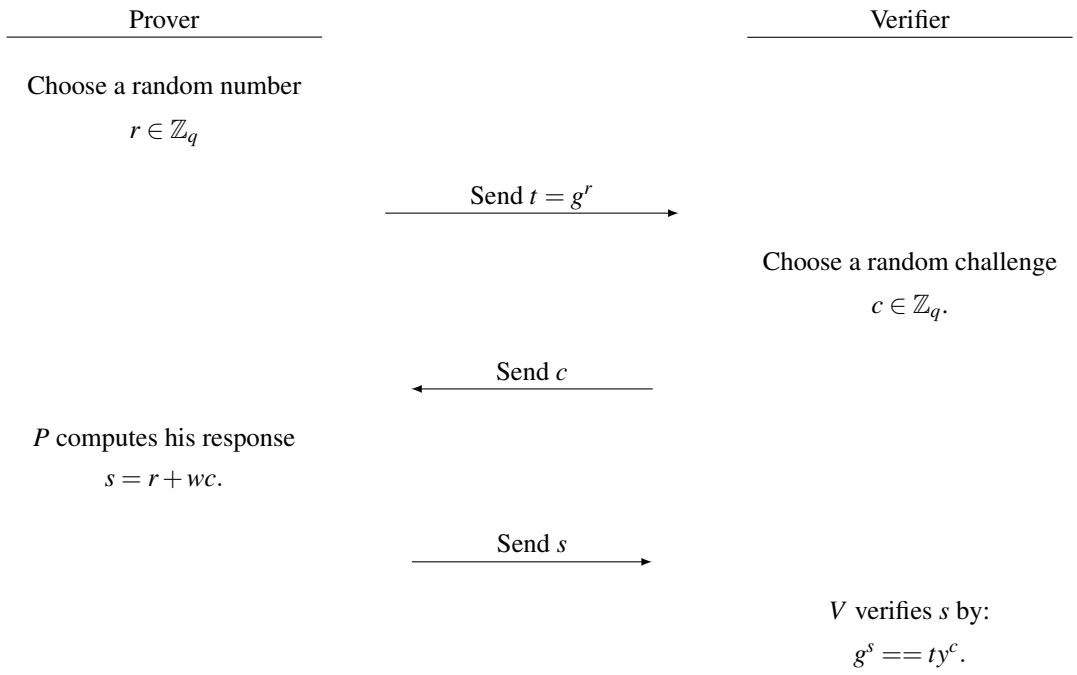

Figure 2.1: Schnorr Protocol

The Schnorr protocol is a 3-step protocol between two interactive parties P,V, and satisfies the necessary properties of a ZKP. 3-step protocols(commitment, challenge, response) like this are called Σ - protocols. A simple variation of Schnorr protocol is the Chaum-Pedersen protocol [30], which is again a 3-step Σ - protocol. There are many variations and 3-step protocols are not the only that exist as there are many other 5-step or 7-step protocols. However, **in the context of e-voting the 3-step protocols are the most important** and the ones that are usually used.

2.2.4.2 Disjunctive and Conjunctive Protocols

In addition to the above, many times it is required to prove knowledge of more complex statements. Thus, many protocols have been created. **Conjunctive** protocols or simply **AND**-proofs provide proof regarding a conjunction of statements, meaning knowing w s.t. $y_1 = g_1^w$, $y_2 = g_2^w$,... (like in the Chaum-Pedersen protocol). Furthermore, there are **disjunctive** protocols or simply **OR**-proofs, where a P can prove a disjunction of statements. More precisely, P proves that he knows either w_1 or w_2 s.t. $y_1 = g^{w_1}, y_2 = g^{w_2}$. The above protocols, are very useful and are used in many modern e-voting systems (e.g. current Helios implementation). Other than the above, there are multiple other systems, based on similar ideas and on the DLOG assumption such as **EQ**(equality) proofs, **NEQ** (not-equality) proofs, etc.

2.2.4.3 Shuffle Proofs

Shuffle proofs are another very important example of ZKP. A shuffle proof, is a proof where a P provides evidence to V that a final collection of ciphertexts $\{\psi_i'\}$ is a shuffle of some initial collection of ciphertexts $\{\psi_i\}$. Moreover, these ciphertexts are normally re-encrypted with new randomness r_i', since the connection between ψ_i and ψ_i' should be 'hard'.

In our dissertation, shuffle proofs are very significant, since we use them in the new enhanced DEMOS system. Shuffle proofs are used in many e-voting systems, mostly Mix-net based as they provide a way of anonymization of the votes-ciphertexts from the voters, but they can also be used in homomorphic based systems as we will see in our case. Many variations and many protocols exist in this kind of proofs [31, 32, 33] (shuffle re-encryption proofs, shuffle decryption proofs etc.) but we will mainly focus on re-encryption proofs as this is what we need.

2.2.4.4 Non-interactive Proofs

In the previous subsections we stated some of the most important ZKP. All of the above systems, are interactive, with two parties P, V which communicate and exchange some values. However, in e-voting systems, such interactive systems are useless as the voter should not communicate and the voting process should be an one-time thing.

The problem of transforming an interactive system to a non-interactive is usually done through **Fiat-Shamir Heuristic** [34].

Furthermore, the Fiat-Shamir Heuristic uses a hash function (call to a random oracle) that randomizes the challenge creation by hashing some data that are not controlled by P. So, P produces a proof based on a random challenge c, which he cannot control. However, there are many problem that arise with the use of random oracles and Fiat-Shamir heuristic function [35, 36], with the main being that there is not a proper way to prove that the random oracle is truly random. The above randomness may be also produced in some systems by using a common reference string (CRS), but this can again be manipulated by malicious EA and voters. On the other hand, DEMOS novelty, is that it uses a novel way to generate this random challenge without calls to random oracles as we will see. Finally, our new protocol, follows the principles of the initial system and does not depend on calls to random oracles.

2.3 Voting Systems

We described some of the most important notions in cryptography, that are related to voting systems. We have, also, stated in the introduction, the importance of modern E2E voting systems as well as the problems they solve. We will now refer to how these systems are generally implemented, what conditions they try and should satisfy, and the core concepts behind each implementation.

2.3.1 General Implementations and Requirements

The following are some basic requirements that e-voting should follow. We have, more or less, described some of them in the previous chapters, but we just provide them as a whole here:

- The connection of the ballot's decrypted message and the voter's identity must remain secret, so as to ensure the **privacy** requirement.
- Nobody should be able to tamper an encrypted vote in a meaningful way, for the **integrity** of the process.
- The whole process should be **publicly verifiable** by anyone.

- The whole system should be **E2E** verifiable, meaning that the ballots should be cast as intended, recorded as cast and tallied as recorded.

- Ideally, a system should be **coercion resistant**, since the voter should choose whatever he prefers without anyone else being able to coerce a choice. In order to increase the coercion resistance, a **receipt free** protocol may be implemented. Therefore, the voter will not be able to prove what he voted to any third person.

- The system should be **fair** against all users, while it should also be **fault tolerant**, and keep functioning, despite attempts from malicious users deviating the protocol.

In modern E2E systems the basic entities of a voting process are the election authority (EA) which is responsible for conducting the voting, the bulletin board (BB) where all public information can be posted by anyone, the voters and the candidates. We should state that the BB is considered as a database where everyone has access and can eventually post what he wants.

Now, we present how an e-voting is basically implemented. Generally, in E2E voting systems two main implementations are chosen. The first one is based on **Mix-nets** [37] approach, and the second one is connected to systems that are created based on an **homomorphic encryption protocol**. There are many implementations that might even be a combination of both, or with other variations but we just describe the general cases here. A simple analysis on what each system offers, and what is the main idea of the current implementations is shown below.

2.3.2 Mix-nets

A Mix-net or a Mix network is a system that is based on a network with one or more server layers, where each layer has as input the output of the previous layer, which it randomly shuffles and sends to the next layer.

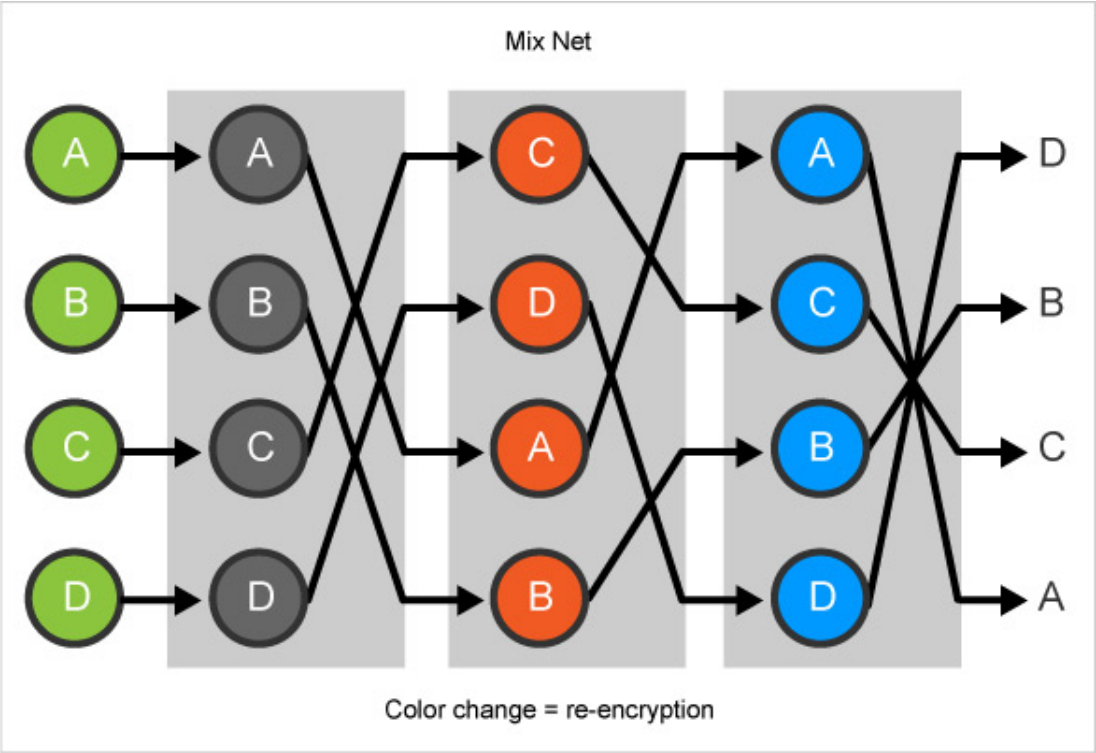

Figure 2.2: A simple Mix net with 3 layers

There are many uses of Mixnexts e.g. in Tor [38] and many other cryptographic applications. In the context of voting, these systems are used to anonymize the initial votes cast by the voters. The way a Mixnet based system works in practise is the following:

- The voters cast their votes as ciphertexts encrypted with an encryption scheme e.g. ElGamal. and under some randomnesses $\{r_i\}_{i=1}^{n}$.
- The above votes are then inserted to the first layer of the mixnet, where they are shuffled and re-encrypted with some randomnesses $\{r'_i\}_{i=1}^{n}$. This process is repeated for all the layers that the system has. For every shuffle and re-encryption a shuffle proof is provided by the EA responsible for that.
- Finally, the fully anonymized votes are opened, tallied and counted, producing the final result.

The above, is a simple description about how these systems work, however, there might be some differences e.g. instead of re-encryption there might be a partial decryption.

Several systems are based on a mixnet e.g. Zeus system [39]. The main problems with these systems, are connected to the computational complexity of performing all of the computations, which burden the EA. Other more important problems are that the EA

has some extra power as it may easily extract who voted what, while in the meantime there are many single points of failure in each layer. Finally, the most significant problems are connected with the security and the integrity of the system as a correct shuffle proof protocol should be chosen, while privacy is not guaranteed, because all votes get public, something that may lead to serious privacy issues (due to the order of the candidates in each ballot, etc.).

