Two New End to End Verifiable Voting Schemes Based on Mixnet Based Helios Architecture

Student Name: Muhammed Noufal K

IIIT-D-MTech-CS-IS-12-007
July 17, 2015

Indraprastha Institute of Information Technology
New Delhi

Thesis Committee
Dr. Donghoon Chang, IIIT Delhi (Chair)
Dr. Ratna Dutta, IIT Kharagpur
Dr. Somitra Sanadhya, IIIT Delhi

Submitted in partial fulfilment of the requirements
for the Degree of M.Tech. in Computer Science,
with specialization in Information Security

Keywords: End to end verifiable voting, Mixnet, Helios, Proof of integrity, Tweak, Dummy vote

Certificate

This is to certify that the thesis titled **"Two New End to End Verifiable Voting Schemes Based on Mixnet Based Helios Architecture"** submitted by **Muhammed Noufal K** for the partial fulfilment of the requirements for the degree of *Master of Technology* in *Computer Science & Engineering* is a record of the bonafide work carried out by him under our guidance and supervision in the Security and Privacy group at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Dr. Donghoon Chang

Indraprastha Institute of Information Technology, New Delhi

Abstract

Fair conduct of elections are essential for smooth existence of democratic societies. Voting systems are the method and infrastructure we use to conduct elections. Paper ballot based voting schemes, electronic voting machines based schemes, etc., are the some of the traditional voting schemes. These schemes full fills most of the basic requirements, and simplicity of these schemes makes them attractive even today. But there have a major drawback for these schemes, that is, for a voter it is nearly impossible to verify the correctness of election results. Voters left with no choice other than trusting on election administrators and party representatives for the correctness of election result. To address this issue, researchers come up with end to end verifiable voting schemes, where voter can verify that her vote is cast-as-intended, recorded-as-cast and counted-as-recorded, and anybody can verify correctness of each and every step, all the while preserving voter privacy. From the last three decades, researchers have proposed many such schemes, many of them make use of mixnet for anonymization. Pîret á voter, Helios are the example for such mixnet based voting schemes. Here the mixnet work as a black box, and used for removing the mapping between voters and the encrypted votes. In the voting schemes case, mixnet receives a set of encrypted votes as input and output another set of ciphertexts. This makes proof of integrity is essential, to ensure that mixnet haven't added or removed any vote ciphertext. In this dissertation, we propose two new schemes, both of them follows same structure of Helios but facilitate new ways for proof of mixnet integrity. The idea of first one is based on adding tweak values in ElGamal ciphertext. And the second one make use of dummy votes to provide proof of mixnet integrity. While the tweak based scheme is only applicable to ElGamal encryption scheme based mixnets, the dummy vote based idea can be extended for proof of integrity in any other mixnet based schemes.

Acknowledgments

First and foremost, I would like to express my sincere gratitude and thanks to my academic advisor Dr. Donghoon Chang, for his thought provoking and motivating words, understanding and patience during my research work. His guidance and support helped me to explore deep in to the area of verifiable voting schemes and to complete this thesis work.
I would also like to thank Dr. Somitra Sanadhya for supporting me during my master studies and teaching me the wonderful course CSE-546 Applied Cryptography, that motivated me to study more and do research in the area of cryptology.
I would like to thank Jinkeon Kang for helping me with useful suggestions and for reviewing the first draft of thesis writing.
I am very grateful to all teachers and mentors who taught and guided me throughout my academic journey.
I would also like to thank all my friends here in IIIT Delhi who made my stay here enjoyable by sports and other fun activities.

Finally, my deepest gratitude goes to my parents, for their unconditional love, support and encouragement throughout my journey. Without their support and guidance, I would have never been who I am today.

Contents

List of Figures

Chapter 1

Introduction

The election problem, the process of choosing a person from multiple candidates based on voters consensus is a common problem in democratic societies. Voting systems are the method and infrastructure we use to conduct the election. The trustworthiness of the voting scheme is crucial to record people consensus correctly. Paper ballot voting schemes, electronic device based voting schemes, etc., are the some of the voting schemes we use currently. The integrity of currently practicing voting schemes relies on procedures. For example, in case of electronic voting machine based election in India, procedures are, conduct trial voting in front of representatives from competing parties, ensure machine holds zero votes before the beginning of actual voting, seal the machine just after voting phase etc. In a large scale implementation, it is very hard for an election authority to enforce these procedures. For a voter, it is impractical to verify that authorities followed all the correctness measures and procedures, and there is no way to ensure that her vote is properly included in final tally. This makes voters trust on election authority and reliability of machineries as crucial component in successful conduct of an election. Once election result announce, result can be surprising to many of the voters and public, this can lead to doubt among voters about integrity of election, and have the potential to break the smooth running of democratic systems. To avoid this kind of situations, researchers come up with a new kind of election system where, correctness of each and every step is verifiable, all the while preserving voters privacy. This kind of schemes are known as end to end verifiable voting schemes. Apart from the benefits offered by currently using election schemes, end to end verifiable voting scheme allow voters to ensure that her vote is cast-as-intended, recorded-as-cast and counted-as-recorded. And anyone from public can verify the correctness of each and every steps of election.

1.1 History

The idea of end to end verifiable voting scheme is first introduced in 1980s. The first such scheme is proposed by Josh D. Cohen and Michael J. Fischer in 1985 [12], where election integrity is

verifiable but government is able to read any vote. Later on, researchers come up with several such schemes ([3], [19], [13], [18], [11], [21]). Pîret à Voter [7], PunchScan [2], Scantegrity [6], ThreeBallot [22] and Helios [1] are the recent major proposals. Pîret à voter is proposed by Peter YA Ryan in 2004 and it is a paper based scheme uses mixnet using onion encryption for voter privacy. PunchScan is introduced by David Chaum in 2005, later merged with Scantegrity. Scantegrity is proposed by David Chaum in 2008 and later gone through several major modification. The current version is known as Scantegrity II. It can be implemented using existing optical scan voting system infrastructure. ThreeBallot scheme is proposed by Rivest in 2006, it is purely paper ballot based, without using any cryptographic tools. But later, ThreeBallot scheme found out to be vulnerable to several attacks. Helios is a web based open audit voting scheme implemented by Ben Adida, the main idea of Helios is based on verifiable election schemes proposed by Benaloh [4]. In this dissertation, we explain the mixnet based implementation of Helios in detail and propose two new schemes, both are similar to Helios but facilitates better proof of mixnet integrity.

1.2 Motivation

The mixnets are used for anonymization in many of the end to end verifiable voting schemes, including Helios. A mixnet is the sequential arrangement of mix servers. Each mix server read a set of ciphertext as input, re-encrypt, permute, and output the resulting set ciphertexts. To hide the mapping between input and output set, mix server keeps all internally chosen random parameters as secret. The challenge here is, how we can prove that mixnet is correctly transformed input set to output set, without revealing the input-output mapping. Helios provides proof of mixnet integrity by using Sako-Killian shuffle and proof [14]. This proof need each mix server have to generate many shadow mix outputs, and verification task is not easy for an ordinary person. To make public verification of voting schemes accessible for ordinary people, we need easy to explain and straight forward proof of mixnet integrity.

1.3 Contribution

In this dissertation, we propose two new verifiable voting schemes. The first one is based on tweak in ElGamal ciphertext and the second scheme is based on dummy votes. Both the schemes follows same architecture of mixnet based Helios scheme. The major difference from Helios is on proof of mixnet integrity. The dummy vote based idea is general and can be extended to other mixnet applications also. The tweak based idea is specific to ElGamal encryption based mixnets only.

1.4 Outline

In the first part of this work, in the preliminaries chapter, we explain the cryptographic tools and major terms that used for explaining further parts of this dissertation. Our proposal is mainly based on mixnet based Helios scheme architecture, so we explain mixnet in detail. Internally, mixnet make use of ElGamal re-encryption scheme, so we explain ElGamal encryption and re-encryption in details. In the next chapter, we explain the security requirements of end to end verifiable voting schemes.

After these two preliminaries chapter, we explain existing scheme mixnet based Helios in detail. Helios is one of the major implementation of verifiable voting scheme and it is developed by Ben Adida. We also explain how Helios fulfilling the security requirements of End to End verifiable voting schemes.

In the next chapter, we explain our proposed schemes in detail. There have two schemes we proposed, the first one is based on tweak in ElGamal ciphertext and the second proposal is based on dummy votes. In the coming chapter, we explain how proposed schemes fulfills the security requirements of end to end verifiable schemes.

In the final part, we give a comparison between Helios and proposed schemes, and finally we summarize with conclusion and future work.

Chapter 2

Preliminaries

In this chapter, we explain details of cryptographic tools and terms used for explaining further sections of this dissertation.

2.1 ElGamal Encryption System

The ElGamal encryption system is an asymmetric key encryption algorithm designed by Taher ElGamal in 1985 [9] . It is an extension of Diffie-Hellman key exchange algorithm. Both the algorithms, Diffie-Hellman key exchange and ElGamal encryption, are based on the assumption that computing discrete logs in a large prime modulus is computationally infeasible. The security of ElGamal encryption scheme relies on the decision Diffi-Hellman assumption [23].

ElGamal encryption scheme consists of three components: key generator, encryption algorithm and the decryption algorithm. Here we explain the ElGamal encryption scheme that support semantic security [1]. Let us assume that Alice is the receiver and Bob is the sender.

2.1.1 Key Generation

Like any other asymmetric encryption scheme, here the receiver have to create and publish the public key parameters in advance. The key generation procedure as follows:

1. Alice generate a large (1024 bits) random prime q, compute $p = 2q + 1$, repeat the process until p also a prime number.
2. Alice chooses a generator g of the q-order subgroup of Z_p^*.
3. Alice select a secret key $x \in Z_q$ and compute corresponding public key $y = g^x \, mod \, p$.
4. Alice keep x as secret key and publishes y, along with descriptions $g,\ p,\ q$ as public key parameters.

2.1.2 Encryption

Consider Bob want to send a message m to Alice. The encryption algorithm works as follows:

1. Bob chooses $r \in Z_q$, compute $\alpha = g^r \ mod\ p$.
2. Bob map his secret message m in to m_0, where m_0 is an element of q-order subgroup of Z_p^*. The mapping procedure is as follows:
   (a) If m is already an element of q-order subgroup of Z_P^*, return $m_0 = m$.
   (b) Else, compute $m_0 = m$ and, if $m_0^q \equiv 1\ mod\, p$, return m_0. Else return $m_0 = -m_0 mod\, p$.
3. Bob compute $\beta = m_0\ y^r \ mod\ p$.
4. Bob sends the ciphertext $C = (\alpha, \beta)$.

2.1.3 Decryption

Consider Alice received the ciphertext $C = (\alpha, \beta)$. Alice possess the secret decryption key x. The decryption algorithm works as follow:

1. Alice compute $m_0 = \beta\ (\alpha^x)^{-1} \ mod\ p$.
2. Alice reverse map m_0 in to plaintext message m as follows:
   (a) If $m_0 \leq q$, set $m = m_0$. Else $m = -m_0 \ mod\ p$.
   (b) Assign $m = m - 1$.

The decryption algorithm produces intended message, since $\beta\ (\alpha^x)^{-1} = m_0\ y^r\ (g^{rx})^{-1} = m_0\ g^{xr}\ (g^{rx})^{-1} = m_0$.

2.2 Re-encryption schemes

Re-encryption schemes are cryptosystems that allows any third party to alter the ciphertext without affecting the plaintext. By re-encrypting a ciphertext, third party is not gaining any information about either plaintext or secret key. Re-encryption schemes are useful in the cases where one have to change the ciphertext without damaging plaintext.

ElGamal Re-encryption

Let $m \in Z_q$ be the message and $y,\ g,\ q,\ p$ are public key parameters. Alice compute the cipher text C such that $C = (\alpha, \beta) = (g^r mod\, p,\ m\ y^r mod\, p)$, where $r \in Z_q$ is the random secret chosen by Alice . The re-encryption process as follows:

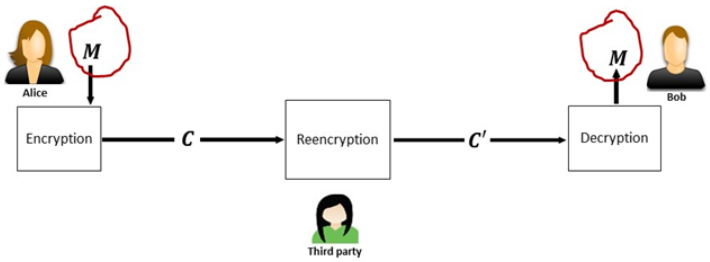


Figure 2.1: An overview of Re-encryption

1. Third party chooses a re-encryption exponent s such that $s \in Z_q$.
2. Compute $C' = (\alpha\ g^s,\ \beta\ y^s)$.

Decryption will work same as usual ElGamal decryption, such that Bob compute $m = \beta\ (\alpha^x)^{-1}\ mod\, p$. It is clear that C and C' decrypt to the same plaintext, C with randomness r and C' with randomness $r + s$.

2.3 Mixnet

The concept of mixnet is first introduced by Chaum in [5]. The word mixnet stands for mix network. Mixnet is a cryptographic construction consists of two or more mix servers arranged sequentially, each will receive set of encrypted messages, re-encrypt them, and output them in an unrevealed, randomly permuted order [10]. Mixnets are commonly used for providing anonymization. Below image shows an example mixnet, with m mix servers and an input set consist of n ciphertexts. Here C_i^j means, i^{th} output from j^{th} mixnet. A mixnet consist of many

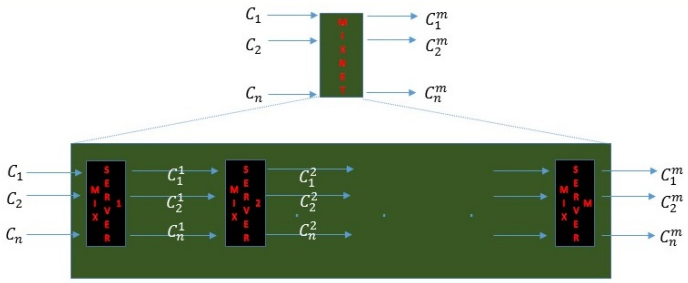


Figure 2.2: An example mixnet with m mix servers and an input set consist of n ciphertexts

mix servers, each mix server will work as independent and managed by different stakeholders. Each Mix server conduct mixing operation and keep all the internal variables used for mixing as secret. It is possible that all mix servers collude and share the secret variables used while mixing. In this case, anonymity compromised. To avoid this and to ensure the expected user anonymity service from mixnet, at least one mix server must be honest. Here honest mix server means, the mix server which do not reveal the random variables used while mixing. To avoid collusion, usually mix servers are handled by different competing stake holders. In case of voting

schemes, each competing party representatives can place a mix server in mixnet, so the chance of collusion can be avoided.