2.3.3 Homomorphic Systems

Homomorphic systems are another famous solution to the voting problem. An homomorphic voting system is based on an homomorphic encryption scheme [40]. We formally define homomorphic encryption and then explain what it means.

Definition 2.3.1. Homomorphic encryption: We say that a system is homomorphic if given two ciphertexts $c_1 = Enc(pk, M_1)$ and $c_2 = Enc(pk, M_2)$ and an operation $\cdot$ then:

$$c_1 \cdot c_2 = Enc(pk, M_1) \cdot Enc(pk, M_2) = Enc(pk, M_1 \cdot M_2)$$

and $Dec(sk, c_1 \cdot c_2) = M_1 \cdot M_2$

What the above definition means in simple words, is that an operation that is performed to the ciphertexts is transferred to the plaintexts, after the decryption. This is a very useful property especially in e-voting systems, as if the ciphertexts are, for instance the encrypted votes, and the operation is the addition then we can perform a sum of the ciphertexts and then decrypt them, achieving, thus, privacy of each vote. Furthermore, other than the simple homomorphic definition we gave above, there is the notion of fully homomorphic encryption (FHE) [41]. In such a system, any function performed at the ciphertexts results in the same function in the plaintext: $f(c_i) = f(Enc(pk, M_i)) = Enc(pk, f(M_i))$. FHE schemes are relatively recent, but FHE is a new domain worth investigation, but with many constraints.

Back to simple homomorphic systems, there are many e-voting systems based on this e.g. Helios [42], while DEMOS [43] is also based on homomorphic encryption. Homomorphic systems are generally faster than mix-net based systems. However, they, also, have some problems, for instance the EA can still learn any individual vote by just decypting it. This problem is generally tackled by distributing the decryption to multiple EA. The biggest problem with this kind of systems, is that each vote now requires a ZKP of correctness, since it is impossible to know what each voter has voted.

Therefore, many other factors should be taken into consideration. One of the most significant is that due to the ZKP required, the voter should perform, some additional, potentially complex computations. Thus, these systems may become complicated and impractical, or they may compromise the integrity of the process if these computations are left for an unsupervised program [44]. This problem, among others, is solved by DEMOS, as we will present in the next chapter.

Chapter 3

DEMOS Protocol

In this chapter we describe the DEMOS protocol, on which our work is based. Furthermore, the main contributions of DEMOS, are stated and explained, alongside with the novel techniques it uses. In this dissertation, we focus on the first implementation of DEMOS [43]. However, there are multiple papers about this protocol, like d-DEMOS [45] which is actually the distributed version of DEMOS, and DEMOS-2 [46] that is based on DEMOS-1 but has some performance enhancements for practical reasons. However, we extend the initial DEMOS paper which from now on, we will call DEMOS-1, for simplicity. In the next section, we will present the core ideas of the system and which exactly novel characteristics make it special.

3.1 Overview - Main Idea

DEMOS-1 is an E2E verifiable system in the standard model. It achieves this verifiability due to some novel techniques and does not depend on Random Oracles (RO) [47] or other setup assumptions. It is information theoretically secure and verifiable, under some assumptions we stated in the previous chapter (DDH, DLOG assumptions). DEMOS-1 is also a receipt free system, enhancing the privacy and its coercion resistance.

Furthermore, DEMOS-1 is a very practical system, since the users/voters do not require to perform any complex operations. They just select their choices, and all the proofs, are handled by the EA. Finally, in DEMOS-1 there is a novel technique by which the voters contribute 'random bits', in order to create a random challenge, for a sound ZKP.

But, how are all these achieved by DEMOS-1? We present in some bullet points the core idea and a simple example, of how the system actually manages to accomplish what it intends.

- The EA creates the election, by producing n ballots for n voters. These ballots have two parts, one for voting purposes, and the other for audit. Each ballot is shuffled and encrypted and committed to the BB, so as to be private for each voter. Finally, each voter receives his (decommitted) ballot and is ready to vote. The above process, is a very simplified version of the setup protocol in DEMOS-1, without all the cryptographic notions, we will later describe further.
- The voter receives his ballot, picks one of the two parts, and makes a choice on the anonymized ballot by choosing one (or potentially more) choice. He then sends his choice to the EA. This process, is the simplified version of the cast protocol.
- The EA after receiving all the choices from the users, and when the cast phase is over, begins the tally phase. Now the EA gathers all the encrypted commitments to the ballot, where they are homomorphically combined, so as to get the final result. The exact process of how this happens will be explained in much more details in the next sections.
- The final result is calculated and is made publicly verifiable, for everyone.

Above we gave a rough description of the DEMOS-1 system. In the following sections we analyze the contributions of DEMOS-1, as well as some key points in the design of the system. We also present the whole protocol so as to clarify what exactly we manage to do in our dissertation.

3.1.1 Notation in DEMOS-1

Before the analysis that follows in the next sections, we state some important notation that the authors of DEMOS-1 use. Below, we present some important points that someone should keep in mind, in order to understand the work that follows:

- The encryption scheme, or better the commitment scheme is based on lifted ElGamal over elliptic curves [48]. Everything is very similar to what we have already defined in the previous chapter about the standard ElGamal, however, in-

stead of $Enc(pk,m)$ we rename that to $Com_{ck}(m;r)$ $(=(g^r, g^m h^r))$ where $pk = ck$ and r is the randomness of the encryption. The generation algorithm is similar, with the only difference now being in the decryption part where instead of receiving m we receive g^m, and from there we again extract m (normally by 'brute forcing' over all the possible values for m). Of course, now the system is additively homorphic under multiplication as it holds that:

$$c_1 \cdot c_2 = Com_{ck}(m_1;r_1) \cdot Com_{ck}(m_2;r_2) = Com_{ck}(m_1+m_2;r_1+r_2)$$

- The e-voting system consists of n voters denoted by $\mathbb{V} = \{V_1,...,V_n\}$, m candidates denoted by $\mathbb{P} = \{P_1,...,P_m\}$ and the possible selection set is $\mathbb{U}$ which holds all the subsets of candidates that a voter is allowed to vote, while there is a security parameter λ, and $m,n = poly(\lambda)$.
- As it was previous shown there are five main algorithms the **Setup(), Cast(), Tally(), Result(), Verify()**, each of which will be defined later.
- The **E2E Verifiability** is proven through an E2E Verifiability game that, as well as the **Voter Privacy** is proven through a voter privacy game, which is based on the receipt-freeness of the system.

3.2 The Protocol

An overall presentation, of how all the protocols work, follows. We will try to explain the significant parts, emphasizing at some ideas that make this system special. The presentation will be made phase by phase, while we, mainly, try to explain how things work, and not dive into details, as this will be made later in our protocol presentation:

1. **Setup:** During this phase, the EA initiates the election. It produces, n double ballots, and assigns a tag to each of them. For each of the ballots, it chooses unique vote codes for all of the possible candidates, so overall there will be $2nm$ unique vote codes. Now, the ballots are ready, with each consisting of the tag and the two parts with the vote codes. Moreover, the EA chooses two random permutations for each ballot (one for each side of a ballot), so $2n$ permutations, in total. The EA should 'hide' the ballots and 'commit' to the values. This is done, by selecting $2mn$ random numbers $\{t\}$, where each one is used as the randomness of a commitment. Then, for each side of each ballot the chosen permutation is

performed, by which each side of the ballot will be shuffled and encrypted. Thus, finally, we have a shuffled, encrypted ballot consisting of the unique vote codes in committed form. Furthermore, with the same permutation chosen, another $2mn$ random numbers $\{r\}$ are selected, with which the values that represent each candidate in the homomorphic tallying, are shuffled and encrypted. In DEMOS-1 context, for candidate P_j the value that represents him is $(n+1)^{j-1}$. This value is the one needed, as each candidate P_j may receive at most n votes, and therefore, the result of the received votes can be extracted for each candidate separately, by performing just one homomorphic addition, as we will see in the **Result** phase. Finally, in order for this phase to be over, the EA, computes and commits to the values of the first step of a Σ - protocol for each candidate commitment, so as to perform nm ZKP for all the ballots later. Actually the number of ZK-proofs are kmn ,as we will see in the next section, since m ZKP are performed for the ballot side selected, as the other side will be completely opened and does not need a ZKP. The Σ - protocol will be stated later, alongside with the novel approach in each random challenge generator. In conclusion, the tags of each ballot, the committed vote codes of each ballot, alongside with the committed values and the commitments for the ZKP are all made public, by posting them to the BB.

2. **Cast:** Each voter receives the decommitted personal ballot, and chooses one of the two sides, by which he will vote. He then finds the vote code that corresponds to the preferred candidate, and casts his vote, which consists of **his tag, his choice of the side of the ballot, and the vote code**. The voter keeps the not selected part of the ballot, as a receipt, which can be used to ensure that the values in this part (which are opened later) are what they should be. It is noted, that despite this being a receipt, it is not a normal receipt, as it leaks no information, regarding to what the voter actually voted. So, that is why the protocol is called receipt free, despite having this kind of receipt.

3. **Tally:** After receiving all the votes, the EA, sends to the BB for each voter, the vote code chosen, alongside with the decommitted side of the ballot **not** chosen from the voter with all the randomnesses that were used for the commitment. This is done, so as anyone to be able to check that the one side of the ballot, that reveals nothing about the choice of the voter, properly shuffles and encrypts what it is supposed to. Furthermore, the EA finds out to which ciphertext corresponds

each vote code, and places them to a tally for the homomorphic multiplication. **While the ciphertexts, alongside with their encryption randomnesses, that belong to the part not selected are also added to another set, again for proving that they were properly shuffled and encrypted the same way as the vote codes**. Finally, based on each bit contributed by each voter from the random selection of the side of the ballot they voted, the challenge of the ZKP is extracted. We note that, the voters' coins/bits are considered based on their serial numbers, because otherwise a malicious EA can schedule some malicious votes to be cast whenever it wants and drop the minimum entropy to a logarithmic value of honest users. Afterwards, the third step of the Σ - protocol, which proves that the values on the side that the candidate voted are correct (belong to a set of valid values $(n+1)^j$), is made public and available to anyone. Other than the above, the sum of the randomnesses of the homomorphicaly multiplied ciphertexts is given, alongside with the actual decommitted value of the homomorphicaly multiplied tally, so as anyone to check the correctness of decryption of the tally.

4. **Result:** The result can be easily acquired from the above decommitted value of the homomorphicaly multiplied tally, since supposing that candidate P_j was chosen by x_j voters, then the decommitted value will be equal to $\sum_{i=1}^{m} x_i(n+1)^{i-1}$. So, by repeatedly 'modding' by $n+1$ and then dividing by the proper value, at the j-th repetition of the above we get $x_j = X \mod (n+1)$, and $X = \frac{X-x_j}{n+1}$.

5. **Verify:** In this stage the verification, of what we described above, takes place. We specifically noted, what is important to be verified in each of the steps above. Despite those, we will precisely state every verification process during the explanation of our enhanced protocol. For now, it is just reminded that since the system is publicly verifiable, the verification process is open to anyone who has access to the data published to the BB.

Above, we explained the whole process of DEMOS e-voting protocol. What is more, we re-state some fundamental intuitions that achieve the desired properties of DEMOS protocol:

- The protocol is receipt-free since the proofs of correctness are handled by the EA.

- The EA is considered malicious in the verifiability game, and honest in the privacy. However, privacy can be amplified by distributing the system (e.g. in d-Demos [45]).

- In addition to the above, the system is very simple from client (voter) perspective, as he only chooses and casts a simple code, with no computational requirements.

- The correctness of the encrypted values is satisfied by two facts. Firstly, all unused ballot parts are fully opened and due to the binding of the system we are sure about their values. Secondly, a novel Σ - protocol is given, so as to ensure that no malicious values have been given, as this protocol ensures that all the committed values in the 'voted' belong to a specific set: $\{(n+1)^j\}_{j=0}^{m-1}$. More details regarding the ZK-proof that it is used, as well as its novel way of producing the verifier's challenge, are given in the next subsection.

3.2.1 Σ - Protocol and the Verifier's Challenge

We will not explain thoroughly the Σ - Protocol of DEMOS-1, but we will just state how it works and what it achieves. The Σ - Protocol that is used here, is a 3-step ZKP, which proves that a value encrypted in a ballot, corresponds to a commitment to some value in $\{n+1\}_{i=0}^{m-1}$. In most 3-step protocols, during the second step, the Verifier should produce a random challenge as a response to the commitment to some values from the Prove. However, this random verifier's challenge is not a trivial task, since most e-voting systems should be non-interactive, as there must be no communication between the voter and the EA.

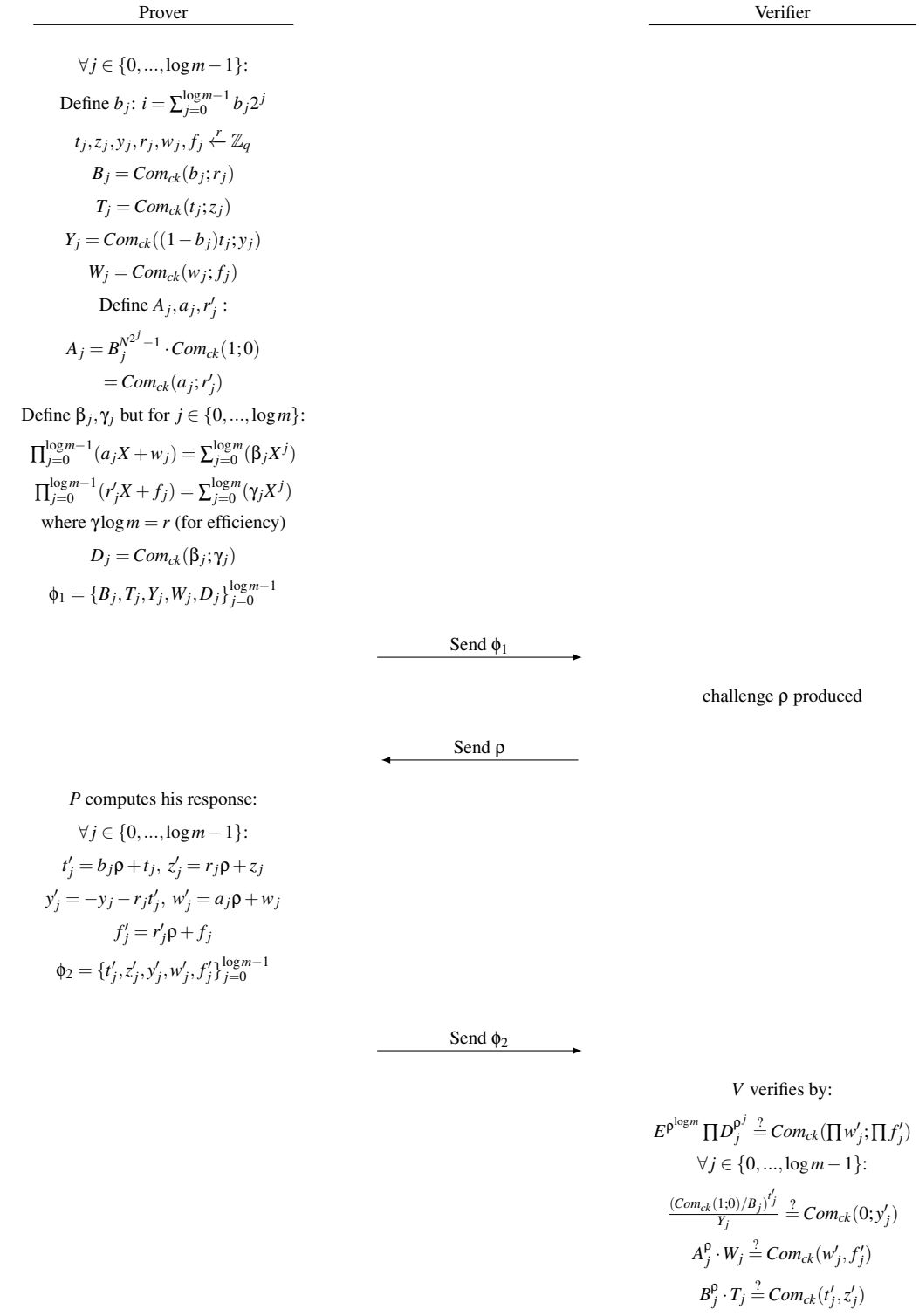

Figure 3.1: Demos Σ - Protocol as shown in DEMOS-1

The main contribution of this paper, is the novel approach in producing the verifier's challenge. Usually, a call to a RO is used so as to produce this 'random' challenge. This random challenge is a fundamental part in every e-voting system that has a ZKP.

In DEMOS-1, this ZKP is prepared by the EA when it has to prove that the ballots encrypt the value that are supposed to (each commitment can be a commitment to the set $\{(n+1)^{j-1}\}$, as we have seen before. Therefore, a challenge should exist such that the protocol is sound and secure.

This challenge, is extracted through the random bits that each voter contributes when he chooses on of the two sides of the ballot (bits 0/1). Supposing that the voters are n, then we have n random bits. However, this is not a real scenario, since the EA might be malicious, and could corrupt some voters. Therefore, it is required that there is a mechanism that will ensure, that a random challenge is produced. Many, such protocols have been proposed [49], but they have some restrictions regarding the maximum number of random bits that may be produced [50] (up to $\frac{n}{k}$). Therefore, in DEMOS-1 the condenser approach [51] is used.

Based on this approach, the authors of DEMOS-1 use a ZK amplification technique, where they segment the whole challenge into k blocks, getting as a result k sub-challenges $\{a_i\}_{i=1}^{k}$. Then the Σ - protocol is run k times per commitment, and should produce k valid ZK-proofs for each commitment. As a result, it is proven that the probability that an EA successfully cheats all k proofs drops exponentially with $\theta - k(\log\log m + 1)$. The above scheme ensures that a commitment on the side of the ballot the voter chose to vote, belongs to a value in $\{n+1\}_{i=0}^{m-1}$.

The above system does not ensure, though, that all the values are different, or that each value is the right one (for candidate P_j the decommitted value must be equal $(n+1)^{j-1}$). A malicious EA might try to change the values on the side that the voter is going to choose, and leave the not selected side correctly unaffected. However, if the EA tries to cheat and guess for a specific voter the right ballot side, the probability of such an event is equal $\frac{1}{2}$ (considering that the voter will select randomly between the two random sides). But even in this case, as it is proven in DEMOS, the difference will be just one vote, while the EA will be caught with probability $\frac{1}{2}$. Thus any significant variation of at least d votes will be caught with probability $1-(\frac{1}{2})^d$. In the paper of DEMOS-1 a full analysis regarding the clash and the modifications attacks is given, which we will not restate here.

3.3 Problems and Open Questions

In the previous section we described how the DEMOS-1 protocol works, and what are its novel contributions. However, the DEMOS-1 protocol is not perfect and has many problems and open questions, which we will present here.

- The DEMOS-1 protocol faces some efficiency problems, because of all the heavy computations performed by a single EA. This is improved in the later publications of d-DEMOS and DEMOS-2, but it is not solved.
- For a small number of voters n, the system will not work correctly, as if an EA can brute force or guess with non-negligible probability the challenges, then the Σ - Protocol will be susceptible to adversarial attacks. Therefore, the number of voters, as well as the number of potential malicious voters (min-entropy of the challenge), should be taken into consideration.
- While we require a not small amount of voters, on the other hand, we require that the number of voters to be limited. Actually, the DEMOS protocol, has an upper bound that the relation of number of voters n, and the number of choices m, and this is: $n \cdot (n+1)^{m-1} \leq |M|$ which means that $n \cdot (n+1)^{m-1}$ should be always inside the message space, which in general cases limits us a lot, e.g. for

  $$n = 10^6, m = 40 \Rightarrow n * (n+1)^{m-1} \approx 10^{240} = (10^3)^{80} \approx 2^{800}$$

  The system is not practical as the order q should become huge (≈ 800).
- For now only approval voting elections are supported from the system. This kind of voting is not always sufficient, since there are many other possible schemes that cannot be currently supported with the existent protocol.

The above problems have driven us to propose a new enhanced DEMOS protocol, with a new tally mechanism. This protocol will be thoroughly explained in the next chapter, alongside with its new improvements and the necessary proofs.

Chapter 4

New Enhanced Demos Protocol

4.1 Overview

In this chapter we will present the new enhanced DEMOS protocol. We transform the existing DEMOS protocol, enhancing it and including different type of voting schemes, other than approval. Initially, we provide the changes we perform to the old system so as to clarify that the core ideas remain unchanged, while we perform a few important modifications. Furthermore, we describe the new form of our system by providing the new **Setup, Cast, Tally, Result, Verify** protocols, following the notation of the initial paper [43]. Finally, we provide the new proofs of security, where required, so as to ensure that nothing has been compromised, while we present an example of the new system and how it functions.

4.2 Modifications and Achievements

In this section we describe what we modified from the old system and what we achieved with these changes. Our main contribution is a systematic way to perform borda count, and more generally preferential elections, where each voter can order the candidates in a preferred order, and then each candidate will get a specific value based on this order. We remind here, that in the simple borda count the m candidates are ordered by each voter, so as the first selection to get $m-1$ points, the second $m-2$,... and the last 0. Of course, this ranking of $\{0,...,m-1\}$ is not binding and we will see that any ranking is available in our system.

Moreover, we provide new proofs of correctness where required, while we also maintain the backbone of DEMOS-1. Furthermore, other than the borda count, we also provide evidence that generalized score voting is available in our system if we restrict the ranking to be some integer value in some range. Finally, we provide a theoretical way of implementing generalized STV voting systems, but this implementation is purely theoretical and is not the same as the previous ones, since now we do not provide the necessary proofs (ZKP, encryption system), but just a concept of how to implement STV, if we use FHE. The reason why we do not try to implement the STV case, is mainly because FHE is not practical with the current known protocols but in the future this might be possible [52].

We will now present the main changes we perform to the current protocol. We, briefly, state the reasons and the results of each of the changes. So, the main modifications are:

- The main difference of our system, is that **instead of just one tally we have** m **different tallies, one for each candidate**. Furthermore, in the new system **when a voter makes a choice, he does not choose the candidate, but he chooses the value (ranking) this candidate will receive**.
- From the above, it follows that **the voter will not just send one vote code but** m. In simple borda count, this m vote codes will be different and will represent all the different values. Therefore, we will have m tallies, where each one will have the individual result for each candidate.
- We provide a new ZKP, so as the system to continue functioning with the new parameters. Actually, we provide two approaches, where in the first one, we use the same Σ - protocol used in DEMOS-1, but by providing a completely new proof concept, while in the second approach we use a ZKP of a shuffle so as to accomplish what we need. We decide to provide both approaches, because we will notice that they have some trade-offs with each other.
- The values encrypted are also changed, but we use the same encryption scheme (lifted ElGamal). In our system, we choose to encrypt values in $\{0,...,m-1\}$ or $\{l^0,...,l^{m-1}\}$ (l will be defined later), instead of $\{(n+1)^0,...,(n+1)^{m-1}\}$. This way, we efficiently increase the maximum number of voters/candidates, and solve the scaling issues of DEMOS-1. Generally, it is stated that the committed value for a candidate will depend on the approach of the ZKP.