2.3.1 Internals of a Mix Server

A mix server consists of two major parts, a re-encryption part and a permutation part. The

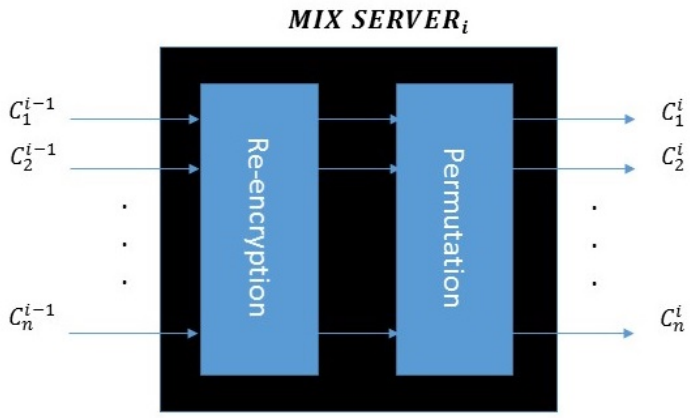


Figure 2.3: Internals of a mix server

re-encryption part will alter each ciphertext received, without affecting the plaintext inside. The permutation part will permute the input set by choosing a random permutation π_i from the all possible permutations Π. Following section explain each part in details.

In short, here a single mix server receive set of ciphertexts as input, and outputs another set of ciphertexts, both input and output sets represents same set of plaintexts. By keeping the mapping between input and output secret, a mix server work as a building block for a mixnet to provide anonymization functionality.

ElGamal Re-encryption inside a Mix Server

Here we shows an example for re-encryption inside a mix server using ElGamal re-encryption scheme. The ElGaml re-encryption scheme is explained in detail in section 2.2. Inside a mix server, every ciphertexts will be re-encrypted using different re-encryption exponent. Figure 2.4 shows the internal re-encryption process in details, where C_j^i means j^{th} output from i^{th} mix server and s_j^i is the j^{th} re-encryption exponent generated by i^{th} mix server. Mix server will keep each of these re-encryption exponent as internal secret.

Permutation inside a Mix Server

Once re-encryption is done, mix server will permute the set of ciphertexts using a randomly chosen permutation $\pi_i \in \Pi$. Mix server will keep choosen permutation π_i as an internal secret. Figure 2.5 shows the permutation inside a mix server.

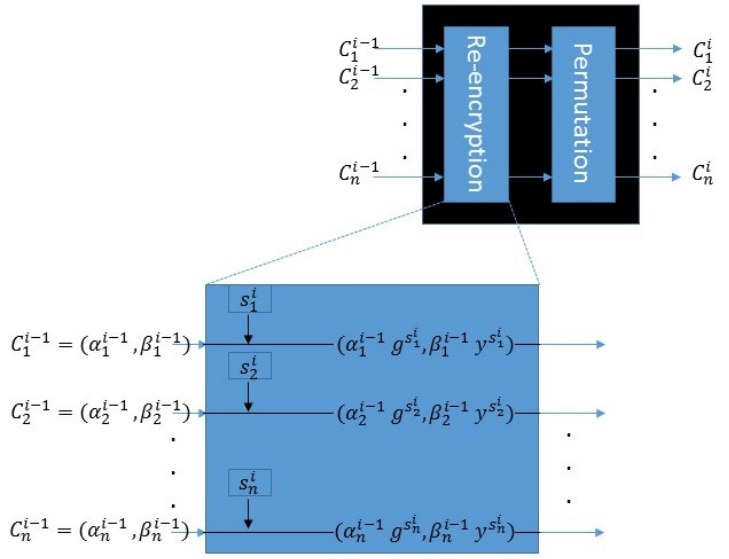


Figure 2.4: ElGamal re-encryption process inside a mix server, where C_j^i means j^{th} output from i^{th} mix server and s_j^i is the j^{th} re-encryption exponent generated by i^{th} mix server.

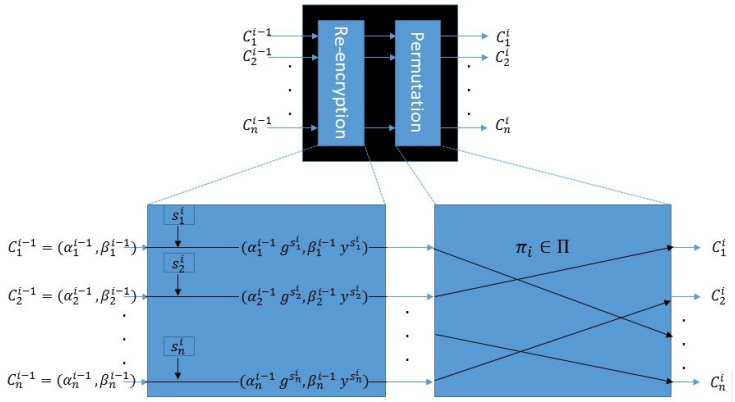


Figure 2.5: Permutation inside a mix server

2.4 Other major terms used and meaning

Here we explain the meaning of some of the major terms used for explaining verifiable voting schemes.

Vote

Vote is a formal indication of choice. A vote consist of a selection from predetermined set of candidates [16].

Ballot

Ballot is the mean by which voter can express her choice. In case of elections where a voter have to do multiple votes, we can say that a ballot is a structure combined of multiple votes [16].

Bulletin board

In this study, we mean, A bulletin board is a public web page that contains all participated voters identity information and corresponding encrypted votes.

Chapter 3

Security requirements of End to End Voting Schemes

The purpose of a voting system is to facilitate election process, where a winner will be chosen from a set of candidates based on the collective voters choice recorded. If choosing winner was the only requirement, the task of designing a voting scheme would have been straight forward and easy. But in practice, for the smooth conduct of an election process, the loser have to accept the election result. This implies that the trustworthiness is crucial requirement a voting scheme must have. Similarly, there have some other requirements a voting scheme should possess such as privacy of the voter, integrity of election process, prevention of vote selling etc. In this chapter, we explain the security requirements of an end to end voting scheme in detail.

3.1 Voter Privacy

The voter privacy means, other than voter, no one else should be able to gain any knowledge about the choice made by voter, even if there exist collusion between multiple parties [17]. This will empower voter to use her free will without worrying about the possible adverse effects due to the leakage of choice she made. In ideal, voter privacy should be preserved even if whole voting system collapses.

3.2 Verifiability

In general, the term verifiability means the process of establishing a valid proof of something. A voting scheme is called as verifiable end to end voting scheme if one can prove the correctness of each step in the voting process. End to end verification prevent any adversarial attempt to malfunction the voting process. This increase the trustworthiness of the voting scheme. The verifiability can be sub divided in to two classes such as individual verifiability and universal verifiability.

3.2.1 Individual Verifiability

Every eligible voter should be able to verify that their vote is recorded correctly and included in final tally [20]. This is also known as ballot casting assurance.

3.2.2 Universal Verifiability

Anyone should be able to verify that all recorded votes are correctly included in final tally [17]. Since the voter privacy is an important requirement, mapping between voter and recorded encrypted votes should be removed before decryption. The integrity of this anonymization process should be able to verify by any third party.

3.3 Receipt freeness

The voter should not be able to neither obtain nor produce a receipt that can prove the content of their vote to a third party [15]. This is to prevent vote selling and coercion. Without receipt freeness, an adversary can ask a voter to prove the choice she made in the ballot by paying money or by threatening voter.

3.4 Eligibility

Eligibility refers to two things, the first one is, only the votes from eligible voters should be counted in final tally. The second one, the maximum number of votes from voter must be restricted, in general, one vote per voter [20].

3.5 Fairness

While voting process is going on, voting system should not leak any information that can influence voters decision. For example, while voting phase is going on, any information about the current lead can influence the voters decision. This must be avoided. In this regards, fairness means, no voters should be able to gain any partial information about the tally before finishing the voting phase [20].

Chapter 4

Existing Scheme: Helios

In last three decades, researchers have done significant amount of work on verifiable election schemes. The first such major scheme was proposed by Cohen and Fisher in 1985, named as "A robust and verifiable cryptographically secure election scheme" [12]. Later on, many such schemes proposed by making use of mixnet, blind signature algorithms, homomorphic encryption etc. ThreeBallot , Pîet á voter, Scantegrity II and Helios are the recent major end to end verifiable voting schemes. In this chapter, we explain mixnet based Helios scheme in detail.

4.1 Helios

Helios is a web based end to end verifiable voting system implemented by Ben Adida [1]. The original idea of Helios scheme is mainly based on verifiable election scheme proposed by Josh Benalaoh in 2006 [4]. The first version of the Helios scheme is proposed in 2008, and later on Helios gone through several changes in underlying cryptographic tools and the software implementation. Here we study about the original first version of Helios scheme. Helios provides remote voting facility using web based implementation. Any remote voting scheme is inherently vulnerable to voter coercion problem, such as an adversary can influence the voter decision either by incentivizing or by threatening. Due to this, use of Helios is not recommended for high stake elections like government elections. Other than this inherent weakness associated with remote voting, Helios satisfies all the security requirements of an end to end verifiable voting scheme. The web based implementation of Helios is ideal for low coercive election environments such as student committee, local club etc. Helios scheme make use of cryptographic tools such as ElGamal encryption, ElGamal re-encryption, Sako-Killian mixnet etc. Helios implementation follows web server-client architecture. Helios server module will be running in a separate server machine managed by election administrators. Helios web client module will run inside web browser of voters. For explaining in modular way, we divide the whole voting process in to three phases: election setup phase, voting phase and post voting phase. Following sections explains each phases in detail.

4.1.1 Phase 1: Election Setup Phase

This is mostly administrative phase. Here the election administrators do the following things with the help of Helios server.

- Generate encryption keys
- Setup the ballot
- Generate and send voter authentication credentials

Generate Encryption Keys

Helios uses ElGamal encryption scheme for encrypting the marked ballot. Helios server will generate ElGamal encryption parameters such as $x,\ y,\ g,\ p,\ q$, where $PK_s = y,\ g,\ p,\ q$ are the public key parameters and $SK_s = x$ is the private key. The ElGamal key generation is explained earlier in section 2.1. Voter privacy is relies on secrecy of private key. In practice, these ElGamal key parameters are generated by using threshold encryption scheme, where representative of each competing party holds a share of private key. For simplicity, here we assume that Helios server generate and keep private key as secret.

Setup the Ballot

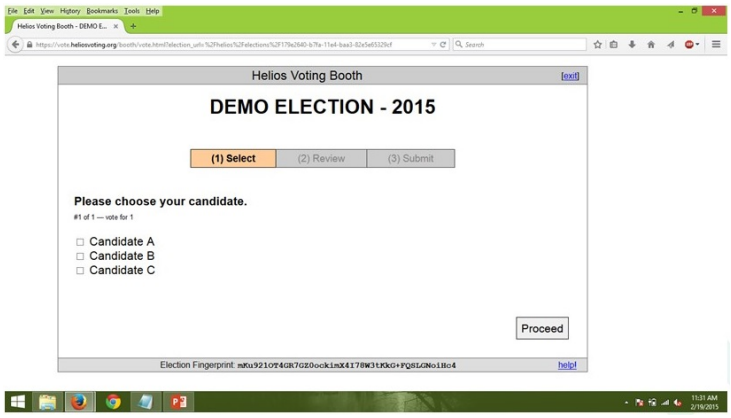


Figure 4.1: Screenshot of an example Helios ballot

This is purely administrative task. Administrators finalize the competing candidates list and create electronic ballot with the help of Helios web interface. Figure 4.1 shows an example Helios ballot.

Generate and Send Voter Authentication Credentials

Only eligible voters should be able to cast the vote. Voter authentication is the mechanism to enforce this requirement. So before the beginning of voting phase, every eligible voter should

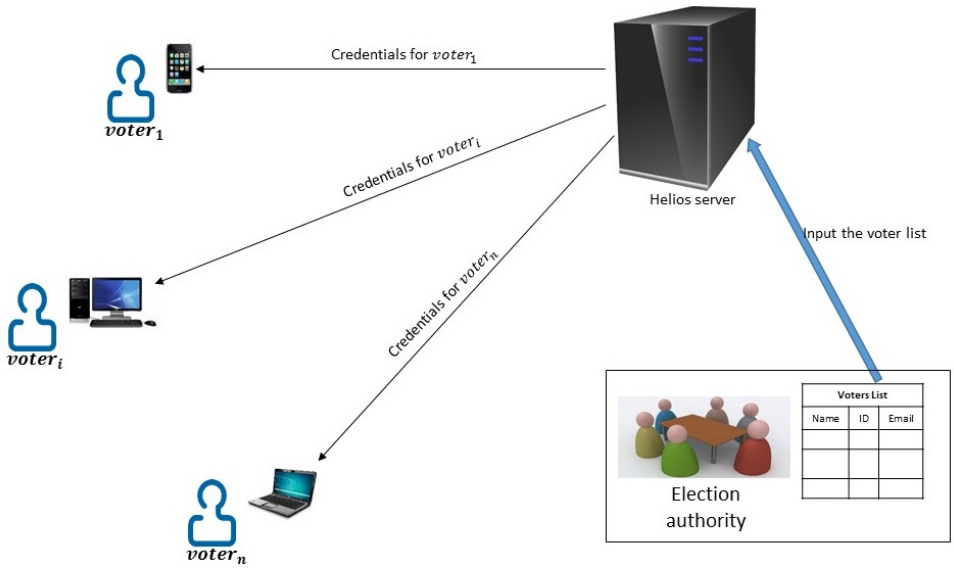


Figure 4.2: Figure shows the Helios server generates and send voter authentication credentials

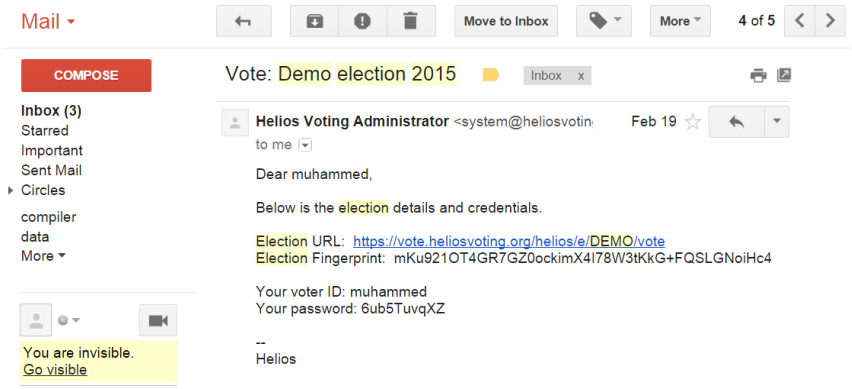


Figure 4.3: Screenshot of Helios credentials received by email

receive the authentication credentials. Helios expect that every voter have personal email account. As shown in figure 4.2, voting authority feed the voters name list with corresponding email id to the Helios server. Helios server generate unique credentials for each voter, and send to the corresponding mail id. Figure 4.3 shows an example for mail received from Helios server. Each voter will receive url to the voting page, user id and password. In post voting phase, list of all voters who participated the election will be displayed in public bulletin board, So that anyone can verify the correctness of voters list.