The above modifications, alongside, with the respective changes in their proofs are

the main contributions of our dissertation, and in the following sections, we present thoroughly how the new system is configured.

4.3 The ZKP Approaches

There are two main approaches regarding the new enhanced protocol of DEMOS. In the following sub-sections we define both, and explain how they will function in our system. We note here that DEMOS-1 the Σ - protocol uses $O(\log m)$ operations per commitment (exponentiation operations). Thus in total $O(m \log m)$ operations needed per ballot. The best known shuffle proofs require $O(m)$ exponentiation operations. However, the practical difference is not that big since m is generally small in e-voting, while DEMOS-1 Σ - protocol needs only one challenge in contrast to the more efficient shuffle proofs. which would require at least m.

4.3.1 Shuffle Proof

A ZKP of a shuffle would be ideal in our case. As we have previously defined, in a shuffle proof the Prover provides an argument about m ciphertexts, proving to a verifier that they are indeed a valid shuffled re-encryption of the initial set of m ciphertexts. In our context, we require a 3-step protocol since it is an e-voting system, while, *ideally* the challenge should be a single one, so as to perform the same ZK amplification technique that it is performed to the current DEMOS.

However, after our research we concluded that there were no shuffle arguments with just a single challenge, since almost all 3-step shuffle proofs require at least m challenges (maybe in the future there will be). The only shuffle proofs with a single challenge are 4-step protocols (section 3 from [53]), which are not feasible in our case. Thus we need a technique, to manage and get m challenges from the total bits contributed by the voters. Then, we can use the shuffle proof of Jun Furukawa (section 3 from [54]), which is a relatively simple shuffle proof, with the proper transformation in the protocol. The whole process will be the following:

– The EA produces the ballots, but now instead of committing to the value $Com_{ck}((n+1)^{j-1}; r_j)$, the commitment is to the value $Com_{ck}(j-1; r_j)$. So, the EA for a single side of a ballot has produced $\{c_j\}_{j=0}^{m-1} = \{Com_{ck}(j-1; r_j)\}_{j=0}^{m-1}$

each of which correspond to a commitment to the possible rankings of the system (for borda count these values are $\{j-1\}_{j=1}^{m}$). The EA then produces $\{c'_{j'}\}_{j'=0}^{m-1} = \{Com_{ck}(j'-1; r'_{j'})\}_{j'=0}^{m-1}$ where $j' = \pi(j)$ (the permutation performed). Then, the EA provides the $\{r_j\}_{j=0}^{m-1}$ so as everyone to know that the initial $\{c_j\}_{j=0}^{m-1}$ indeed commit to the correct values. Finally, the EA should prove that the $\{c'_{j'}\}_{j'=0}^{m-1}$ are a valid permuted re-encryption of $\{c_j\}_{j=0}^{m-1}$, and this is done via the shuffle proof argument that follows.

– In the first step of the protocol, the prover P (EA in our case) will produce normally all the commitments according to the protocol, and will name all the batch of them as ϕ_1 following the notation in DEMOS-1. The exact commitments are shown in the paper [54] and we do not present them here. Of course we produce k commitments ϕ_1 so as to execute the protocol k times, in order to have the ZK amplification.

– At the second step of the protocol, we segment the ρ challenge in m different challenges $\{\rho\}_{i=1}^{m}$.

– The third step of the protocol, is similar to the current each of the individual challenges is segmented in k pieces and the protocol is executed k times, so as to have the ZK amplification, and we receive k values of ϕ_2.

With the above technique we provide a way to overcome the efficiency problems of the current Σ - protocol. Moreover, **we also solve the problem with the maximum values of** m,n, since the current system can support elections with n,m s.t. $n \cdot (m-1) < q$, where q now is a vast number. However, this implementation has **some problems**. Initially, we have to have **too many bits contributed by the voters, as now we actually partition the voters' coins to** km **blocks instead of just** k. Therefore, the value of n should be big enough so as to have enough bits (and its corresponding entropy). It may be shown, that our ZK amplification may be conducted with less blocks or less repetitions, but that is a tricky task. Finally,**even the current implementations should have a proof of correctness, that the initial partition to the** m **blocks and then each of them to** k **suffices so as to have a sound system**.

Therefore, we provide a novel technique where we manage to overcome this problem in the next subsection. We actually use the current proof setting of DEMOS-1 and produce some additional proofs that ensure the soundness of the new system.

4.3.2 Transforming the Current Σ - Protocol

We offer **a novel approach so as to implement the borda count (or any similar preferential or scoring system) in the context of DEMOS**. The core idea of our implementation is that while we still use the current Σ - Protocol so as to ensure that each of the commitments is indeed a commitment to a value belonging to a specific set. We will also provide **a proof that the homomorphic combination of all the commitments commits to a specific predefined value**. If for instance, we consider that the ciphertexts commit to values in $\{0,...,m-1\}$, then the homomorphic combination of all the ciphertexts commitments should be equal to $\frac{m(m-1)}{2}$.

On the other hand, by just proving that the homomorphic combination is equal to $\frac{m(m-1)}{2}$ does not mean that the values commit to the values they should. For example, a malicious EA can choose to commit all, but two, values correctly, and for the specific two ones, suppose k,l $(0 < k,l < m-1)$ the EA could commit to $k-1, l+1$. Then, the homomorphic sum of the plaintexts would, indeed, be equal to $\frac{m(m-1)}{2}$, while the proof of the Σ - Protocol would also work, since $k-1, l+1 \in \{0,...,m-1\}$.

Therefore, we should think of such values, such that, their homomorphic sum and the fact that all values belong to a specific set, means that all the values will be distinct. The problem can be expressed in the following way.

1. Suppose that the possible values belong to the set $\{x^0,...,x^{m-1}\}$ (which is proven by the Σ - Protocol).

2. Then, suppose that there are exactly m such values chosen from that set, with the only requirement that their sum is equal to $\sum_{i=0}^{m} x^i$.

3. Finally, suppose the following setting, where an EA chooses values $\{a_0,...,a_{m-1}\}$, where a_i, is the number of objects chosen for the value x^i. It must hold that $a_i \in \mathbb{Z},\ a_i \geq 0$, since a value can be chosen 0 or more times, and $\sum_{i=0}^{m-1} a_i = m$ since we need m choices, exactly.

From the above setting, we should find an x, that ensures that all $a_i = 1,\ \forall i \in \{0,...,m-1\}$ is the only solution.

We should now express mathematically what we described above. We are, actually, looking for a value $x \in \mathbb{Z}^{>}$ (actually $x \geq 2$, since for $x = 1$, we have a trivial useless

system, as everything is encoded as 1) s.t.:

$$a_i \in \mathbb{Z}^{\geq}, \tag{4.1}$$

$$\sum_{i=0}^{m-1} a_i = m, \tag{4.2}$$

$$x \in \mathbb{Z}^{\geq}, \tag{4.3}$$

$$\sum_{i=0}^{m-1} a_i x^i = \sum_{i=0}^{m} x^i, \tag{4.4}$$

$$(4.1), (4.2), (4.3), (4.4) \Rightarrow a_i = 1,\ \forall i \in \{0, ..., m-1\}$$

From the above equations, $(4.1), (4.2) \Rightarrow a_i \in \{0, ..., m\}$, since any larger value would violate (4.2). Now having as a starting point the (4.4) we find the minimum value of x s.t. our requirement is fulfilled. And we have:

$$\begin{aligned}
&(4.4) \equiv \sum_{i=0}^{m-1} a_i x^i = \sum_{i=0}^{m} x^i \Rightarrow \\
&a_0 + a_1 x + ... + a_{m-1} x^{m-1} = 1 + x + ... + x^{m-1} \Rightarrow \\
&(a_0 - 1) + (a_1 - 1)x + ... + (a_{m-1} - 1)x^{m-1} = 0 \Rightarrow \\
&(a_{m-1} - 1)x^{m-1} = -((a_0 - 1) + (a_1 - 1)x + ... + (a_{m-2} - 1)x^{m-2})
\end{aligned} \tag{4.5}$$

From (4.5), we can distinguish 3 cases, (a) $a_{m-1} - 1 > 0$, (b) $a_{m-1} - 1 < 0$ and (c) $a_{m-1} - 1 = 0$. Thus, we have:

$$(4.1) \overset{(a)}{\Longrightarrow} kx^{m-1} = -((a_0 - 1) + (a_1 - 1)x + ... + (a_{m-2} - 1)x^{m-2}),\ k \in \mathbb{Z}^{>}$$

However,

$$
\begin{aligned}
&-((a_0-1)+(a_1-1)x+...+(a_{m-2}-1)x^{m-2}) \leq \\
&|-((a_0-1)+(a_1-1)x+...+(a_{m-2}-1)x^{m-2})| \leq \\
&|a_0-1|+|a_1-1||x|+...+|a_{m-2}-1||x^{m-2}| \leq \\
&\max(|a_i-1|)(1+|x|+...+|x^{m-2}|) \overset{x>0,\ a_i \leq m}{\leq} \\
&(m-1)(1+x+...+x^{m-2}) = \\
&(m-1)\frac{x^{m-1}-1}{x-1} \qquad (4.6) \\
&x \geq 2 \Rightarrow \frac{x}{2} \geq 1 \Rightarrow -\frac{x}{2} \leq -1 \Rightarrow \\
&x-\frac{x}{2} \leq x-1 \Rightarrow \frac{1}{x-\frac{x}{2}} \geq \frac{1}{x-1} \\
&\Rightarrow \frac{x^{m-1}-1}{x-\frac{x}{2}} \geq \frac{x^{m-1}-1}{x-1} \Rightarrow \\
&\frac{x^{m-1}-1}{x-1} \leq \frac{2x^{m-1}-2}{x} \Rightarrow \\
&\frac{x^{m-1}-1}{x-1} < \frac{2x^{m-1}}{x} = 2x^{m-2} \qquad (4.7) \\
&(4.6) \overset{(4.7)}{\Longrightarrow} -((a_0-1)+(a_1-1)x+...+(a_{m-2}-1)x^{m-2}) \leq \\
&2(m-1)(x^{m-2}) \qquad (4.8)
\end{aligned}
$$

In the meantime,

$$kx^{m-1} \geq x^{m-1} \qquad (4.9)$$

If we find an x s.t. equation (4.9) is always greater than equation (4.8), then the system will have no solution for any a_i, meaning that case (a) can never hold for such values of x. So,

$$
\begin{aligned}
&(4.9) > (4.8) \Rightarrow \\
&x^{m-1} > 2(m-1)(x^{m-2}) \Rightarrow \\
&x > 2(m-1)
\end{aligned}
$$

So for a value, of $x \geq 2(m-1)+1$, we proved that (a) has no x satisfying it. We will now consider case (b) where we have:

$$
\begin{aligned}
&(4.5) \overset{(b)}{\Longrightarrow} -kx^{m-1} = -((a_0-1)+(a_1-1)x+...+(a_{m-2}-1)x^{m-2}),\ k \in \mathbb{Z}^{>} \Rightarrow \\
&kx^{m-1} = (a_0-1)+(a_1-1)x+...+(a_{m-2}-1)x^{m-2}\ k \in \mathbb{Z}^{>}
\end{aligned}
$$

However, performing the exact same analysis as before, since again

$$(a_0-1)+(a_1-1)x+...+(a_{m-2}-1)x^{m-2} < |(a_0-1)+(a_1-1)x+...+(a_{m-2}-1)x^{m-2}|$$

For $x \geq 2(m-1)+1$ there is again no solution. So, for $x \geq 2(m-1)+1$, only case (c) is possible. But from (c), if $a_{m-1}-1=0$ then $a_{m-1}=1$, is the only solution. Following, with the same logic we prove respectively that if $x \geq 2(m-1)+1$ then the only possible solution is $a_0 = a_1 = ... = a_{m-1} = 1$.

Therefore, if we combine the Σ - protocol of DEMOS, alongside with the new proof above, then we will be sure that the values committed are m distinct values chosen from the set they should have been chosen. This is actually again a shuffle proof, but the committed values belong to a specific set. **The above result by its own, is a very important cryptographic result, and is useful further than the scope of this dissertation.** Our new proof will work in the following way. Suppose that the EA has to prove that the chosen side of the ballot indeed is a commitment to ciphertexts $c_{\pi(i)} = Com_{ck}(x^i, r_i)$, where $\pi()$ the permutation chosen for this side. Then, with the Σ - protocols, the EA will prove that each of the c_i commits to such a value, and then it will prove that their homomorphic combination is a commitment to $1+x+...+x^{m-1}$, by providing the two values

$$\left(\prod_{i=0}^{m-1} c_i = Com_{ck}\left(\sum_{i=0}^{m-1} x^i; \sum_{i=0}^{m-1} r_i\right), \sum_{i=0}^{m-1} r_i\right)$$

From the above everyone will be sure that the values committed are the permutation of the right ones. On the other hand, we can still not be sure whether the permutation is indeed correct since, despite proving the validity of a permutation, we cannot be sure that the permutation is the one the voter sees to his ballot, but again the EA will be caught with probability $1-\frac{1}{2^d}$, if she cheats d times. A further analysis will be provided in a next chapter, where we will prove the resilience of the system to such modification attacks. Finally, we should say that even the current DEMOS-1 Σ - protocol may be sufficient as a proof. Nevertheless, we provide the above shuffle proof argument because it provides an easier way of modeling the possible attacks, since now a malicious EA can only interchange the values between two commitments, while at the same the above shuffle proof argument is an important result that can be used in many other applications.

4.4 Description of the New System

We now present a simple borda count voting system, and how this will be performed in our new system. We have explained that a borda count voting is actually a voting where we prioritize m candidates in an order of preference. In the general case, these candidates get a score of $\{0,1,...,m-1\}$ according to the position they were placed by the voter. This is the case we present and investigate in our current implementation. However, this score can be easily changed and take values in any set we prefer, without loss of generality. Moreover **we present both cases, the first where a regular shuffle proof is chosen and the commitments are of the form $Com_{ck}(i,r_i)$, and the second where we use the Σ - protocol and the homomorphic combination proof, with commitments $Com_{ck}(x^i,r_i)$**. We also have to state that we proved that $x \geq 2m-1$, but x should also be greater or equal than $n+1$, so as the system to continue being correct, and the tallying to be feasible. Therefore, we set $x = \max\{2(m-1)+1, n+1\}$, so as both requirements to be fulfilled.

We note that in our case we could consider two Election Authorities, EA^1, EA^2, the first responsible for the initial setup and the second for the tally computations. That way everyone could be sure about the safety of the procedure. However, we follow the current DEMOS-1 notation and have just one EA, so our new protocol will be of the form:

- **Setup:** The **Setup** protocol takes the same input as before. More precisely, the security parameter: 1^λ, the voters: $\mathbb{V} = \{V_1, V_2, ...V_n\}$, and the list of candidates: $\mathbb{P} = \{P_1, P_2, ...P_m\}$. Again we have the PPT algorithms: (Gen, Com, Ver), as in the existing system. The new protocol is:

  1. $ck \leftarrow Gen(Param, 1^\lambda)$, $l \in \{1,...n\}$.
  2. The EA selects tag_l, the unique identifier for the l-th ballot.
  3. The EA selects permutations $\pi_l^{(0)}$, $\pi_l^{(1)}$ over $\{1,...,m\}$, so as to shuffle the order of the vote-code and the choices, in the two parts of the ballot, following partially the existing protocol. The difference now is the new Zero Knowledge Proofs, and the form of the ballots.
  4. The EA selects unique vote-codes $C_{l,j}^{(0)} \leftarrow \mathbb{Z}_q$ (resp. $C_{l,j}^{(1)}$) with $j \in \{1,...,m\}$. The m different codes are associated with the m different possible position/rank each of the candidate might get. As, it is shown in

the DEMOS-1 system the values are not necessarily randomly chosen from $\mathbb{Z}_q$, but they might belong to a (much) smaller subset of it, so as to be more user friendly. **More precisely, in our practical implementation and in the example section we provide a way of getting meaningful vote codes in a systematic way.**

5. The EA generates the ballot s_l consisting of two parts $s_l^{(0)}$, $s_l^{(1)}$ with each part consisting of:

$$s_l^{(a)} = \{(P_j, C_{l,j}^{(a)})\}, a \in \{0,1\},$$

and $s_l = (tag_l, s_l^{(0)}, s_l^{(0)})$

6. The EA computes $j' = \pi_l^{(a)}(j), \forall j \in \{1,...,m\}$, as the new indexes of the ciphertexts.

7. The EA chooses randomnesses $t_{l,j'}^{(a)}$. These randomnesses will be used to commit in permuted form, to the vote-codes we have previously generated as:

$$U_{l,j'}^{(a)} = Com_{ck}(C_{l,j'}^{(a)}; t_{l,j'}^{(a)})$$

8. Similarly to the above, EA chooses randomnesses $r_{l,j'}^{(a)}$, that will be used to actually encode the position/rank commitment, which in our case will be either a simple value in $\{0,...,m-1\}$ or a value in $\{x^0,...,x^{m-1}\}$ depending on the ZKP approach we choose. The commitments now will be either:

$$E_{l,j'}^{(a)} = Com_{ck}(j'-1; r_{l,j'}^{(a)})$$

if a typical shuffle proof approach is chosen or:

$$E_{l,j'}^{(a)} = Com_{ck}(x^{j'-1}; r_{l,j'}^{(a)})$$

if the our novel protocol is chosen.

9. Then, the EA would have to be prepared to prove that the $E_{l,j'}^{(a)}$ is a valid permuted re-encryption of the $\{0,...,m-1\}$ or of the $\{x^0,...,x^{m-1}\}$, with the ZKP techniques we described. So, it prepares the pre-audit data $\phi_{1,l,j'}^{(a)}$ which is the first step of the ZKP we have chosen that is either the commitments of a shuffle proof [54] or the commitments of the Σ - Protocol. The above proof, is crucial for the part that the voter is going to use for voting, since for the unused part the proof that the $U_{l,(i,j')}^{(a)}$ were encrypted with the same permutation, is ensured from the opening.

10. The *EA* publishes the public information. The form of the public information is very similar to the current DEMOS protocol. More precisely it is:

$$Pub = (ck, \mathbb{P}, \mathbb{U}, \{Pub_l\}_{l\in\{1,\dots,n\}}),$$

with

$$Pub_l = (tag_l, \{(U^{(a)}_{l,j'}, E^{(a)}_{l,j'}, \phi^{(a)}_{1,l,j'})\}^{a\in 0,1}_{j'\in\{1,\dots,m\}})$$

The secret key of *EA* will be:

$$msk = \{Pub_l, s_l, msk_l, state_{\phi,l}\}_{l\in[n]}$$

with:

$$msk_l = \{(C^{(a)}_{l,j}, t^{(a)}_{l,j}, \pi^{(a)}_l(j) = j')\}^{a\in 0,1}_{j\in\{1,\dots,m\}}$$

The state information includes the values that are private in the first part of the ZKP (the witness of the ZKP), and are needed from the EA to complete the protocol.

- **Cast:** The Cast protocol is also similar to the existing protocol, but with some main differences. The voter V_l flips the coin and chooses the $a_l \leftarrow \{0,1\}$, and selects the $s_l^{a_l}$ to vote and the $s_l^{1-a_l}$ for audit. Only now, **instead of just one vote-code, V_l sends m.** Suppose now that V_l has an order of preference ω_l over the m candidates, meaning that he considers the candidate P_1 as his $\omega_l(1)$ favorite choice, the candidate P_2 as his $\omega_l(2)$ favorite choice, and so on. So, he will arrange all vote codes in an order of preference and then cast them as $(\omega_l(j) = j'')$:

$$\psi_l = (tag_l, a_l, \{C^{(a_l)}_{l,j''}\}^m_{j''=1})$$

and he sends it to the *EA* which updates its state.

- **Tally:** The Tally process is very similar to the existing protocol. Only, now we have m different tallies, one for each candidate. The whole process is the following, supposing that $\bar{\mathbb{V}} \subseteq \mathbb{V}$ is the set of voters who voted successfully and correctly:

  1. *EA* uses (tag_l, a_l) and finds $s_l^{1-a_l}$, sending it alongside with ψ_l to the BB for each voter in $\bar{\mathbb{V}}$.

  2. The BB is also updated by opening all the vote-code commitments, and sending all the pairs $\{(C^{(a)}_{l,j}, t^{(a)}_{l,j})\}$ to the BB.

3. For each vote ψ_l, the *EA* does the following:

   (a) For each of the m vote codes $C_{l,j}$, she finds the cast vote-code that matches the $C_{l,j''}^{(a_l)}$, and finds and adds the corresponding commitment $E_{l,\pi_l^{(a_l)}(j'')}^{(a_l)}$ to the $\mathbf{E}_{tally}^i$ set. There are m - tally sets and each $\mathbf{E}_{tally}^i$ set corresponds to the values that the i-th candidate will get.

   (b) For each ballot *EA* places all the $\{E_{l,j}^{(1-a_l)}\}_{j\in\{1,...,m\}}$ to the $\mathbf{E}_{open}$ for the audit part from voters.

4. The *EA* produces the verifier's challenge of the 3-step protocol. If the shuffle proof approach is chosen we follow the directions from the chapter 4.3.1, and if our novel approach is chosen the challenge is produced according to the prescriptions of DEMOS-1.

5. After the production of the challenge the EA, moves on the third step of the protocol, where the response to the challenge is produced, for all the sides of the ballots, that were chosen from the voters. The response is denoted as ϕ_2. Moreover, if our ZKP is chosen then the EA should also give the sum of the randomnesses for the chosen sides' commitments of the ballots. The total randomnesses are denoted as $\{Q_i\}_{i=1}^n$, with $Q_i = \sum_{j=1}^m r_j^{(a_l)}$.

6. The *EA* combines homomorphicaly all the $i \in \{1,...,m\}$ tallies by performing:

$$\mathbf{E}_{sum}^i = \prod_{E\in\mathbf{E}_{tally}^i} E$$

   All these sums are the aggregated results for each of the m candidates. **The only thing that differs, depending on our implementation, is how these results are interpreted and decoded.** The final m results are then decrypted each producing a result T_j with total randomness R_j for $j \in \{1,...,m\}$. These results, alongside with their total randomness are also sent to the BB, so as everyone to be able to verify that the value was decrypted correctly.

7. The *EA* sends to the BB the $\mathbf{E}_{open}$ alongside with the decommitted information (randomnesses and values encrypted), which are all denoted as Open.