4.1.2 Phase 2: Voting Phase

This is the phase where actual voting happen. Voter open the Helios voting page in web browser, fill the online ballot. Then the ballot will be encrypted using ElGamal encryption algorithm, using the public key parameters PK_s of Helios server. To achieve receipt freeness, a random number that is hidden from voter will be concatenated with ballot before encrypting. The

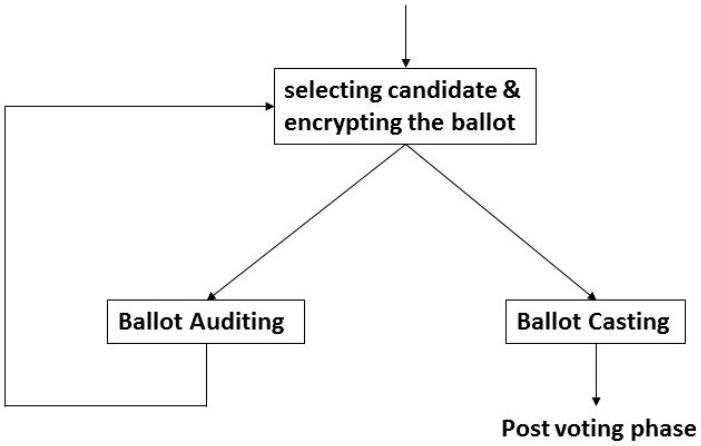


Figure 4.4: Control flow in voting phase

encrypted ballot, voter either can audit to verify the correctness or cast (submit). If voter choose to audit, voter have to repeat the process from candidate selection step onwards again. Voters can repeat the process until voter gain trust on ballot encrypting module. If voter choose to cast, voter have to authenticate herself by using credentials received earlier. Upon successful authentication, encrypted ballot will be added to Helios server database. The voting phase can be divide in to three steps as follows.

1. Selecting candidate & Encrypting the ballot
2. Ballot auditing
3. Ballot casting

Figure 4.4 shows control flow among above three steps. Following sections explain each steps in detail.

Selecting Candidate & Encrypting the Ballot

Here the voter mark her choice in the ballot and Helios web client module create encrypted ballot. For a voter $voter_i$, let V_i be the vote choice and r be the random number generated by Helios web client module. The ballot encryption process as follows.

1. Voter mark her choice in ballot, say V_i
2. Helios web client module generates an one time secret random number r
3. Helios web client module encrypt the vote V_i with r using ElGamal encryption scheme as follows.
   $M_i = V_i \mid\mid r$
   $C_i = ElGamal(PK_s, M_i) \ = \ (g^{r_i}, \ M_i y^{r_i})$, where $r_i \in Z_q$ is random secret chosen by web client.

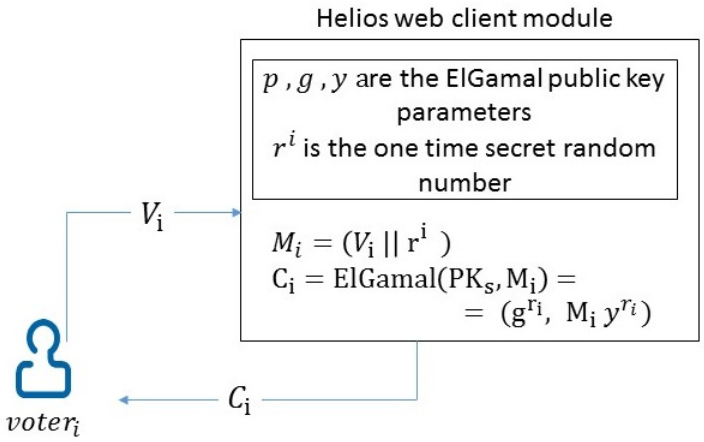


Figure 4.5: Figure shows $voter_i$ submit the vote V_i, Helios web client module concatenate the vote with one time random number r and then encrypt using ElGamal public key PK_s

4. Helios web client shows ciphertext C_i to the voter

Now the voter can see her encrypted vote C_i and voter already knows her vote V_i and ElGamal public key parameters, but due to the secret random number r concatenated by Helios web client, voter cannot demonstrate ballot encryption to anyone else, so that voter cannot prove to a third party that to which candidate she voted. This is important to achieve receipt freeness.

Ballot Auditing

Once voter receives her encrypted ballot, voter can either audit or cast the ballot. Voter choose audit when she want to assure that Helios web client module didn't cheat her by changing the vote while encryption. Due to the concatenation of secret one time random number by Helios web client, voter cannot verify the correctness of encryption process herself without the help of Helios web client. The ballot auditing process as follows.

1. Voter choose ballot audit option
2. Helios web client module reveals the random number added while encryption.
3. Now voter knows ciphertext C_i and all the parameter used while encryption, vote V_i, random number r and ElGamal public key parameters. Voter can verify the correctness of encryption process by repeating encryption in her trusted machine.

Note that once voter chosen audit option, voter have to start from candidate selection step again, and then Helios web client encrypt the vote by concatenating a newly generated random number. Voter can repeat this process until gain trust on Helios web client module.

Ballot Casting

In this step, Helios web client ask for authentication information from voter. Upon successful completion of authentication, the Helios web client module send the encrypted ballot to the Helios server and server store this in database. Figure 4.6 shows the overview of ballot casting process.

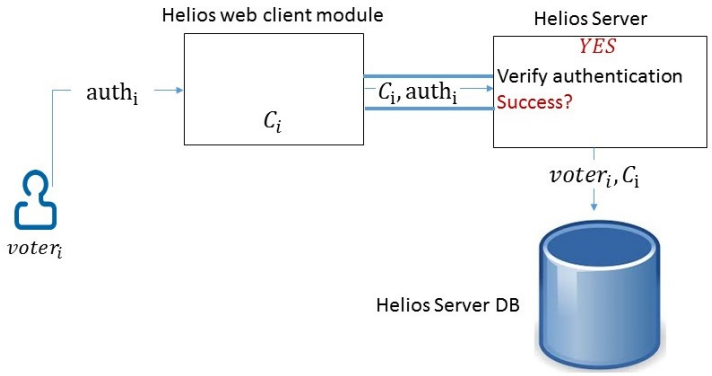


Figure 4.6: Figure shows ballot casting in Helios. Voter submit authentication credentials to Helios web client, Helios web client submit encrypted ballot and credentials to the Helios server. After successful authentication, Helios server add the encrypted vote to database

4.1.3 Phase 3: Post Voting Phase

Once voting deadline is over, the task remaining is tally and announce the result. The task is challenging because, system have to ensure voter privacy and election integrity. If voter privacy was not a concern, Helios could have decrypt and count all encrypted ballot straight forward. Since voter privacy is important, before decrypting, Helios have to remove the mapping between voter identity and encrypted ballot. At the same time, Helios have to ensure that no new votes are added or removed in the system. Helios do following steps once voting phase is over.

1. Publish the bulletin board
2. Mixnet operation
3. Decryption and Tally

Following sections explains each steps in details.

Publish the Bulletin Board

In Helios, bulletin board means a public web page that shows the copies of all the voters name and corresponding cast votes (encrypted). A voter can see and verify that her vote is present in the bulletin board. Similarly, anybody from public can verify the correctness of voters list, which means, all cast votes are from eligible voters. Figure 4.7 shows the model of bulletin board.

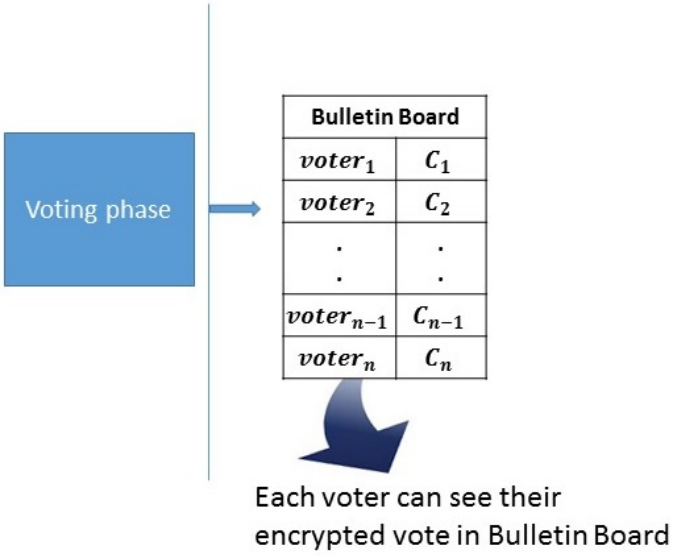


Figure 4.7: Helios bulletin board, where C_i means encrypted ballot from voter $voter_i$

Mixnet Operation

We have seen that bulletin board contains all the voted voters name and corresponding casted votes. If Helios directly decrypt the casted votes, the voter choices will be revealed and voter privacy will be compromised. To avoid this, Helios have to remove the mapping between voter and encrypted votes. Helios do this vote mixing process by using mixnet. A Mixnet consist of several mix servers, each managed by different parties. Each mix server re-encrypt and permute the set of votes. Internal details of mixnet working is explained earlier in section 2.3. In short, here mixnet transform a set of ciphertexts in bulletin board with voter identity to another set of ciphertexts without voter identity mapping.

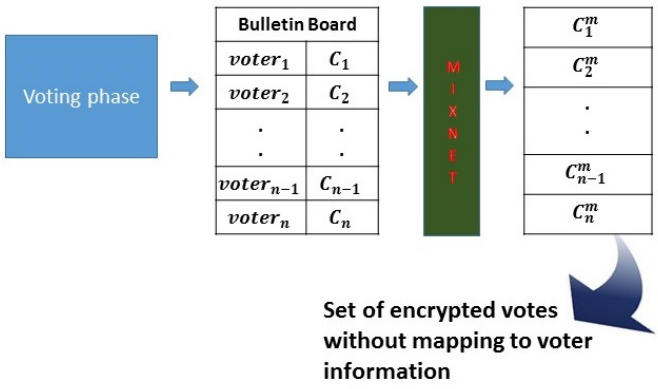


Figure 4.8: Helios removes mapping between voter and encrypted vote using mixnet, where C_i^m means i^{th} output from m^{th} mix server.

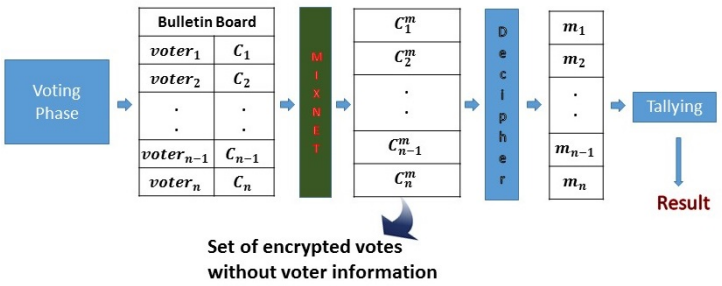


Figure 4.9: Helios Decryption and Tally process

Decryption and Tally

This is the final stage where Helios will decrypt all the votes using Helios server private key SK_s. Figure 4.9 shows an overall view. Note that Helios will not reveal the key even after completion of election process, because once private key is revealed, anybody can decrypt the public bulletin board announced earlier, that will compromise voter privacy. Instead of that, for each ciphertext vote C_i^j, Helios claim that plaintext is M_i and Helios prove this by using Chaum-Pederson protocol for proving discrete logarithm equality [8]. The proof is explained in Helios security analysis part.

4.2 Analysis of Helios Voting Scheme

In this section, we explain how the Helios scheme fulfill the security requirements of end to end verifiable voting scheme mentioned earlier in the chapter 3. The below explanations are the elaboration of security analysis given by Ben Adida in [1].

4.2.1 Voter Privacy

In Helios, Voter privacy is achieved as follows:

- Vote encryption
- Vote mixing

Vote Encryption

As explained in section 4.1.2, All votes are encrypted using ElGamal public key PK_s. This will ensure confidentiality. Note that, for simplicity we assumed that encryption key pairs are generated and Helios server keep secret the private key SK_s. But in practice, Helios use threshold encryption schemes, each party representative hold a part of key, so that single point dependency for confidentiality can be avoided.

Vote Mixing

Helios have to decrypt the encrypted votes, for the purpose of counting. But if Helios directly decrypt all ciphertexts from bulletin board, voters vote secrecy is compromised. To avoid this, Helios anonymize the ciphertext by the help of mixnet. Mixnet will remove mapping between voter and corresponding ciphertext of vote. Once mapping is removed, Helios will decrypt the ciphertexts and announce the result. Note that Helios does not embed any voter identification information in plaintext while ballot encryption process, so that there is no way to map the voter based on decrypted ciphertext.

4.2.2 Verifiability

Individual Verification

Voter have to get assurance that her vote is cast-as-intended and recorded-as-cast. Cast-as-intended assurance is gained by repeated ballot auditing explained in section 4.1.2. By repeated ballot auditing, voter gain confidence on Helios web client module. Recorded-as-cast assurance is gained by seeing the encrypted ballot in bulletin board. Bulletin board contains all voters identification information and corresponding ciphertext of vote. Each voter can verify that the ciphertext correspond to her name is same as what she cast. Figure 4.10 shows the bulletin board example.

Universal Verifiability

This is the most important requirement of end to end verifiable voting schemes. Any one from public should be able to verify the integrity of election. In Helios system, all the operation are transparent except three: Ballot encryption, Vote mixing and decryption. Anyone can verify the integrity of ballot encryption by repeated ballot auditing explained in section 4.1.2. In the vote mixing part, Helios facilitate proof of mixnet integrity, and in decryption part, Helios facilitate proof of correct decryption. Figure 4.11 shows the places where Helios need proof of integrity and proof correctness pictorially. Both are explained in detail below sections.

Proof of Mixnet Integrity

Why we need Proof of mixnet integrity

A mixnet consist of several mix servers. Each mix server receives a set of ciphertext and output another set of ciphertext, after re-encryption and permutation. To remove mapping between input set and output set, the re-encryption parameters and permutation used are kept secret by mix server. So each mix server act as a black box for external observer. To ensure that mix server haven't done any malfunction, such as adding new ciphertext or removing a ciphertext, Helios have to give proof of integrity. Note that each mix server used inside mixnet have to give

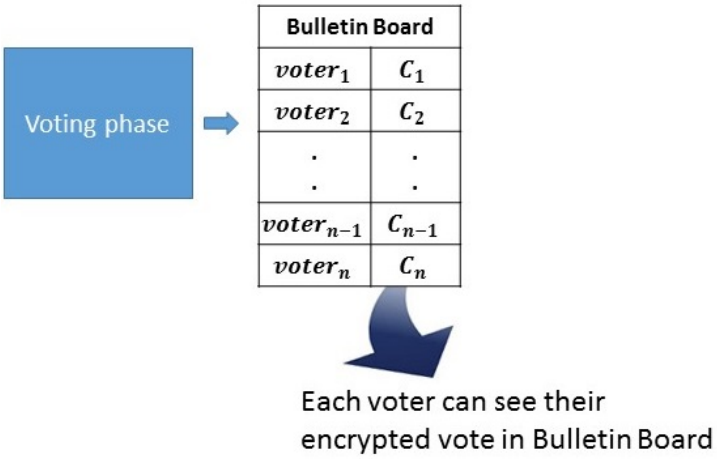


Figure 4.10: Voter can check the bulletin board and verify whether her vote is recorded-as-cast or not

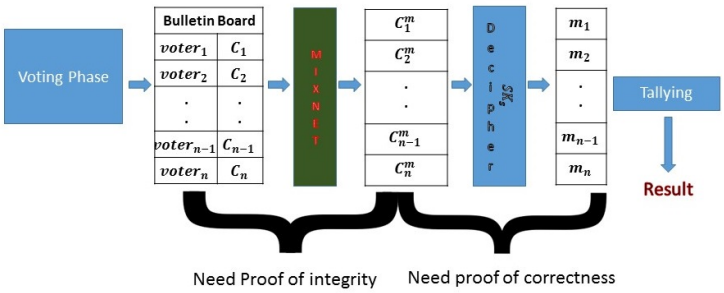


Figure 4.11: Figure shows the places where proof of integrity and proof of correctness required in Helios

proof of integrity. Figure 4.12 and 4.13 shows the inside view of mixnet and why we need proof of integrity.