- **Result:** If the shuffle proof is chosen then the production of the result is an immediate outcome of the decryption of each individual E_{sum}^i. However, for the

second approach the result is produced and interpreted similarly to the current system:

Algorithm 1 Algorithm for producing the results

1: $Res_i \leftarrow T_i$
2: **for** $j = 1$; $j < m$; j++ **do**
3: $\quad s_i \leftarrow X \mod (x)$
4: $\quad Res_i \leftarrow \frac{Res_i - s_i}{x}$
5: **end for**
6: **return** $\{s_1^{(i)}, s_2^{(i)}, \ldots, s_m^{(i)}\}$

An $s_j^{(i)}$ contains the value of how many times the candidate P_i was voted as the j-th choice in total. It is apparent that with this encoding we can perform any variation of borda count and generalized preferential voting system, as the value that each position gets is not connected to the encoding.

- **Verify:** The verification process is the following (the process is almost identical to current DEMOS setting):

  1. There are n different ballots, and each one has a different, distinct tag. Moreover, all $2nm$ vote codes are also distinct and different with each other.
  2. All the not selected parts of the ballots are opened. No selected part is opened. Therefore, each vote code that has been cast should not be opened.
  3. All the 3-step ZKP are valid. Moreover, the homomorphic combination of the commitments of each ballot (selected), is a commitment to $1 + x + \ldots x^{m-1}$.
  4. All the openings of the not selected commitments are correct. Also, the order of the commitments when opened (the value that they are supposed to encrypt) is indeed the one the EA claims.
  5. The final $\mathbf{E}_{sum}^{i}$ of each tally is indeed the product of everything it contains.
  6. Each vote ψ_l contains m exactly vote codes, the one after the other, representing the order of preference (or just the rank) of each of the candidates.
  7. All the vote codes that are cast in a single vote ψ_l, are part of the ballot with $tag = tag_l$.

Regarding score voting, the process is very similar to this one. The only difference is that now we are given m' choices per candidate, where m' is the number of different 'score' options (mm' vote codes per side of the ballot, in total). So, m different values are chosen, one from each of the m' different 'score' choices, with no additional requirement.

From now on, we will continue and refer to the new system while taking into account that we use our ZKP which is a hybrid shuffle proof. We do this so as to perform the proofs under the same logic as shown in DEMOS-1. However, even with the typical shuffle proofs, the results are generally similar.

4.5 Correctness, Security and Verifiability

The proof of **correctness** is exactly the same as shown in section 3.6 of DEMOS-1 [43]. The only difference is that instead of considering $N = n+1$ we consider that $N = x = \max\{n+1, 2m-1\}$. Someone can understand that in most cases the value of $n+1$ is much greater than $2m-1$, but we do not get into such simplifications and that is why we denote this value as x. Since, the proof is exactly the same and relatively simple we omit it here and anyone can check it from the DEMOS-1 paper.

Furthermore, regarding the **verifiability** of the new system everything seems similar, with one important difference. In the new system the modification and the clash attacks can be more severe. That is the case, because while in DEMOS-1 each vote can cause a deviation of 1 in the final result, in our case, the deviation is equal to $\max(val) - \min(val)$. So, in the borda count, if the EA manages to cheat, (either by modifying the contents of a ballot and presenting a different permutation that the one the voter sees, or by performing a clash attack) just for 1 ballot the EA could cause a difference in result of $m-1$ ranking. Therefore it is required to model these kind of threats and show how our system handles them.

For simplicity, we only refer to the attack where a malicious EA modifies the contents of a ballot and presents a different permutation than the one the voter sees. In DEMOS-1 the relation of both attacks is shown, so we will not prove something that is already proven. We will study the scenario where the EA knows exactly what some voter V_l is expected to vote. Moreover, in our case we suppose that the final expected result of candidates $\{P_1, P_2, ...P_m\}$ is known and they will get a rank of

$\{R(P_1), R(P_2), ..., R(P_m)\}$. Then, suppose also that $\min(|R(P_i) - R(P_j)|)_{i \neq j} = d$, where P_i and P_j are the two candidates who are the closest in the final ranking. Finally, in the worst case scenario a malicious EA can find a voter V_l whose valuation for candidate P_i is $U^{(l)}(P_i) = m - 1$ and $U^{(l)}(P_j) = 0$, and just for this case $R(P_i) > R(P_j)$. Then, the malicious EA can simply perform an interchange of these two committed values that correspond to the values for rank i and j on one of the side of the l-th ballot. Then with probability $p = \frac{1}{2}$ the malicious EA can alter the rank of a maximum value so the $|R(P_i) - R(P_j)| = d - 2m + 2$ if $d > 2m - 2$ or to $2m - 2 - d$ if $d \leq 2m - 2$.

The above scenario is just an example of what a malicious EA can perform. If a malicious EA can find k such voters as V_l, then it can perform an attack causing a maximum deviation of $2k(m-1)$ votes but with expected deviation of $k(m-1)$. The EA will be caught with probability $p = 1 - 2^{-k}$ (k bernouli trials). Therefore we define the following.

Definition 4.5.1. We consider that our system is secure under security parameters k, d, s.t. if a malicious EA 'alters' k ballots the result will not change iff

$$d = \min(|R(P_i) - R(P_j)|)_{i \neq j} > k(m-1)$$

We state that for lower expected values of d the system will not be safe. We also claim that if the EA tries to corrupt more than k ballots, then she will be caught with high probability $p > 1 - 2^{-k}$.

The above definition is the most strict definition we could devise, since it makes many assumptions that are too difficult to be simultaneously fulfilled. The most difficult of them is that, the EA will know exactly what some voters voted, and the expected result. There are many ideas on how we could actually produce a more realistic and less strict definition. One alternative idea, is to define some partial knowledge of the EA and what some voters' sets are going to vote with the definition of some utility functions. Then, the EA will be able to 'guess' the result based on the utility functions and some external knowledge and will be able to modify some ballots. This kind of analysis is based on algorithmic game theory and is far beyond the scope of our dissertation (or any MSc dissertation). We just claim that in our definition we do not consider at all the correlation of the choices of a voter, and state that there are many alternatives that would actually show that the threshold values of k, d could differ a lot. Actually, our system could be shown to be much more resilient to the classical attacks than we just defined, if other asssumptions (than the worst case scenario) were taken into consideration.

Finally, **regarding the voter privacy of our system everything remains the same, as the receipt freeness remains intact and the privacy games are still in effect**. Actually, **the only main difference from the classical DEMOS-1, other than the ranking, is that now the EA provides the opening of the homomorphic combination of all the commitments in the selected side of a ballot.** However, this opening leaks no information regarding any partial randomness (under the DLOG assumption).

4.6 Example of Our System

We will now present a simple example of our system in practise. The numbers presented will not be accurate, but user friendly so as anyone to understand how it works. Suppose that in our system, we have $n = 3$ voters (V_1, V_2, V_3, V_4), and $m = 3$ candidates P_1, P_2, P_3, while we have a simple borda count. Therefore, during the **Setup** phase, four double-sided ballots will be created, each receiving one unique tag which for simplicity we set $tag_1 = 1, tag_2 = 2, tag_3 = 3, tag_4 = 4$, while for each ballot 6 vote codes will be created for each side. The form for the ballots will be:

Table 4.1: Ballots for sample Election

1	
$C^{(0)}_{1,1}$	1^{st} pos.
$C^{(0)}_{1,2}$	2^{nd} pos.
$C^{(0)}_{1,3}$	3^{rd} pos.
$C^{(1)}_{1,1}$	1^{st} pos.
$C^{(1)}_{1,2}$	2^{nd} pos.
$C^{(1)}_{1,3}$	3^{rd} pos.

2	
$C^{(0)}_{2,1}$	1^{st} pos.
$C^{(0)}_{2,2}$	2^{nd} pos.
$C^{(0)}_{2,3}$	3^{rd} pos.
$C^{(1)}_{2,1}$	1^{st} pos.
$C^{(1)}_{2,2}$	2^{nd} pos.
$C^{(1)}_{2,3}$	3^{rd} pos.

3	
$C^{(0)}_{3,1}$	1^{st} pos.
$C^{(0)}_{3,2}$	2^{nd} pos.
$C^{(0)}_{3,3}$	3^{rd} pos.
$C_{3,1}$	1^{st} pos.
$C^{(1)}_{3,2}$	2^{nd} pos.
$C^{(1)}_{3,3}$	3^{rd} pos.

4	
$C^{(0)}_{4,1}$	1^{st} pos.
$C^{(0)}_{4,2}$	2^{nd} pos.
$C^{(0)}_{4,3}$	3^{rd} pos.
$C^{(1)}_{4,1}$	1^{st} pos.
$C^{(1)}_{4,2}$	2^{nd} pos.
$C^{(1)}_{4,3}$	3^{rd} pos.

We have given values to vote codes equal to $\{C_{1,1}, C_{1,2}\}$. A systematic way of getting these values so as to be unique is by denoting that the vote codes of the part $a \in \{0,1\}$ under the $tag_l = l$ will belong to the set:

$$2ml + am < C^{(a)}_{l,j} \leq 2ml + am + m$$

So by randomly choosing all the possible values for the m possible choices in a ballot part, from the set $\{2ml + am + 1, 2ml + am + 2, ..., 2ml + am + m\}$ we can

be sure that the vote codes are unique. Each of this vote codes correspond to a rank/place which will be given to a candidate. Moreover, as we have already told the 1^{st} position will be denoted by a commitment to 1, the 2^{nd} position by a commitment to x and the 3^{rd} position by a commitment to x^2, where $x = max\{n+1, 2m-1\} = 5$. Then the EA, randomly permutes each of the sides of the ballots so as to post them to the BB. Therefore, we can suppose that the ballot with $tag = 1$, becomes the following committed ballot, if the permutation for the first side is the identity matrix which leaves it unchanged, while in the second part everything changes order. Thus:

Table 4.2: Sample ballot in committed form

1	
$Com_{ck}(C^{(0)}_{1,1}; t^0_{1,1})$	$Com_{ck}(1; r^0_{1,1})$
$Com_{ck}(C^{(0)}_{1,2}; t^0_{1,2})$	$Com_{ck}(5; r^0_{1,2})$
$Com_{ck}(C^{(0)}_{1,3}; t^0_{1,3})$	$Com_{ck}(25; r^0_{1,3})$
$Com_{ck}(C^{(1)}_{1,3}; t^1_{1,3})$	$Com_{ck}(25; r^1_{1,3})$
$Com_{ck}(C^{(1)}_{1,2}; t^1_{1,2})$	$Com_{ck}(5; r^1_{1,2})$
$Com_{ck}(C^{(1)}_{1,1}; t^1_{1,1})$	$Com_{ck}(1; r^1_{1,1})$

The above logic is applied to all the ballots, with different random permutations, of course. Afterwards, the first step of the ZKP is executed and posted to the BB alongside with the total sum of randomnesses for each side of the ballots.