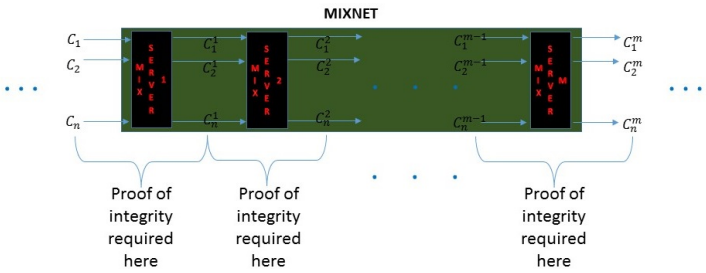
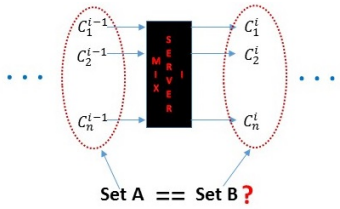


Figure 4.12: Figure shows the inside view of mixnet.

Proof of Mixnet Integrity

Here we explain the proof of mixnet integrity procedures in mixnet. The proof is based on Sako-Killian Shuffle and proof [14]. The core idea of the proof as follows: Each mix server will produce t shadow mix outputs. For each shadow output, the verifier challenges the mix server by asking either of the following question.

Question 1: Reveal the mapping between set A and shadow output.



We have to ensure that set B is exactly the permutation and re-encryption of set A elements. That means, malfunctions such as adding/ removing / replacing the votes didn't happen. This is challenging because, mix server will keep re-encryption exponent and permutation as secret (to hide the mapping between input and output)

Figure 4.13: Explanation to why we need proof of integrity.

Question 2: Reveal the mapping between set B and shadow output.
With malfunction, a mix server can make at most one answer correctly. This makes probability of mix server successful cheat without detection is less than half. This process will continue for t shadow outputs. So that probability of mixnet succeed in cheat without detection will reduce further.
Figure 4.14 shows the mix server and shadow outputs.

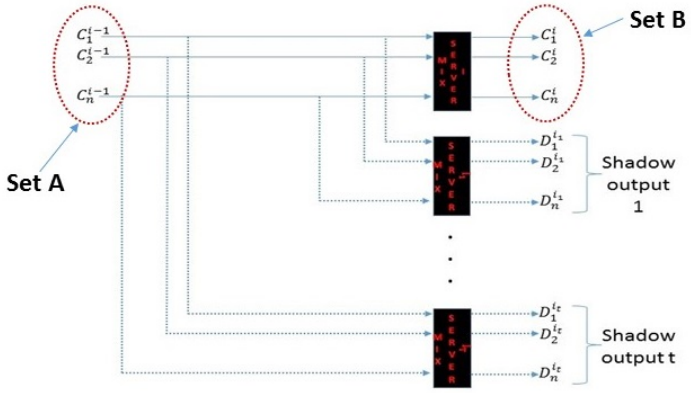


Figure 4.14: Mix servers and shadow mix outputs.

Let Mix_server_i received ciphertexts set A as input and output set B as shown in figure 4.15.

Then proof of mixnet integrity conducted as follows.

- Step 1: Mix_server_i generates t shadow mix outputs.
- Step 2: Verifier generates a random string, say rs, of 0's and 1's with size t.
- Step 3: Reveal mapping based on random bit. For $k = 1$ *to* t
  If k^{th} bit in rs is 0, Mix_server_i have to reveal mapping between set A and k^{th} shadow output.
  If k^{th} bit in rs is 1, Mix_server_i have to reveal mapping between set B and k^{th} shadow output.

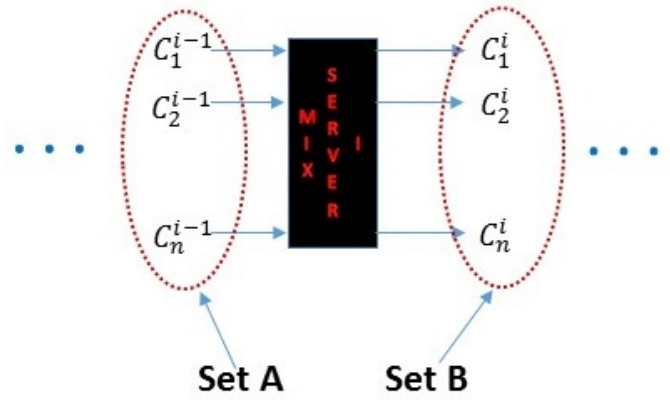


Figure 4.15: Mix_server_i receives set A and output set B.

If the Mix_server_i answered all the queries correctly, verifier can conclude that Mix_server_i is honest with a high probability. Following section explains each step in details.

Step 1: Mix_server_i generates t shadow mix outputs.

This is same as in Figure 4.14. Mix server receives set of ciphertexts set A and outputs set B as original mix output for next mix server and t dummy outputs for verifications.

Step 2: Verifier generates a random string, say rs, of 0's and 1's with size t.

Here verifier generates a random string of 0's and 1's of size t. Let us say random string generated is $rs = 0101...1101$.

Step 3: Reveal mapping based on random bit.

In the example case, first bit of rs is 0. So the Mix_server_i have to reveal mapping as shown in figure 4.16. Revealing mapping between set A and shadow output can be done by revealing the re-encryption exponents and permutation used while mixing. Figure 4.17 shows this pictorially. Here secrets revealed are set of re-encryption exponents $S^1 = s_1^i,\ s_2^i,\ ..., s_n^i$ and permutation $\pi_1 \in \Pi$. Now verifier can verify that set A is transformed in to 1st shadow output correctly.

In the example case, t^{th} bit of rs is 1.So the Mix_server_i mapping between set B and t^{th} dummy output as shown in figure 4.18. Here revealing mapping means, reveal the difference in re-encryption exponents and combined permutation. Figure 4.19 shows the difference in re-encryption exponent. Figure 4.20 shows combined permutation.

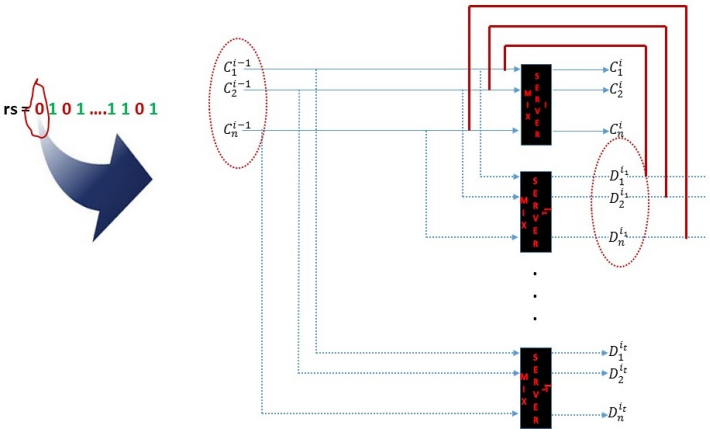


Figure 4.16: Mapping between set A and shadow output.

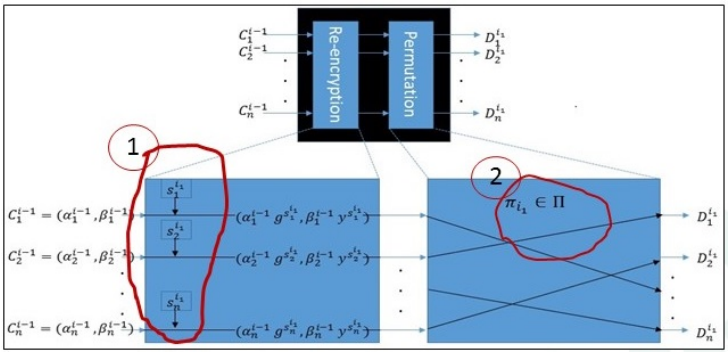


Figure 4.17: Mix_server_i revealing the secret parameters.

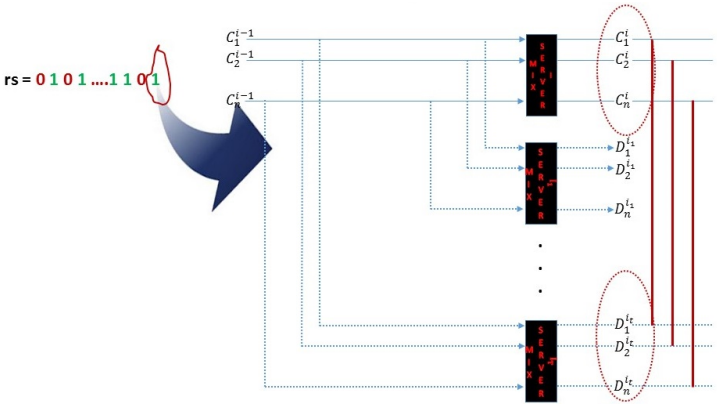


Figure 4.18: Mapping between set B and t^{th} shadow output.

Why it works

The idea is that, verifier can ask to reveal either of the two mapping, based on random bit chosen, if an adversarial mix server do malfunction, mix server either can answer

Question 1: Reveal the mapping between set A and shadow output.
Or

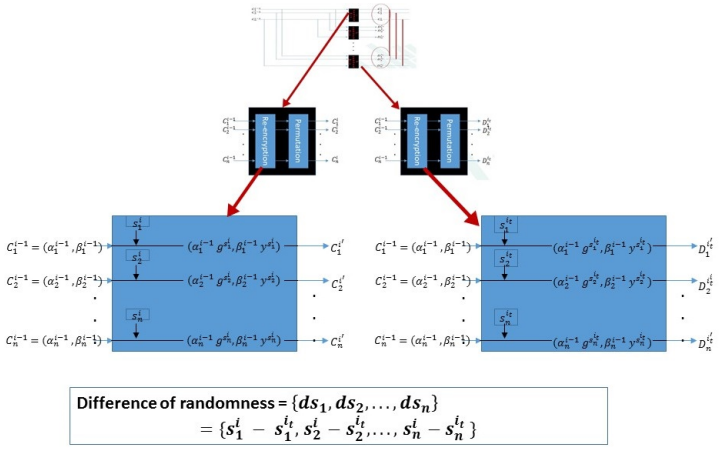


Figure 4.19: Re-encryption difference between set B and t^{th} shadow output .

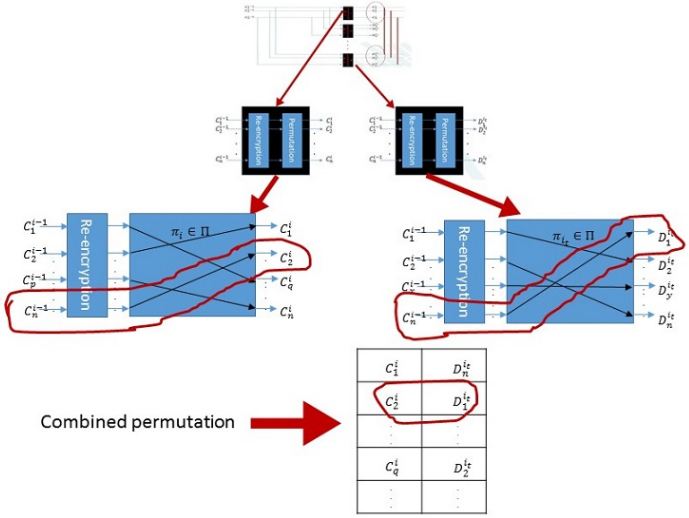


Figure 4.20: Mix_server_i revealing the combined permutation between set B and t^{th} shadow output.

Question 2: Reveal the mapping between set B and shadow output.
But not both. Since adversary doesn't know which question verifier is going to ask, chance of adversary to leave without detecting the cheating attempt is half, by checking only one shadow output. Below figures 4.21, 4.22, 4.23 4.24 explains this idea in details.

Note that mix server should note answer both the questions for same shadow mix output. If that happens, the mapping between mixnet input and output (set A and set B)will be revealed as shown in figure 4.25, that defeat the purpose of a mix server.

Above process have to repeat for each shadow output in a mix server, and also for each mix server in a mixnet. This is the way Helios scheme facilitate proof of mixnet integrity.

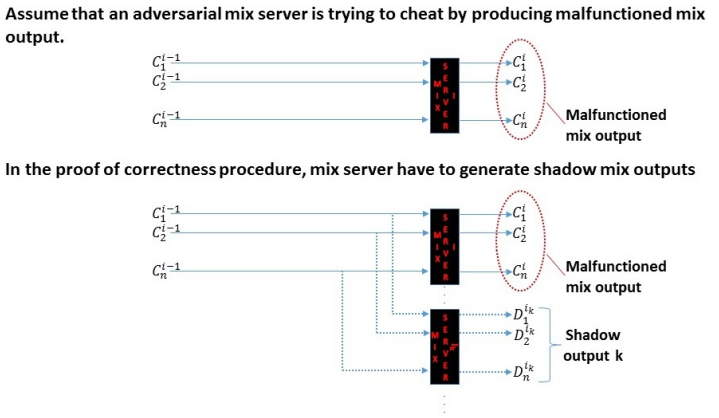


Figure 4.21: A mix server with malfunctioned mix server output and k^{th} shadow mix output.

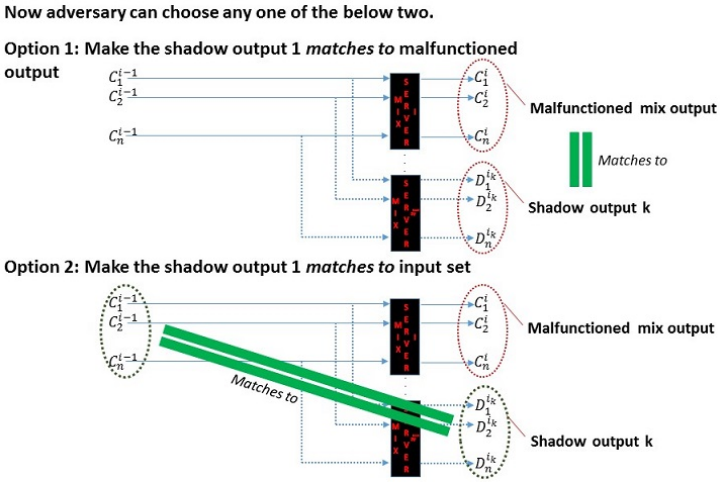


Figure 4.22: Figure show the choices available in front of a cheating mix server while generating shadow mix output.

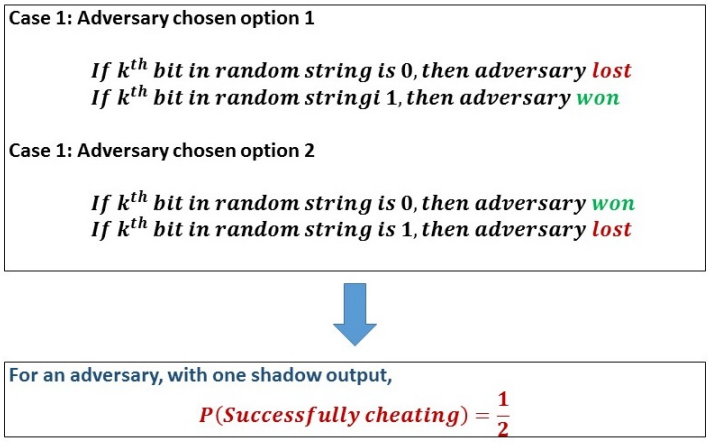


Figure 4.23: Probability of successful cheating by answering only one question of verifier.

Proof of Correct Decryption

For tallying, voting server have to decrypt all the encrypted votes. Obvious way is reveal the private key. But even if election process is over, Helios server should not reveal private key SK_s.

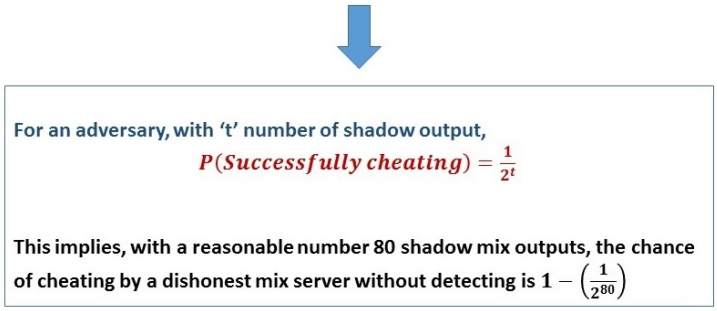


Figure 4.24: Probability of successful cheating by answering t questions from verifier.

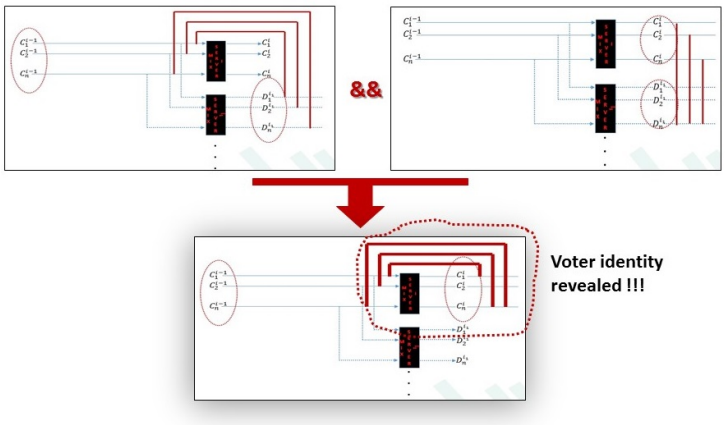


Figure 4.25: Figure shows, if a mix server answer both Qn 1 and Qn 2 for same shadow output, the input-output mapping will be revealed.

This is because, once key revealed, anybody can decrypt all ciphertexts from bulletin board. This will compromise voter privacy. So what Helios do is, Helios server read set of ciphertexts from mixnet output and output a set of plaintexts, and prove that the set of plaintext provided is exactly the decryption of set of input ciphertexts. This proof is called as proof of correct decryption. Helios proof of correct decryption is based on Chaum-Pederson protocol explained below.

Let x, y, g, q are be the ElGamal encryption scheme parameters explained in section 2.1. Server claims that plaintext m is the decryption of ElGamal ciphertext $c = (\alpha, \beta)$.
To prove this, server have to show that $log_g(y) = log_\alpha \beta/m$

1. Prover selects $\omega \in Z_q$ and sends $A = g^\omega$, $B = \alpha^\omega$ to the verifier.
2. The verifier challenges with $c \in Z_q$.
3. The prover responds with $t = \omega + xc$.
4. The verifier checks that $g^t = Ay^c$ and $\alpha^t = B(\beta/m)^c$.

4.2.3 Receipt Freeness

Voter should not be able to demonstrate that which candidate she have chosen. This is to prevent vote selling. Helios achieve this by using one time secret random number r while ballot encryption. The Helios web client module will concatenate r with the vote while ballot encryption. Due to this, voter cannot repeat the ballot encryption process herself, so she cannot demonstrate the candidate choice to a third party.

Section 4.1.2 explain Helios ballot encryption in details.

4.2.4 Eligibility

Only eligible voter should be allowed to cast vote, and one vote per voter. This is termed as eligibility. Helios ensure eligibility property by publishing bulletin board. Bulletin board contains all participated voter identification information and corresponding encrypted votes. Nobody can add or remove vote later, ensured by proof of integrity and proof of correctness. Any one from public can verify that, bulletin board consist votes from legitimate voters only and each voter have done only vote.

4.2.5 Fairness

Fairness means, No information about the vote count should be revealed before finishing voting phase. This is to ensure that voter decision are not influenced by voting system procedures. In case of Helios, vote decryption initiates only after completion of voting phase, so that no information of about partial tally while voting is going on. This way Helios ensures fairness.

Chapter 5

Our Proposals

Helios uses mixnet to remove the mapping between voter and encrypted vote. A mixnet consist of several mix servers managed by different parties. Mixnet take encrypted ballots from bulletin board as input, where voter to encrypted ballot mapping is clearly visible. And mixnet output another set of encrypted ballots, where mapping between voter and encrypted ballots does not exist. In this case, it is important to ensure that no cheating happened in between these input to output transformation. The cheating possibilities are, removing genuine encrypted ballot, adding a new encrypted ballot etc. To ensure the integrity here, Helios provide proof of mixnet integrity using Sako-Killian shuffle & proof method. But the Helios proof of integrity is cumbersome for a verifier. Here we propose two new schemes that facilitate proof of mixnet integrity procedure straight forward for verifiers. The schemes are,

1. Tweak based scheme
2. Dummy vote based scheme

Both the scheme follows overall same structure of Helios scheme. Both schemes follows web based server-client architecture. The major change from Helios is on proof of mixnet integrity part. The first one is based on adding tweak on ElGamal ciphertext. The second scheme make use of dummy votes to make integrity proof straight forward.

5.1 Tweak based Scheme

Here we propose a new proof of mixnet integrity method for ensuring integrity in mixnet based anonymization. The core idea is, add a hidden tweak in ElGamal ciphertext before passing through mixnet, so that mix servers or anybody else cannot add a valid ciphertext in to the set of encrypted ballots. Any attempt to replace a genuine ciphertext vote with a new ciphertext vote can be detected once decryption is done. Following section explains the proposed scheme in details.

5.1.1 Phase 1: Election Setup Phase

This phase is exactly same as Helios scheme that we explained earlier in 4.1.1. The voting scheme server will generate ElGamal key parameters, administrators setup the online ballot paper and voting server will send voter credentials to the eligible voters.

5.1.2 Phase 2: Voting Phase

In voting phase, the tweak based system have minor difference from Helios scheme in the ballot encryption. The tweak based system concatenate a fixed size length bit string of 1's with the vote V_i and random number r before encrypting. Figure 5.1 shows the difference between Helios and tweak based scheme. The additional fixed bit string of 1's is used for differentiate the valid votes from the invalid votes that added in between by adversary. Remaining steps in voting phase such as ballot auditing and ballot casting are same as in Helios scheme voting phase explained earlier in the section 4.1.2.

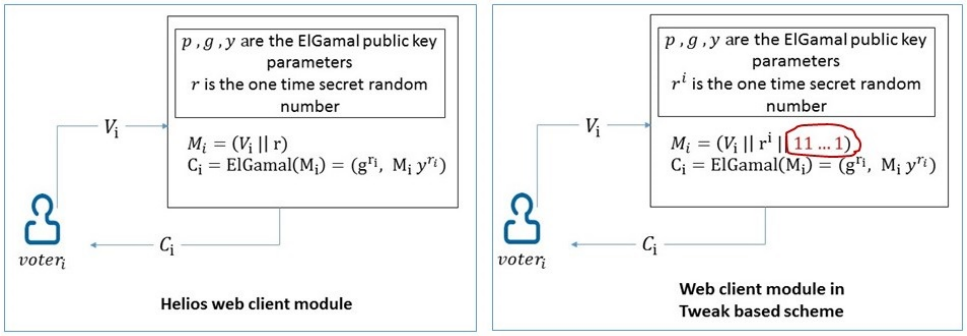


Figure 5.1: Figure shows the difference in ballot encryption between Helios and Tweak based scheme

5.1.3 Phase 3: Post Voting Phase

This is the phase where tweak based system is vary heavily from Helios scheme. Here the voting server have to decrypt and tally the casted votes, all the while preserving voter privacy and integrity. Likewise Helios, here also we use mixnet for anonymization, but we provide the proof of correctness in a better way by the help of hidden tweak. We can divide the post voting processes as following steps.

1. Publish the Bulletin board
2. Mix&Tweak network operation
3. Mixnet operation & Publish final Bulletin board
4. Reveal Mix&Tweak parameters
5. Remove Tweak variable

6. Decrypt and Tally

Below sections explain each steps in detail.

Publish the Bulletin board

This step is same as in Helios. Once voting phase is over, voting server will release bulletin board, a publicly accessible web page that contain all the participated voter names and corresponding encrypted ballot. Voters can confirm that their vote is recorded-as-cast, by seeing the ciphertext in bulletin board. Anybody from public can verify that all the votes are from eligible voters, by checking the voters identity. Figure 5.2 shows overall procedures happened until bulletin board.

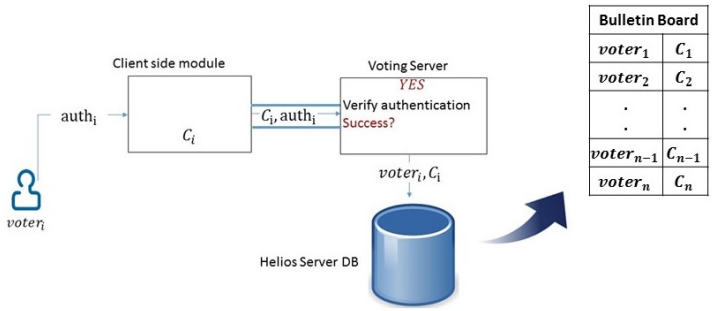


Figure 5.2: Figure shows the overall procedures until bulletin board. Here C_i is the ciphertext of ballot

Mix&Tweak Network Operation

In this step, the set of encrypted ballots from bulletin board will pass through mix&tweak network as shown in figure 5.4. Here mix stands for the mixnet that we discussed in section 2.3. The tweak operation is similar to ElGamal re-encryption. It is defined as follows.
Let $C = (\alpha, \beta) = (g^r mod\ p, (my^r) mod\ p)$ be a ElGamal ciphertext.
The tweaked ElGamal ciphertext $C'' = (\alpha'', \beta'') = (\alpha, (\beta\ y^Q) mod\ p)$.
Where $Q \in Z_q$ is the secret tweak value. Figure 5.3 shows the difference between tweak operation and re-encryption.

A mix&tweak network consist several mix&tweak servers, each one will be managed by election verifier or representatives of competing parties. Any one from public can act as a verifier. Each mix&tweak server will read the set of encrypted ballots from previous state and conduct re-encryption, permutation and tweak operation. Each mix&tweak server keep the parameters used for mix and tweak as secret for now. Figure 5.5 shows internals of a mix&tweak server.

Mixnet Operation & Publish Final Bulletin Board

In this step, the output from mix&tweak network will be passed through mixnet, same as in Helios. The output from mixnet will be announced publicly, named as final bulletin board. This

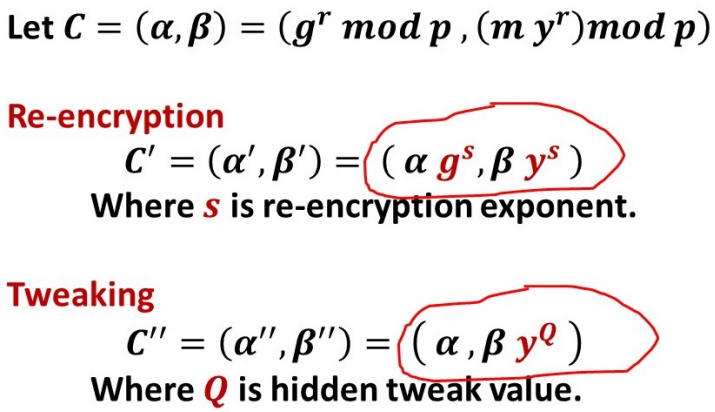


Figure 5.3: Figure shows the difference between tweak and re-encryption

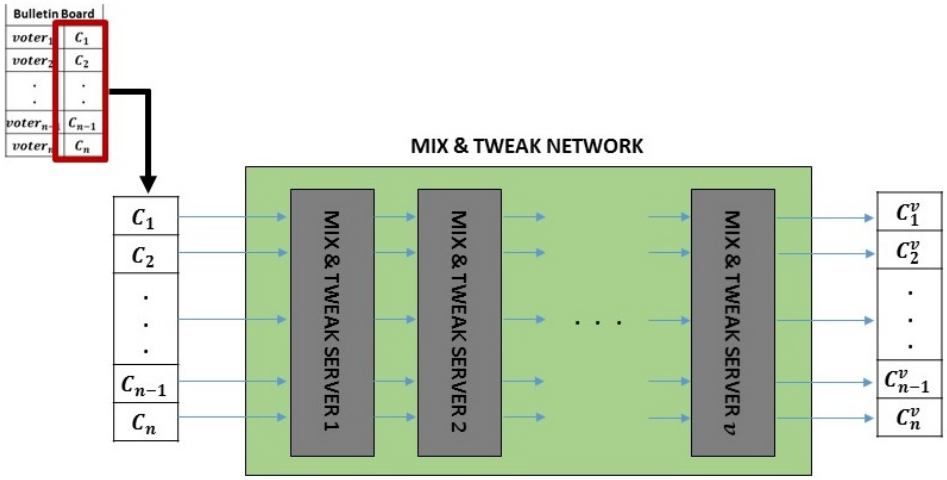


Figure 5.4: Overview of a Mix&Tweak network. Here C_i is the encrypted ballot from $voter_i$, n is the total number of encrypted ballots, v is the number of mix&tweak servers and C_j^v means j^{th} output from v^{th} mix&tweak server.

final bulletin board contains ciphertexts of votes without any mapping information to the voter identity.

Figure 5.6 shows the overview of the procedures.

Reveal Mix&Tweak Parameters

The final bulletin board is in public now. In this step, each mix&tweak server will reveal the secret variables that used while mix and tweak operation. The variables are, re-encryption exponents, permutation and tweak value. Now the operations happened inside mix&tweak network is transparent, anybody can verify that no malfunction happened while mix&tweak operation. Figure 5.7 shows the secret variables in j^{th} mix&tweak server.