Then the *Cast* phase begins. Suppose that V_1 gets the ballot with $tag = 1$. He randomly tosses a coin and suppose again that the result is 1. Therefore, he will chose to vote based on the second part of the ballot and he will keep the first part as his receipt. Suppose, now that he wants the P_3 to get the highest rank/position, P_1 is his second favorite and P_2 his third. Then, based on his ballot he casts the following vote:

$$(1, 1, \{C^{(1)}_{1,2}, C^{(1)}_{1,3}, C^{(1)}_{1,1}\})$$

This indicates that the tag field has the value 1, the coin base has the value 1 while his choices mean that he has given the value represented by vote code $C^{(1)}_{1,2}$ to P_1, the value represented by vote code $C^{(1)}_{1,3}$ to P_2, and the value represented by vote code $C^{(1)}_{1,1}$ to P_3. The above process is over when all voters have voted and suppose that the votes from

each are the following:

$$V_1 : (1,1,\{C^{(1)}_{1,2}, C^{(1)}_{1,3}, C^{(1)}_{1,3}\})$$
$$V_2 : (2,0,\{C^{(0)}_{2,1}, C^{(0)}_{2,2}, C^{(0)}_{2,3}\})$$
$$V_3 : (3,0,\{C^{(1)}_{3,3}, C^{(1)}_{3,2}, C^{(1)}_{3,1}\})$$
$$V_4 : (4,0,\{C^{(1)}_{4,1}, C^{(1)}_{4,2}, C^{(1)}_{4,3}\})$$

Afterwards, the **Tally** phase begins where the EA finds each of the commitments that the vote codes corresponds and adds to the first tally set the commitments corresponding to the vote codes $\{C^{(1)}_{1,2}, C^{(0)}_{2,1}, C^{(1)}_{3,3}, C^{(1)}_{4,1}\}$, to the second tally the commitments corresponding to the vote codes $\{C^{(1)}_{1,3}, C^{(0)}_{2,2}, C^{(1)}_{3,2}, C^{(1)}_{4,2}\}$ and to the third tally the commitments corresponding to the vote codes $\{C^{(1)}_{1,3}, C^{(0)}_{2,3}, C^{(1)}_{3,1}, C^{(1)}_{4,3}\}$. After performing homomorphic combination of the values the result of the first tally is a commitment to value 32, from the second tally a a commitment to value 40 and from the third tally a commitment to value 52. Furthermore, the EA posts the commitments to this values alongside with their randomnesses and their openings. It also produces and publishes the third step of the Σ - protocol with the challenge produced from the coins ($\rho = 1011$), alongside with the openings of the not chosen sides from the ballots. Finally, the **Result** phase begins where following the 1 we extract the following results:
P_1 gets ranked with score 5
P_2 gets ranked with score 3
P_3 gets ranked with score 4
Then, anyone can perform the verification of the system from the publicly information of the BB.

4.7 STV

In this section we present a theoretical approach to how an STV voting could be actually implemented in the context of DEMOS-1. The solution is purely theoretical and is based on systems that currently are not feasible for a practical implementation, due to the FHE that is required, and is still under research.

The biggest problem of STV and similar systems arises from its multi-phase nature. Despite, the initial counting which can be implemented with our system with the new proposed protocol, the problem becomes apparent when the second phase begins.

More precisely, supposing that no winner was selected from the first phase then the votes from the candidate who came last (suppose P_j), are transferred, to the second choice of the voters who have voted the candidate P_j as their first choice. Another scenario is the case when we need more than one winners and then after the first winner is selected, the additional votes are distributed among the second choice of the voters, who have voted the winner candidate as their first choice.

Generally, the STV voting has many sub-cases and anyone can read about those [55], but as we explained **there should be a connection among the choices of a voter. On the other hand, this connection among the choices of a specific voter seems infeasible in an homomorphic system, if we require privacy.** However, we provide a way of maintaining the privacy of each voter, while in the meantime our system remains homomorphic. The description of the system is the following.

Suppose, that there are m numbers $\{N_1, ..., N_m\}$, each of which is used to encode the place that a candidate will be placed. Then we state that the first place will be given by powers of N_1, the second place by powers of N_2, etc. An important note here is that we do not actually need to have all these powers. We could actually choose m^2, discrete different numbers (under some assumptions shown in our ZKP approach), as we get rid of them with mod and div operations, but we choose the notation of N_i for simplicity. However, now whenever a voter votes for a candidate let's say P_k he will actually commit to the value (for the first choice):

$$\left[\prod_{i\neq k}(N_1^i+1)\right]N_1^k$$

The ballot now has m^2 choices, m possible for each of the m positions. Therefore, for the first position the m choices will be commitments to $\left\{\left[\prod_{i\neq k}(N_1^i+1)\right]N_1^k\right\}_{k\in\{1,...,m\}}$, for the second position will be $\left\{\left[\prod_{i\neq k}(N_2^i+1)\right]N_2^k\right\}_{k\in\{1,...,m\}}$, etc.

That way we can easily find out the final decrypted result by the homomorphic adding the ciphertexts, which in our FHE system will transfer the addition to the plaintexts (as long as we choose values for N_i such that correctness is satisfied). Thus, in the tallying phase we can find out how many 'first' votes each candidate P_j got by performing:

Algorithm 2 STV Algorithm for counting the i-th position results

for $j = 1$; $j < m$; j++ **do**
  $X \leftarrow T_i$
  for $k = 1$; $k < m$; k++ **do**
    if $k \neq j$ **then**
      $X \leftarrow X \mod N_i^k$
    else
      $X \leftarrow X \div N_i^k$
    end if
  end for
  $s_j^{(i)} \leftarrow X$
end for
return $\{s_1^{(i)}, s_2^{(i)}, ..., s_m^{(i)}\}$

In case we need to transfer votes now, we will need **homomorphic multiplication**. The vote transfer will be done as the following example shows.
Suppose that **candidate P_k has won the election** with l extra votes than the threshold required. Then we want to transfer these l votes evenly, to the second choice of those who voted P_k, as their first choice. We will perform the following for each voter. Suppose $c_1^{(i)}, c_2^{(i)}$ are the first and the second choice of voter V_i, and $\cdot$ is the homomorphic multiplication):

$$res_i = c_1^{(i)} \cdot c_2^{(i)}$$

Then the final result will be produced by simply homomorphically adding the partial res_i, producing the $Res = \sum_{i=1}^{m} res_i$, and performing a similar mod and div to the normal result. Only now we will have that **the candidate P_j, was selected as second to winner candidate P_k with a number equal to $t_j^{(k)}$**. An algorithm for that is shown bellow.

Algorithm 3 Algorithm for counting how many times each candidate was voted as second choice after candidate P_k was selected first.

for $j = 1$; $j < m$; j++ **do**
  $X \leftarrow Res$
  for $i = 1$; $i < m$; i++ **do**
    if $i \neq k$ **then**
      $X \leftarrow X \mod N_1^i$
    else
      $X \leftarrow X \div N_i^k$
    end if
    if $j \neq i$ **then**
      $X \leftarrow X \mod N_2^i$
    else
      $X \leftarrow X \div N_2^j$
    end if
  end for
  $t_j^{(k)} \leftarrow X$
end for
return $\{t_1^{(k)}, t_2^{(k)}, ..., t_m^{(k)}\}$

The above algorithm shows that whenever P_k is not the first choice then the value $t_j^{(k)}$ becomes 0 or else it is just a second position indicator, which again when P_j is not selected as second choice, becomes 0, otherwise we get the exact number of second choices of P_j to P_k.

In conclusion, the ZKP remains similar, only now the EA should prove that the permutation of the correct commitments for each position separately, so we have m shuffle proofs. On the other hand, the main problem is the FHE required, as the order of it should be at least m, something that classifies it as not practical with current FHE implementations, while even the ZKP in the context of FHE is still an unexplored territory. Despite the above, we provide the concept for a future adaptation of STV voting in DEMOS or similar e-voting systems, if and when all this technology regarding FHE becomes available.

Chapter 5

Practical Implementation

5.1 Overview

In this chapter we are going to present our practical implementation of what we described above. More precisely, we will initially refer to the current implementation of DEMOS, and then describe our contribution by explaining we did Additionally, we provide evidence of how our system is performing with the changes. Finally, we will state some bugs of the current implementation and propose some future work that needs to be done so as the new system to be fully functional.

5.2 Current DEMOS implementation

The current DEMOS practical implementation is much different than the theoretical protocol. The initial version of DEMOS was based on the system we are studying here, however, for the final implementation they have chosen a different approach [56]. In this approach, they choose to encrypt values of 0 and 1 and provide simple ZKP similar to the OR-proofs. Moreover, another ZKP is used so as to ensure that the k out of m election is correctly encoded (by proving the permutation-invariant of the ballots). Finally, the system is based on trustees for decrypting the results and producing the ZKP, which means that the ZKP state from the first step of the ZKP is distributed to them, alongside with the randomness of encryption segmented so as to have a threshold encryption similar system [57]. All of the above changes can be found at the current implementation of DEMOS [58].

For the implementation, we keep the current framework that DEMOS has already been created. The current DEMOS implementation is in the Django framework, with Twitter Bootsrap CSS and JS framework. Moreover, the backend of the system is mainly written in Python3 , and we refer to this, because this is where we implement most of our changes. Finally, the petlib library is used so as to offer additional computational capabilities regarding the elliptic curve(p415), where ElGamal is based on.

5.3 Our Implementation

Despite the current practical implementation of DEMOS, we implement our system according to the description we gave to chapter four, which is based on the initial DEMOS theoretical protocol (of course, the practical implementation is a transformation of the current DEMOS implemented changing whatever necessary). Moreover, we choose to implement the version we described with the revised Σ - protocol, due to its proof of correctness and verifiability. It also provides more choices for elections, which (unfortunately) come with an efficiency cost. At a later stage, we will also implement the shuffle proof approach which is more efficient, but there are still some issues to be investigated.

Regarding our practical part, we use exactly the same libraries as the current DEMOS. At this dissertation, we mainly transform and provide the backend of DEMOS, and more precisely the parts that are responsible for the setup of the election. The code we transform for the setup is the one in: 'demos_voting/election_authority/utils/crypto.py' and the corresponding transformed is the one we provide under the name 'EA_setup.py'. In order to review our code we produce some .json files with the necessary data for an election. These can be found under the names: 'ballots_for_users.json', which contains the ballots as they will be received from the voters who will vote, the 'bb_commitment.json', which contains the data that will be posted to the BB, and the 'EA_ballots.json' which contains the data necessary for the decryption of the results and the the third step of the ZKP. All these files will be created into a folder '/setup'. We choose these files to be in .json format so as to handle them better when we finally implement our complete E2E system.

In the meantime, we also provide a code which simulates the (n voters) voting process. This could be done manually, but we provide this code generally as a test of

correctness. This code is the 'voters_vote.py' we provide, which gets as input the '/setup/ballots_for_users.json' and produces the result into a folder '/cast' under the name 'votes.json'. Finally, we also provide a code with filename 'result.py' which is responsible for getting the votes from '/cast/votes.json' and the data from '/setup/EA_setup.py', and simulates the tallying process and the production of the third step of the Σ - protocol. These results are then written to the file 'tallies.txt' located in the folder '/tally'. Finally, we provide a bash script where all the codes can be run sequentially to check the results more easily. In order to run the code (and generally the system) there is a 'requirements' file from the current DEMOS, which consists of the necessary packages that someone needs to download so as to run the system. We propose that these files, are downloaded in a virtual environment (container), while we also provide a configuration file ('config'), with some additional instructions for anyone interested to run an instance of the current DEMOS as web page to their own machine.

Above we gave a brief overview, of what our practical work consisted of. However, for a deeper understanding someone might have to investigate carefully the codes provided. We have properly documented with comments our code, but again the code is not final and is subject to many changes. Finally, for our first implementation we leave the number of k repetitions of the Σ - protocol, to be equal to 1 which apparently is not correct, but we do not try to change it for now, as there is no difference in the results(the same protocol just runs multiple times), while the final implementation will contain multiple trustees and the code will differ a lot from the given one. In the next sub-section we will analyze how our system performs in our local machine.

5.3.1 Performance

The results presented in the following table are obtained when the 'EA_setup.py' is run with different values of n voters and m candidates (n, m are passed as arguments to our file for better management). Moreover, our local machine specs are:

Table 5.1: Laptop specs

Model:	Dell Inspiron 3576
CPU:	Intel Core i7 8550U 1.8GHz
RAM:	8GB
Hard drive:	256GB SSD
GPU:	AMD Radeon 520
OS:	Ubuntu 16.04

while the results we receive are:

Table 5.2: Time Benchmarks

n	**m**	**Time**
100	4	2.6 sec.
1000	4	26 sec.
10000	4	4 min. and 32 sec.
10000	8	13 min. and 20 sec.
10000	16	48 min. and 2 sec.
100000	4	50 min. and 40 sec.

From the above table we can see that even with a conventional laptop we can efficiently simulate big election processes. Moreover, as expected there is an almost linear increase of time with n (despite the problems of the system that will be reviewed later), while different values of m also produce the expected increase in time complexity of $m \log m$ (again leaving aside the bugs that may cause some small delay).

5.4 Problems and Bugs of the System

In this section we analyze some problems and bugs of our system, while we also state what needs to be implemented from now on so as to have a fully functional E2E system. Initially, the biggest problem of our implementation is the use of the petlib library. Petlib uses some recursive calls when trying to find an element of a group e.g. find $y = h^r g^m$. The above will crash occasionally, for random big values of r, m. We manage to overcome this problem by a 'hacky' solution, where instead of directly

performing g^m for a big m we do a successive squaring technique to avoid it. However, this technique adds a delay to the process. In addition to that, the $Bn()$ method from 'petlib.bn' also crashes when an integer larger than 2^{32} is given, which apparently causes a lot of issues as we deal with much bigger numbers in cryptography. The above problems can be avoided if another library package is used as it is done in Helios practical implementation. Therefore, this design decision will be discussed further with the creators of the current system, when we implement the new protocol.

Furthermore, we provide only a part of the backend implementation. This implementation should be part of the whole DEMOS, which means that many models of the current system might need changes. Nevertheless, we necessarily need to provide the frontend of the system, by transforming the Javascript, where we should also provide the verification of the system. Therefore, there are a lot to be done for our system to be complete. Despite that, our practical implementation provides an initial guideline of how the final system will be built, while we also show that the time complexity of the new system is feasible.

Chapter 6

Future Work And Conclusions

In this chapter we give a brief evaluation of the outcomes of this dissertation. Afterwards, we are going to talk about some unresolved issues and how this could be handled in the future, while finally we give a brief overview for the conclusion of our project.

6.1 Evaluation of our Results

Before starting our dissertation there were some expectations regarding the outcomes of our work. We achieved most of them, and many more during our research, while there were only some minor parts that we did not really manage to research in depth. More precisely, the following bullet points describe exactly what we achieved:

- *Presentation and description of the mathematical and cryptographic aspects of modern state of the art cryptosystems, and more precisely demos*: ✓
  These subjects are fully analyzed in chapters two and three of our work.
- *A new enhanced DEMOS protocol, which will provide more options than just approval voting:* ✓
  The new techniques that transform the current DEMOS protocol are fully investigated in the fourth chapter of this dissertation. Moreover, we provide a detailed way of how the new elections will be conducted, by providing sufficient examples and proofs, where required.
- *A practical implementation of what we provided at a theoretical level:* ✓

We provide the practical implementation of the generalized preferential system in the context of DEMOS. Of course, this is a simple back-end implementation written in python, where many changes could be performed in future works.

- *A new ZKP of a shuffle:* (✓!)
  This was not a purpose of this dissertation, but based on the Σ - protocol provided by DEMOS we provide a new efficient ZKP of a shuffle, which can be useful in projects which spread beyond this dissertation.

The only thing that we did not really provide is *a full web-based implementation of our changes in the current web-page of DEMOS.* However, such a project could be a dissertation by its own, since only the development of the theoretical concept is a vast project.

6.2 Unresolved Issues and Future Work

As we stated in the above section the only thing missing is a full web-based implementation. This could be actually classified as future work. Nonetheless, our practical implementation provides a modular code that will be used as a start, where the final practical implementation will be based on. Despite that, we explain in detail at the fifth chapter the exact reasons why we did not move on the final implementation immediately.

Furthermore, while we provide a theoretical protocol for STV voting, the feasibility of such a practical implementation is not currently possible, since the current protocols regarding FHE are still at a very primary level. Other than that, we provide a practical implementation only for generalized preferential voting, but an implementation for score voting is also possible if we consider that a score voting is implemented with the choice of m out of m^2 vote codes (m for each voter). Therefore, this could be a future work for when we transform the current DEMOS system.

Moreover, we provided two possible enhancements of DEMOS based on the ZKP of a shuffle chosen. We look into both but with more attention to the ZKP based on the DEMOS Σ - protocol. Both approaches have advantages and disadvantages as discussed in the previous chapters, and both approaches could be investigated further for additional improvements and maybe more formal security proofs, especially for the case of typical ZKP of a shuffle.

Finally, a more precise analysis regarding the possible attacks and the (practical) resilience of the new system might be provided in future works. Nevertheless, such an analysis is far beyond the scope of this dissertation as we have previously explained.

6.3 Conclusion

In this dissertation we managed to understand in depth complex theoretical protocols and provide our novel solutions to challenging problems. Moreover, we provided detailed and provable results which expand beyond our initial expectations and the scope of this dissertation, while any unresolved issues are considered as potential future work. Generally, we contributed a lot in the foundation of E2E verifiable e-voting and the evolution of DEMOS to a usable system for a wide range of applications.

Bibliography

[1] Paul Barry Clarke and Joe Foweraker. *Encyclopedia of democratic thought*. Routledge, 2003.

[2] Josh Benaloh, Ronald Rivest, Peter YA Ryan, Philip Stark, Vanessa Teague, and Poorvi Vora. End-to-end verifiability. *arXiv preprint arXiv:1504.03778*, 2015.

[3] Matthew Soberg Shugart and Justin Reeves. *Electoral System Reform in Advanced Democracies*. Oxford University Press, 2012.

[4] Peter C Fishburn. Arrow's impossibility theorem: Concise proof and infinite voters. *Journal of Economic Theory*, 2(1):103–106, 1970.

[5] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *International Workshop on the Theory and Application of Cryptographic Techniques*, pages 244–251. Springer, 1992.

[6] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE security & privacy*, 2(1):38–47, 2004.

[7] John B Fraleigh. *A first course in abstract algebra*. Pearson Education India, 2003.

[8] David Steven Dummit and Richard M Foote. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.

[9] Daniel Wichs and Giorgos Zirdelis. Lecture 2: Macs, statistical distance, statistical security. `http://www.ccs.neu.edu/home/wichs/class/crypto-fall15/lecture2.pdf`, 2015. [Online; Accessed 2018-08-14].

[10] Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge university press, 2009.

[11] Hans Delfs and Helmut Knebl. Symmetric-key cryptography. In *Introduction to Cryptography*, pages 11–48. Springer, 2015.

[12] William Stallings. *Cryptography and network security: principles and practice*. Pearson Upper Saddle River, NJ, 2017.

[13] Data Encryption Standard et al. Federal information processing standards publication 46. *National Bureau of Standards, US Department of Commerce*, 23, 1977.

[14] NIST-FIPS Standard. Announcing the advanced encryption standard (aes). *Federal Information Processing Standards Publication*, 197:1–51, 2001.

[15] Eli Biham and Adi Shamir. *Differential cryptanalysis of the data encryption standard*. Springer Science & Business Media, 2012.

[16] Yehuda Lindell and Jonathan Katz. *Introduction to modern cryptography*. Chapman and Hall/CRC, 2014.

[17] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

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

[19] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[20] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[21] Richard E Ladner. On the structure of polynomial time reducibility. *Journal of the ACM (JACM)*, 22(1):155–171, 1975.

[22] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[23] Dan Boneh. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium*, pages 48–63. Springer, 1998.

[24] Mihir Bellare and Phillip Rogaway. Introduction to modern cryptography. *Ucsd Cse*, 207:102–108, 2005.

[25] Mihir Bellare and Phillip Rogaway. Introduction to modern cryptography. *Ucsd Cse*, 207:127–131, 2005.

[26] Yevgeniy Dodis. Lecture 14: Introduction to cryptography. `https://cs.nyu.edu/courses/spring12/CSCI-GA.3210-001/lect/lecture14.pdf`, 2012. [Online; Accessed: 2018-08-14].

[27] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[28] Gerardo I Simari. A primer on zero knowledge protocols. *Universidad Nacional del Sur*, 6(27):1–12, 2002.

[29] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[30] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Annual International Cryptology Conference*, pages 89–105. Springer, 1992.

[31] 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*, pages 116–125. ACM, 2001.

[32] Jun Furukawa, Hiroshi Miyauchi, Kengo Mori, Satoshi Obana, and Kazue Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. In *International Conference on Financial Cryptography*, pages 16–30. Springer, 2002.

[33] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology*, 23(4):546–579, 2010.

[34] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in CryptologyCRYPTO86*, pages 186–194. Springer, 1986.

[35] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In *Interna-*

tional Conference on the Theory and Application of Cryptology and Information Security, pages 626–643. Springer, 2012.

[36] Nir Bitansky, Dana Dachman-Soled, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, Adriana López-Alt, and Daniel Wichs. Why fiat-shamir for proofs lacks a proof. In *Theory of Cryptography*, pages 182–201. Springer, 2013.

[37] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 393–403. Springer, 1995.

[38] Paul Syverson, R Dingledine, and N Mathewson. Tor: The secondgeneration onion router. In *Usenix Security*, 2004.

[39] Georgios Tsoukalas, Kostas Papadimitriou, Panos Louridas, and Panayiotis Tsanakas. From helios to zeus. In *EVT/WOTE*, 2013.

[40] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 539–556. Springer, 2000.

[41] Craig Gentry and Dan Boneh. *A fully homomorphic encryption scheme*, volume 20. Stanford University Stanford, 2009.

[42] Ben Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.

[43] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-end verifiable elections in the standard model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 468–498. Springer, 2015.

[44] Saghar Estehghari and Yvo Desmedt. Exploiting the client vulnerabilities in internet e-voting systems: Hacking helios 2.0 as an example. *EVT/WOTE*, 10:1–9, 2010.

[45] Nikos Chondros, Bingsheng Zhang, Thomas Zacharias, Panos Diamantopoulos, Stathis Maneas, Christos Patsonakis, Alex Delis, Aggelos Kiayias, and Mema Roussopoulos. D-demos: A distributed, end-to-end verifiable, internet voting system. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pages 711–720. IEEE, 2016.

[46] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. Demos-2: scalable e2e verifiable elections without random oracles. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 352–363. ACM, 2015.

[47] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[48] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985.

[49] David Lichtenstein, Nathan Linial, and Michael Saks. Imperfect random sources and discrete controlled processes. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 169–177. ACM, 1987.

[50] Jesse Kamp and David Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. *SIAM Journal on Computing*, 36(5):1231–1247, 2006.

[51] Boaz Barak, Guy Kindler, Ronen Shaltiel, Benny Sudakov, and Avi Wigderson. Simulating independence: New constructions of condensers, ramsey graphs, dispersers, and extractors. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 1–10. ACM, 2005.

[52] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124. ACM, 2011.

[53] C Andrew Neff. Verifiable mixing (shuffling) of elgamal pairs. *VHTi Technical Document, http://www. votehere. net/vhti/documentation/egshuf. pdf, VoteHere, Inc*, 2003.

[54] Jun Furukawa. Efficient and verifiable shuffling and shuffle-decryption. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 88(1):172–188, 2005.

[55] Electoral Reform Society. Single transferable vote. `https://www.`

electoral-reform.org.uk/voting-systems/types-of-voting-system/single-transferable-vote/. [Online; Accessed 2018-08-14].

[56] Overview of demos e-voting practical. http://www-en.demos-voting.com/how-it-works. [Online; Accessed 2018-08-14].

[57] Yvo Desmedt. Threshold cryptosystems. In *International Workshop on the Theory and Application of Cryptographic Techniques*, pages 1–14. Springer, 1992.

[58] Marios Levogiannis. Demos e-voting implementation. https://github.com/mlevogiannis/demos-voting. [Online; Accessed 2018-08-14].