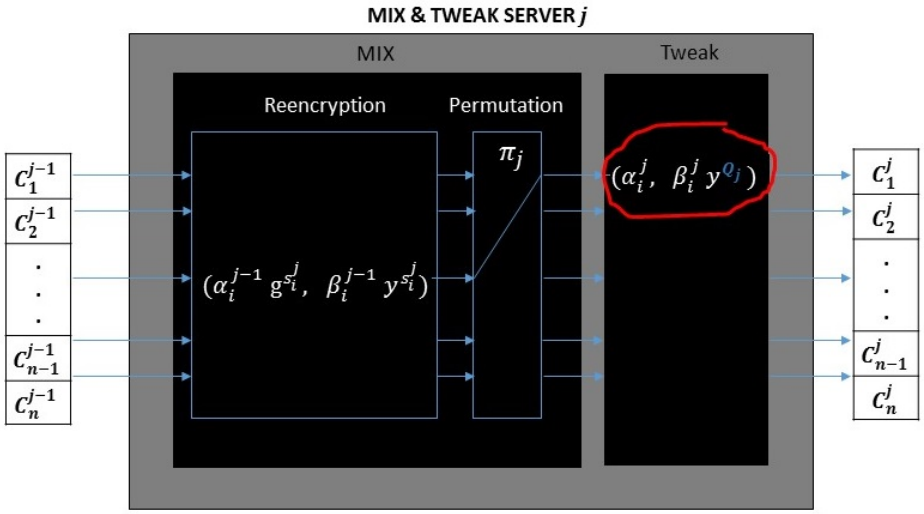


Figure 5.5: Inside view of a single Mix&tweak server. Here current mix&tweak server number is j, C_i^{j-1} means i^{th} output from previous mix&tweak server, $s_i^j \in Z_q$ is the i^{th} re-encryption exponent in mix&tweak server j, π_j is the randomly chosen permutation by mix&tweak server j, $Q_j \in Zq$ is the secret tweak value by mix&tweak server j.

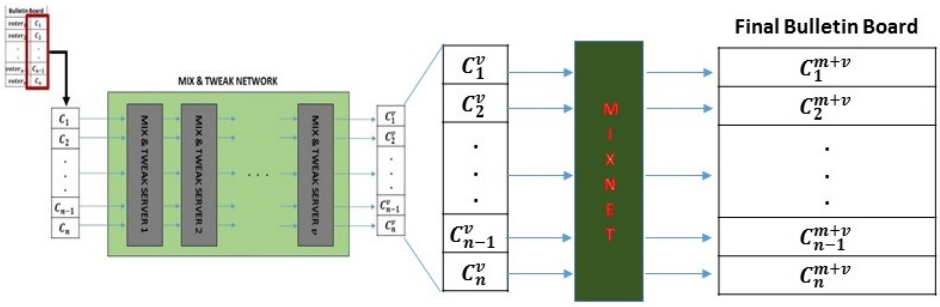


Figure 5.6: Mix then publish Bulletin board. Here v is the number of mix&tweak servers inside mix&tweak network and m is the number of mix servers inside mixnet.

*

Remove Tweak Variable

Now the secret variables used by mix&tweak servers are in public. In this step, system will remove all the tweak values added by mix&tweak servers. Let $C = (\alpha, \beta)$ be the ElGamal ciphertext and Q be the secret tweak value added earlier. Removing the tweak value is done as follows,

Tweak removed ciphertext $C'' = (\alpha'', \beta'') = (\alpha,\ (\beta\ (y^Q)^{-1}) mod\ p)$.

Note that in our scheme, different tweak value is added by more than many mix&servers. So we have to remove all from the ciphertext. Figure 5.8 shows the removal of all tweak values from the ciphertext.

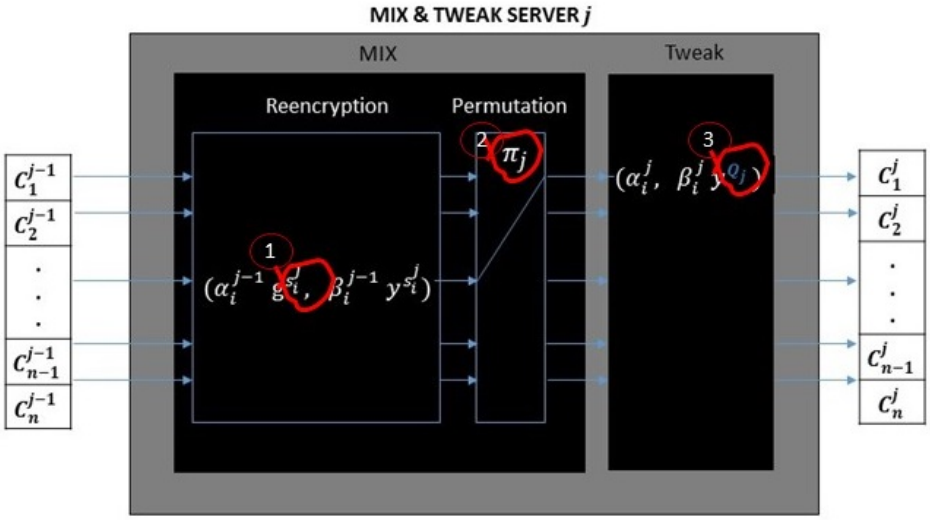


Figure 5.7: Reveal Mix & Tweak details

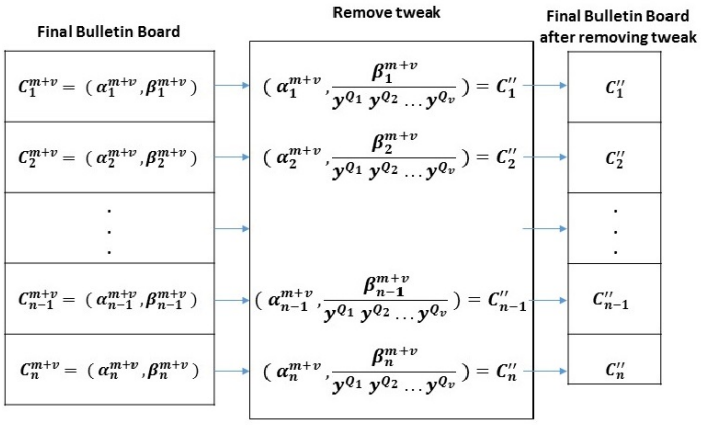


Figure 5.8: Removal of tweak values from the ciphertext. Note that all operations are in modular arithmetic with *mod p*.

Decrypt and Tally

Now we have a set of ElGamal ciphertexts of votes without mapping to corresponding voter identity. The voting server will decrypt the ciphertexts using ElGamal private key SK_s generated earlier. The decryption operation is same as standard ElGamal decryption. Once decryption operation have done, plaintext votes are available in public, server will count and publish the result. Any one from public can view and tally the plaintext votes. Figure 5.9 shows the decryption and tallying process.

One important note here is, same as in Helios, voting server will never reveal the ElGamal private key used. This is because, once private key is in public, anybody can decrypt initial bulletin board, where mapping from voter identity to the ciphertext is clearly available.

Since voting server do not publish private key even after election, server have to give proof of correctness for decryption. That means, server shows ciphertext C and plaintext M, and proves

that M is the decryption of C, without revealing private key. This is same as in proof of correct decryption in Helios scheme explained in section 4.2.2.

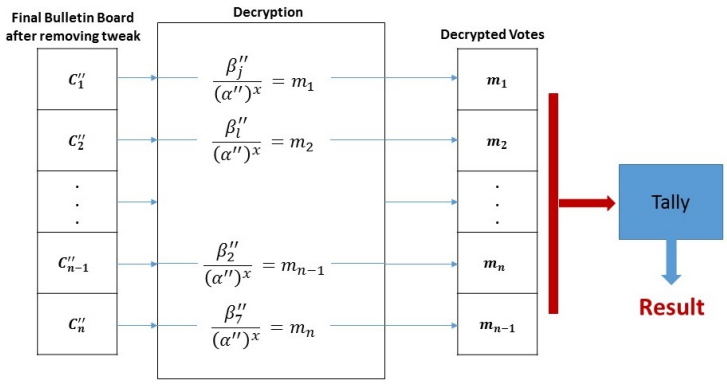


Figure 5.9: Decryption and Tally operation

5.2 Dummy Vote based Scheme

In this scheme, the use of dummy vote is the major change from existing Helios scheme. Here the cast vote can be either a dummy or genuine vote. Only eligible voters are allowed to cast the vote. Each voter will cast a fixed number of votes. Throughout this dissertation, we consider each voter cast five votes. Out of this five, one will be genuine vote and remaining all will be dummy votes. The key pair used for encrypting dummy votes will be different from key pair used for genuine vote encryption. Given a set of ciphertexts of votes, an adversary cannot distinguish between dummy and genuine vote.

The overall idea is, set of ciphertexts from bulletin board, that contains dummy votes and genuine votes will be passed through mixnet. Once mixnet operations have done, server will release private key that can be used for decrypting all dummy votes. Now anybody can decrypt all the dummy votes from mixnet input and mixnet output. If the dummy votes in both sets are same, the integrity can be guaranteed with very high probability. This is because, adversary cannot distinguish dummy vote from genuine votes. So any attempt to manipulate the ciphertexts while mixing can affect set of dummy votes also, with a high probability.

We can divide whole election process as three phases such as election setup phase, voting phase and at last post voting phase. Following sections explain the proposed scheme in details.

5.2.1 Phase 1: Election Setup Phase

As we have seen in earlier schemes, this phase the mostly administrative part. Here voting authority have to do following tasks.

Generate Encryption Keys

Our scheme require two asymmetric key pairs. One key pair is for genuine votes confidentiality and another one is for dummy vote confidentiality. We are using ElGamal encryption scheme here. We can use threshold encryption to avoid single point dependency for integrity. But for simplicity, here we consider that voting server will generate and keep secure the below pair of private keys.

Server Public Key for Genuine vote encryption: $PK1_s$
Server Private Key for Genuine vote decryption: $SK1_s$
Server Public Key for Dummy vote encryption: $PK2_s$
Server Private Key for Dummy vote decryption: $SK2_s$

Setup the Ballot

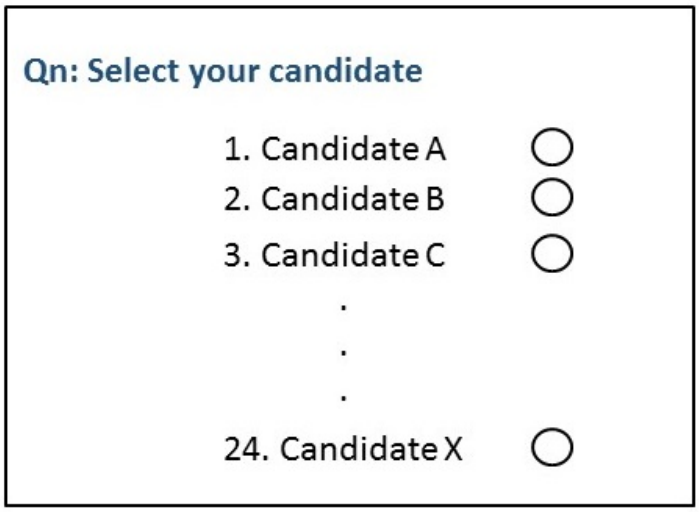


Figure 5.10: Figure shows an hard copy version of an example ballot. In our actual scheme, ballot will be displayed in web page

This step is same as of Helios. Here administrator have to setup the ballot. A ballot consist the names of contesting candidates. Figure 5.10 shows an example ballot. In our actual scheme, ballot will be displayed in web page and voter can choose the candidate by mouse click.

Generate and Send Voter Authentication Credentials

Creating voters list is administrative task. Here we assume that there have a list of genuine voters name and valid mail id is available already. This list will be feed in to the voting server.

Then the voting sever will generate credentials for each voter and send to corresponding mail id. The voter will use this credentials for authentication while vote casting. Figure 4.2 explain the overall process here.

5.2.2 Phase 2: Voting Phase

This is the phase where actual voting happen. Each voter will cast a fixed number of votes, out of these one will be genuine vote. Once voter marked the choice, web client module will encrypt the ballot after concatenating vote with a secret one time random number. By ballot auditing, voter can gain assurance that her vote is properly encrypted. The overall control flow of these phase is same as in 4.1.2 Each voter have to repeat this process five times (five votes per voter).

Selecting Candidate & Encrypting the Ballot

Voter will choose to do either genuine vote or dummy vote. The web client encrypt the ballot accordingly. In case voter chosen for genuine vote, ballot encryption procedure as follows:

1. Voter $voter_i$ choose her vote V_i^j. Where V_i^j stands for i^{th} voters j^{th} vote.
2. Web client module will generate secret one time random number r.
3. Web client module encrypt the ballot as follows:
   Encrypted Genuine vote: $C_i^j = ElGamal(PK1_s, V_i^j||r^{i_j}||111...1)$

In case voter chosen to do dummy vote, ballot encryption procedure as follows:

1. Voter $voter_i$ choose her vote V_i^j. Where V_i^j stands for i^{th} voters j^{th} vote.
2. Web client module will generate secret one time random number r.
3. Web client module encrypt the ballot as follows:
   Encrypted Dummy vote: $C_i^j = ElGamal(PK2_s, V_i^j||r^{i_j}||000...0)$

Note that, genuine vote is concatenated with a fixed length string of 1's and encrypted with public key $PK1_s$, and the dummy vote is concatenated with fixed length string of 0's and encrypted with public key $PK2_S$. Figure 5.11 shows encryption of a genuine vote by web client module. The one time secret random number r^{i_j} is used to ensure receipt freeness. Without knowing r^{i_j}, voter cannot demonstrate to anybody that how she voted. This prevents voters from vote selling.

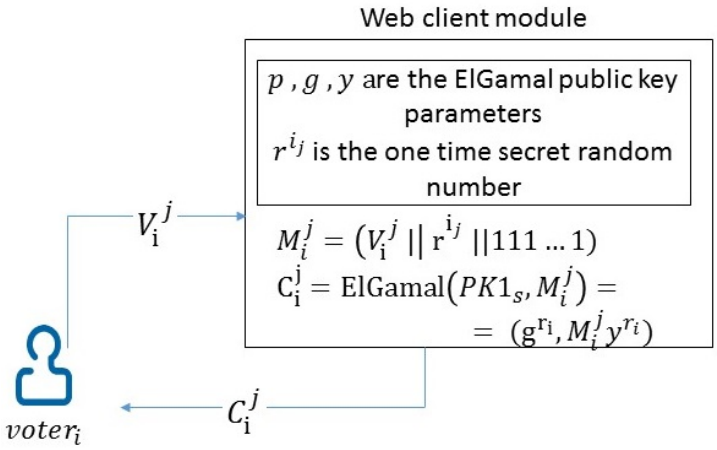


Figure 5.11: Figure shows encryption of a genuine vote.

Ballot Auditing

Now voter have the encrypted ballot. But due to the use of secret random number r^{i_j}, voter cannot verify the correctness of encryption done by web client module. It is possible that a web client is cheating the voter by changing the vote choice she made. To ensure that vote is correctly encrypted, voter can choose ballot auditing. The procedure is simple, web client release the random secret number r^{i_j} that concatenated with vote before encryption. Now the voter knows all parameters used for encryption, she can verify the correctness by executing encryption operation in a trusted machine. If the web client is honest, the ciphertext generated by trusted machine will be same as the ciphertext from returned by web client module. Voter cannot cast the ballot that once audited. This is because, while auditing, random secret r^{i_j} is revealed, this facilitate voter to demonstrate how she voted. This must be avoided. So the voter have to repeat from previous step, selecting candidate & creating the ballot.

Ballot Casting

Here the voter will cast her votes same as in Helios scheme explained in figure 4.6. One difference is that, voter have to repeat this process five times (four dummy votes and one genuine one).

5.2.3 Phase 3: Post Voting Phase

The main task of this phase is decryption and tally. But same time we have to ensure the election integrity and voter privacy. Our proposed scheme will do following steps:

1. Publish the bulletin board
2. MIXNET Operation with $PK1_s$
3. Decrypt and Tally

4. Reveal Private key $SK2_s$

Below sections explain each steps in details.

Publish the Bulletin Board

Here the voting server release the bulletin board as shown in figure 5.12. Figure 5.13 shows an overall view of bulletin board releasing. Bulletin board contains name and votes of each voter. In this scheme, there will be five votes corresponds to each voter, one is genuine and remaining four are dummy votes. But except voter, no one else know which one is genuine and which all are dummy.

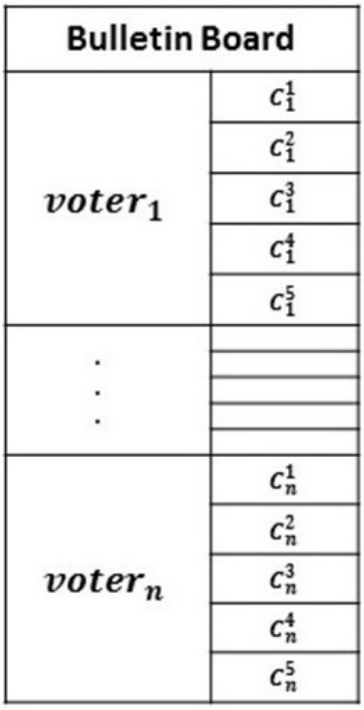


Figure 5.12: A single bulletin board.

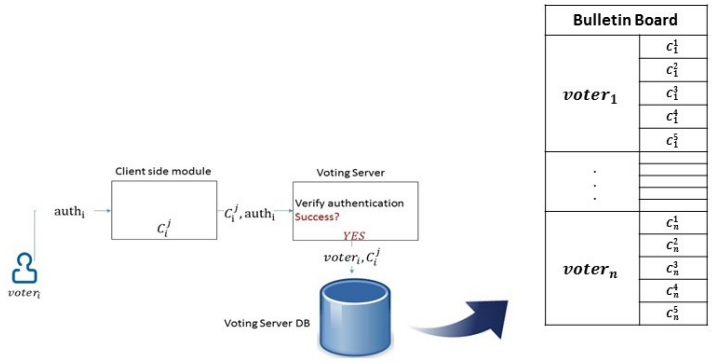


Figure 5.13: Figure shows overall view of bulletin board releasing.

Mixnet Operation with $PK1_s$

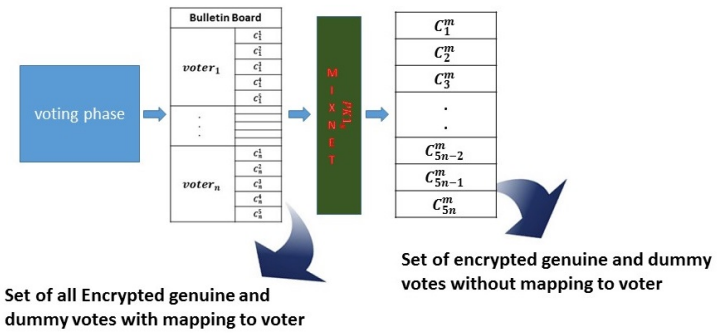


Figure 5.14: Mixnet operation. Mixnet receives set of ciphertexts from bulletin board as input and output another set of ciphertext as output. While re-encryption, mixnet use public key $PK1_s$.

The aim of this step is anonymization. Mixnet receive the set of ciphertexts from bulletin board and output another set of ciphertext. The figure 5.14 shows the overall view of anonymization process using mixnet. Mixnet consist of sequence of mix servers. Each mix server conduct re-encryption and permutation operation. Details of mixnet is explained in the section 2.3. Note that, here the set of ciphertexts consist of both genuine vote ciphertexts and dummy vote ciphertexts. Genuine votes are encrypted with $PK1_s$ and dummy votes are encrypted with $PK2_s$. While re-encryption, mixnet uses $PK1_s$ for all ciphertexts. This will damage the dummy vote ciphertexts.

Decrypt and Tally

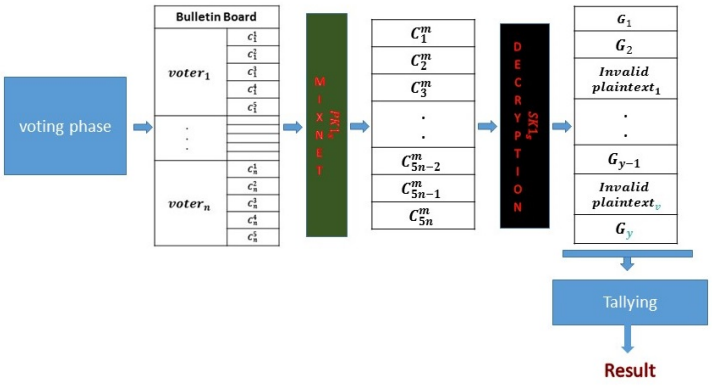


Figure 5.15: Decryption and Tally operation. Here G_i means i^{th} genuine vote plaintext.

This is the final step where server will decrypt all the ciphertexts with private key $SK1_s$. Figure 5.15 shows the decryption and tally operation overview. The decryption will result genuine vote plaintexts and invalid plaintexts. The genuine vote plaintexts are the decryption of genuine vote ciphertexts. The invalid plaintexts are the decryption of dummy vote ciphertexts. Now genuine votes are available for public in plaintext format. Anyone can tally the result.
Note that, same as in Helios, the voting server will not reveal the secret key $SK1_s$. This is

because, once $SK1_s$ is in public, anyone can decrypt the ciphertexts directly from bulletin board. This will break voter privacy. So here voting server read set of ciphertexts and output decrypted set of plaintexts, and proves that plaintexts are exactly the decryption of input ciphertexts. The proof is same as Helios proof of correct decryption explained in section 4.1.3.

Reveal the Private Key $SK2_s$

Here the voting server will reveal the private key $SK2_s$. Now anybody can decrypt the dummy votes from bulletin board correctly. It is indistinguishable between genuine vote ciphertext and dummy vote ciphertext from bulletin board. So the person who decrypt will decrypt all the ciphertexts, then identify the dummy votes based on the resulting plaintext content.
This step is required to provide proof of mixnet integrity. How does the proof of mixnet integrity provided by revealing $SK2_s$ is explained in dummy vote analysis section 6.2.1.

Chapter 6

Analysis of Proposed Voting Schemes

In this chapter, we analyze the proposed schemes based on requirements mentioned in the chapter 3.

6.1 Analysis of Tweak Based Scheme

Security analysis of tweak based scheme is same as in Helios, except proof of mixnet integrity part. Proof of integrity in tweak based scheme is explained below.

6.1.1 Proof of Mixnet Integrity in Tweak Based Scheme

Each mix server in the mixnet receives set of ciphertexts as input and output another set of ciphertexts. A honest mix server will transform the input set to output set by conducting re-encryption and permutation operation. While mixing operation, an adversarial mix server try to manipulate the election tally by following four ways.

1. Add a new ciphertext.
2. Remove a existing ciphertext.
3. Replace an existing ciphertext with a new ciphertext.
4. Replace an existing ciphertext with another existing ciphertext.

In tweak based scheme, the core idea behind proof of mixnet integrity is, due to the presence of secret tweak value in ciphertext, an adversarial mix server cannot generate a valid ciphertext while mixing operation is going on. The proof of mixnet integrity must ensure that none of the above four happened while mixnet operation. Below sections explain each in detail.

Add a New Ciphertext

Adding a new ciphertext can be detected easily by comparing the size of mixnet input set and mixnet output set. A mixnet receives a set of ciphertexts as input and output another set of ciphertexts. Let assume that size of input set is n. And a dishonest mix server present in the mixnet added i number of new ciphertext while mixing operation. Then the size of output set must be $n + i$. That means extra i ciphertexts. By seeing this, we can conclude that mixnet integrity is broken.

Remove an Existing Ciphertext

Removing the ciphertext can be easily detected by comparing the size of mixnet input and mixnet output. Let assume that size of input set is n. And a dishonest mix server present in the mixnet removed i number of existing ciphertext while mixing operation. Then the size of output set must be $n - i$. That means i ciphertexts are less in output. By seeing this, we can conclude that mixnet integrity is broken.

Replace an Existing Ciphertext with a New Ciphertext

Assume that a dishonest mix server present in the mixnet replaced an existing ciphertext with new one. Note that this cannot be detected by comparing the size of mixnet input and output. This type of vote manipulation attempt can be detected once all the votes are decrypted as shown in figure 6.1. The idea is that, since added tweak values are secret while mixing operation, only way a dishonest mix server can do is, generate a random ciphertext and hope that it will decrypt in to a valid plaintext. But since the size of set of all valid plaintexts are much less than size of set of all possible ciphertexts, probability of a randomly generated ciphertext decrypt in to a valid plaintext is very low. So that the chance of this kind of vote manipulation happen without detecting is very low.

Replace an Existing Ciphertext with another Existing Ciphertext

Here a dishonest mix server replace a ciphertext with another existing one. That means duplication. This can be detected once all votes are decrypted, by checking whether is there have any duplicate plaintext present or not. Note that, since we are concatenating random number r with vote while ballot encryption, unless there is manipulation attempt, the chance of two same plaintext in the set of decrypted vote will be very low (Two different voters vote ciphertext decrypt to same plaintext only when both voters chosen same candidate and client module generated same random number. The probability of choosing same random number can be reduced by increasing the size of random number range).

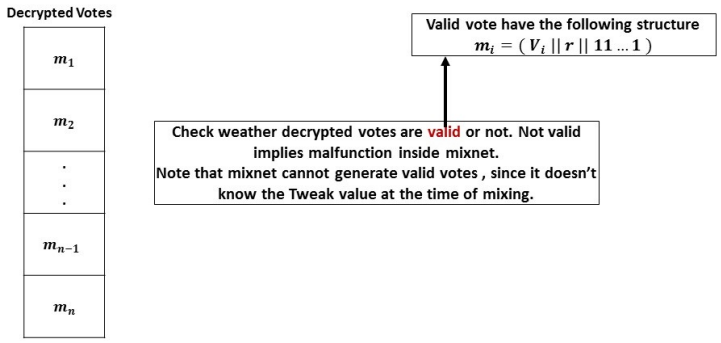


Figure 6.1: Proof of mixnet integrity: Detecting ciphertext replacement by checking validity of decrypted plaintext.

6.2 Analysis of Dummy Vote Based Scheme

Except the proof of mixnet integrity and democracy, dummy vote based scheme security analysis is same as Helios scheme explained earlier. Below sections explains how dummy vote based scheme facilitate proof of mixnet integrity and democracy.

6.2.1 Proof of Mixnet Integrity in Dummy Vote Based Scheme

The need of proof of mixnet integrity is to ensure that no manipulation happened while mixing operation. In dummy vote based scheme, the mixnet input set contains genuine vote ciphertexts and dummy vote ciphertexts. Genuine votes are encrypted with $SK1_s$ and dummy votes are encrypted with $SK2_s$. By analyzing the ciphertext, it is indistinguishable between genuine vote ciphertext and dummy vote ciphertext.

The mixnet receives a set of ciphertexts as input and output another set of ciphertexts as output. The proof of mixnet integrity procedure is as follows.

1. Voting server decrypt all ciphertexts in mixnet output set using $SK1_s$, as shown in figure 6.3.
2. Voting server release the dummy vote decryption key $SK2_s$. Now anyone can decrypt the dummy vote ciphertexts in bulletin board correctly as shown in figure6.2.
3. As shown in figure 6.4, Let v be the number of invalid plaintexts in mixnet output decryption with key $SK1_s$ and let x be the number of dummy vote plaintexts in bulletin board decryption with $SK2_s$. If $x \neq v$ means manipulation happened. Otherwise, manipulation not happened (with a high probability).

Figure 6.4 explain above procedure pictorially. Note that, in step 1, the decryption will result two kind of outputs, one is genuine vote plaintext and other one is invalid plaintexts (While

mixnet operation, dummy vote ciphertexts are damaged due to re-encryption with $PK1_s$. Moreover, here decrypting dummy vote ciphertexts with wrong key. This results invalid plaintexts). Genuine votes can be identified based on the format, that means genuine votes contains a sequence of 1's as explained in section 5.2.2. In step 2, the decryption will result two kind of outputs, one is dummy vote plaintext and other one is invalid plaintexts (Decrypting genuine vote ciphertexts with wrong key results invalid plaintext). Dummy votes can be identified based on the format, that means dummy votes contains a sequence of 0's as explained in section 5.2.2.

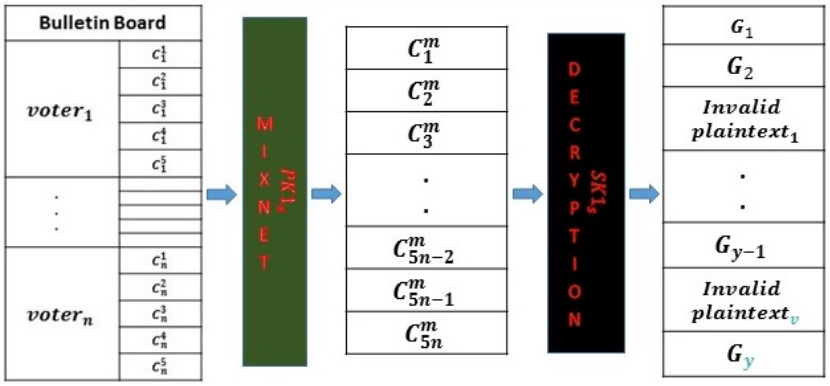


Figure 6.2: Voting server decrypt all ciphertexts in mixnet output set using $SK1_s$. Here G_i meas i^{th} genuine vote plaintext. The decryption of genuine vote ciphertext results genuine vote plain text. The decryption of dummy vote ciphertext results invalid plaintext.

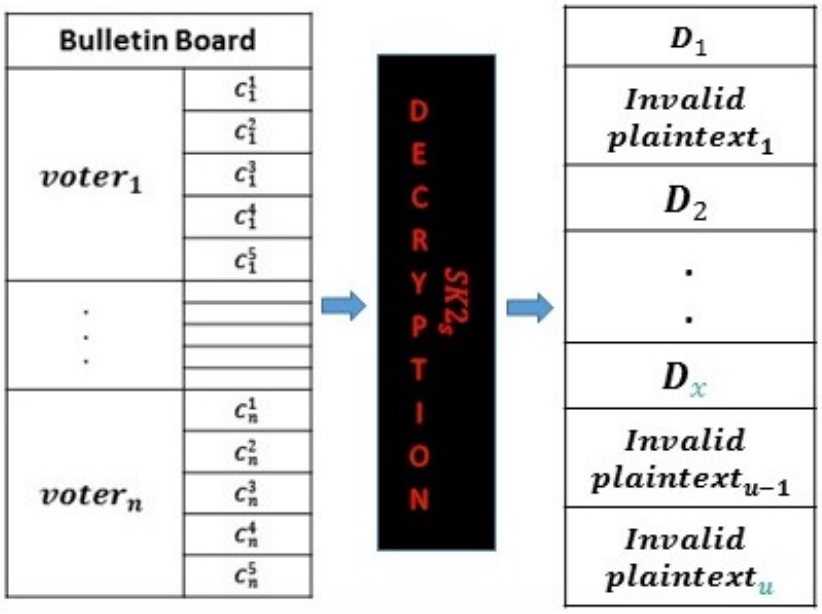


Figure 6.3: Decryption of bulletin board using $SK2_s$. Here D_i means i^{th} dummy vote plaintext. The decryption of genuine vote ciphertext results invalid plaintext, due to wrong key decryption. The decryption of dummy vote ciphertext results dummy vote plaintext.

Now let us see how the above procedure ensure mixnet integrity. Assume that a dishonest mix

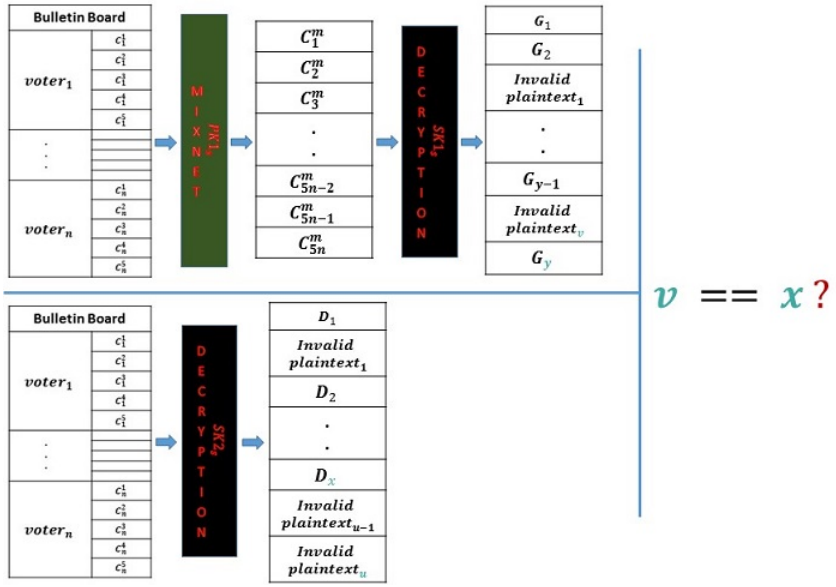


Figure 6.4: Proof of mixnet integrity can be verified by checking whether x equals to v or not. Where x is the number of dummy vote plaintexts and v is the number of invalid plaintexts.

server present in mixnet is trying to manipulate the vote ciphertext. The mix servers cannot distinguish between genuine vote ciphertext and dummy vote ciphertext. Dishonest mix server can try to manipulate by following five ways.

1. Add a new ciphertext.
2. Remove a existing ciphertext.
3. Replace an existing ciphertext with a new genuine ciphertext.
4. Replace an existing ciphertext with a new dummy ciphertext.
5. Replace an existing ciphertext with another existing ciphertext.

The proof of mixnet integrity must ensure that none of the above four happened while mixnet operation. Below sections explain each in detail.

Add a New Ciphertext

Adding a new ciphertext can be detected easily by comparing the size of mixnet input set and mixnet output set. A mixnet receives a set of ciphertexts as input and output another set of ciphertexts. Let assume that size of input set is n. And a dishonest mix server present in the mixnet added i number of new ciphertext while mixing operation. Then the size of output set must be $n + i$. That means extra i ciphertexts. By seeing this, we can conclude that mixnet integrity is broken.

Remove an Existing Ciphertext

Removing the ciphertext can be easily detected by comparing the size of mixnet input and mixnet output. Let assume that size of input set is n. And a dishonest mix server present in the mixnet removed i number of existing ciphertext while mixing operation. Then the size of output set must be $n - i$. That means i ciphertexts are less in output. By seeing this, we can conclude that mixnet integrity is broken.

Replace an Existing Ciphertext with a New Genuine Ciphertext

Assume that a dishonest mix server present in the mixnet replaced an existing ciphertext with new genuine ciphertext. This type of vote manipulation attempt can be detected by checking whether v is equals to x or not as shown in figure 6.4. Since mix server cannot differentiate dummy vote ciphertext from genuine vote ciphertext, dishonest mix server have to choose one ciphertext randomly to replace. If the ciphertext randomly chosen is of dummy vote, then the value of v will be one less than x while comparing v and x as shown in figure 6.4. So the manipulation attempt detected. If the ciphertext randomly chosen is of genuine vote, then the dishonest mix server succeeded in manipulating one vote. Note that, the mixnet input set contains dummy vote ciphertexts five times more than genuine vote ciphertexts. So the probability that the randomly chosen ciphertext is of genuine vote is 1/5. This means, Probability of manipulating one genuine vote without detection is 1/5. Similarly, for manipulating 100 votes, the probability of manipulating 100 genuine votes without detection is $1/5^{100}$, negligible probability.

Replace an Existing Ciphertext with a New Dummy Ciphertext

Assume that a dishonest mix server present in the mixnet replaced an existing ciphertext with new dummy ciphertext. Since mix server cannot differentiate dummy vote ciphertext from genuine vote ciphertext, dishonest mix server have to choose one ciphertext randomly to replace. If the randomly chosen ciphertext of dummy vote, then a dummy ciphertext replace with another dummy ciphertext. This will not make any change in final tally. If the randomly chosen ciphertext is of genuine vote, then the value of v will be increased by one compare to x, while comparison as shown in figure 6.4. So the manipulation attempt detected. In short, by replacing an existing ciphertext with new another dummy vote ciphertext, dishonest mix server cannot change the final tally without detection.

Replace an Existing Ciphertext with another Existing Ciphertext

Here a dishonest mix server replace a ciphertext with another existing one. That means duplication. This can be detected once all votes are decrypted, by checking whether is there have any duplicate plaintext present or not. Note that, since we are concatenating random number r with vote while ballot encryption, unless there is manipulation attempt, the chance of two same

plaintext in the set of decrypted vote will be very low (Two different voters vote ciphertext decrypt to same plaintext only when both voters chosen same candidate and client module generated same random number. The probability of choosing same random number can be reduced by increasing the size of random number range).

6.2.2 Democracy

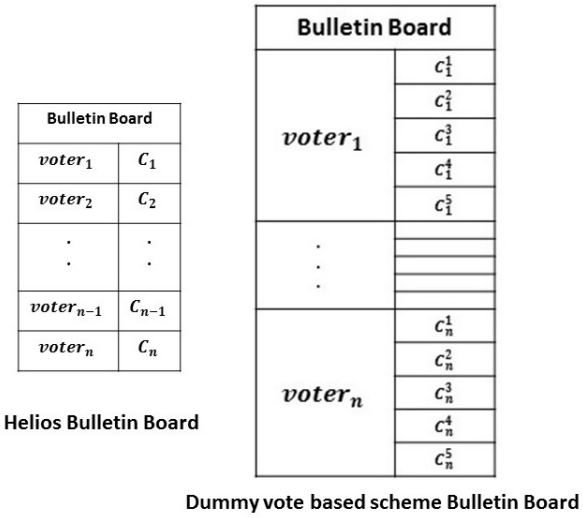
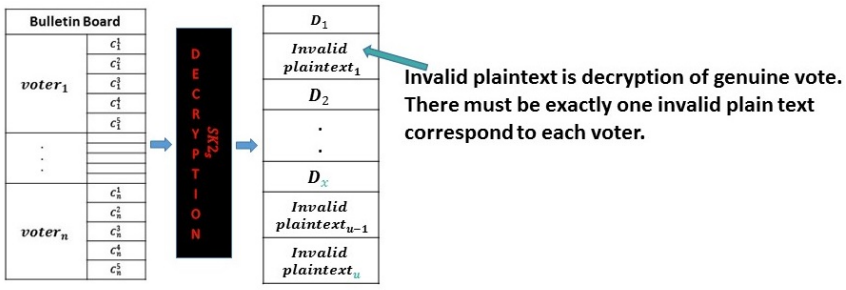


Figure 6.5: Figure shows the Bulletin boards used in Helios and dummy vote based scheme .

Like in Helios, dummy vote based scheme also shows bulletin board. Figure **??** shows the bulletin boards in Helios scheme and dummy vote based scheme. The difference is that, corresponding to each voter, there will be five votes present in dummy vote based scheme bulletin board. So here we have to ensure that no voter haven't done more than one vote. This can be done as follows.



1. Voting server release the dummy vote decryption key $SK2_s$ once mixnet operation is over.
2. Decrypt all the ciphertexts in the bulletin board using $SK2_s$.
3. Corresponding to each voter, there must be four dummy vote plaintexts and one invalid plaintext (decryption of genuine vote with wrong key). Number of dummy votes corre-

sponds to a voter is less than four means, voter have done more than one genuine vote (since the ballot encryption is done by voter web client module, there is no chance for invalid dummy votes).

If more than one genuine vote detected from a voter, the particular votes can be reduced from final tally by decrypting the particular ciphertexts from bulletin board. That means, the voter who tried for vote manipulating lose the voter privacy.

Chapter 7

Comparison of Proposed Schemes with Helios

In this chapter, we compare the computational requirement of proposed schemes with mixnet based Helios scheme. The major change in proposed schemes from Helios is on proof of mixnet integrity part. So our comparison is mainly focused on proof of mixnet integrity part only. Let us consider that total number of voters is n and number of mix servers in a mixnet is m. Following section explain efficiency of each scheme in detail.

7.1 Efficiency of Proof of Mixnet Integrity in Helios

Proof of integrity procedure in Helios explained in section 4.2.2. Here each mix server have to generate t number of shadow mix outputs. Then verifier verify the integrity of each mix server. Figure 7.1 shows a mixnet with m mix servers, t shadow mix outputs and n number of voters. Assume that mix server take n computational steps to mix (re-encrypt and permute) a set of n ciphertexts. Then the total number of computational steps required here is $m * t * n$. As explained in section 4.2.2, to achieve $1/2^{80}$ security, the value of t should be 80. That mean number of operations required here is $80 * n * m$.

7.2 Efficiency of Proof of Mixnet Integrity in Tweak Based Scheme

Proof of mixnet integrity in tweak based scheme is straight forward. Section 6.1.1 explains the tweak based scheme proof of integrity in detail. In short, due to the secret tweak value, adversarial mix server cannot add a valid ciphertext while mixing (re-encryption and permutation). This makes integrity check easy. To verify the integrity of mixnet, decrypt the mixnet output and check whether is there have any invalid votes present or not. All votes are valid means, all mix servers in the mixnet is honest. Invalid votes are present means, malfunction happened

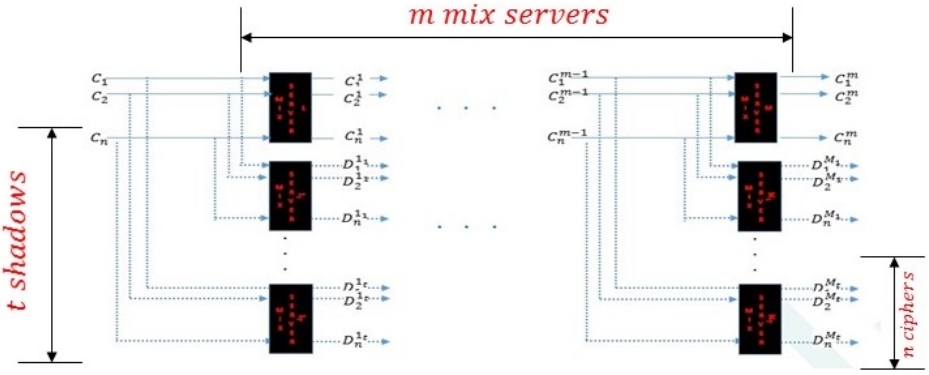


Figure 7.1: Proof of mixnet integrity in Helios: A Mixnet with m mix servers, t shadows and n ciphertexts.

while mixing. In short, any one from public who want to verify the integrity can do it by checking the decrypted mixnet output. Compared to Helios scheme, the additional mechanism required here is mix&tweak network. The only difference of mix&tweak network from mixnet is, the additional tweak operation in each mix&tweak server. So that we can use same mixnet infrastructure for mix&tweak server also, by adding a tweak operation in each mix server.

7.3 Efficiency of Proof of Mixnet Integrity in Dummy Vote Based Scheme

Proof of mixnet integrity in dummy vote scheme is also straight forward. Section 6.2.1 explain the dummy vote based scheme proof of integrity in detail. In short, mixnet input contains both dummy vote ciphertext and genuine vote ciphertext. An adversary cannot distinguish between dummy vote ciphertext and genuine vote ciphertext. So the only choice left for an adversary is, choose a ciphertext randomly and replace it with his own genuine ciphertext. If the chosen ciphertext is of a dummy vote, the vote manipulation can be detected as explained in section 6.2.1. Here only thing verifier have to do is, decrypt the bulletin board with $SK2_s$ and compare the v and x as shown in figure 6.4. If $v = x$, then verifier can conclude that all mix servers in mixnet is honest, with a high probability. Otherwise, verifier can conclude that mixnet integrity is broken.

Compared to Helios scheme, extra operation required in dummy vote based scheme is, each voter have to do five votes (one genuine and remaining dummy). We can remove this overhead from voter by implementing ballot creating web client module such a way that, for voters who don't want to put effort for dummy vote casting, the web client module will auto generate and cast the dummy votes.

Chapter 8

Conclusion and Future Work

The major advantage of end to end verifiable voting schemes over traditional voting schemes is, anybody can verify the integrity of election result. The Helios mixnet based scheme is the one of the latest major such implemented scheme. In this work, we studied Helios mixnet based implementation and proposed two new schemes, both of them follows overall same structure of Helios. Our proposed schemes made proof of mixnet integrity straight forward for verifiers. The first proposal, tweak based scheme is specific to mixnets based on ElGamal encryption scheme. The second proposal, the dummy vote based approach is general and can be extended to other mixnet applications.

In last 30 years, researchers proposed many type of end to end verifiable voting schemes. Some of them are actually implemented in practice, Scantegrity II and Helios are the example. But the problem remaining is, compared to currently using paper ballot based voting scheme or electronic machine based voting schemes, all the currently available end to end verifiable voting schemes are hard to understand and difficult to use for common voters. Due to this, these schemes are not practical for large public elections. So building an end to end verifiable voting scheme that is simple to explain and also simple to use is remains as a open problem.

Specific to this thesis work, the implementation of proposed scheme is remains as future work to do.

Bibliography

[1] Adida, B. Helios: web-based open-audit voting. In *Proceedings of the 17th conference on security symposium* (2008).

[2] Alex Essex, Jeremy Clark, R. C. S. P. The punchscan voting system. In *In proceedings of first university voting system competetion* (2007).

[3] Atushi Fujioka, Tatsuaki Otamoto, K. O. A practical secret voting scheme for large scale elections. In *Advances in Auscrypt* (1992).

[4] Benaloh, J. A robust and verifiable cryptographically secure election scheme. In *Proceeding of the USENIX/Accurate Electronic Voting Technology Workshop* (Microsoft Research, 2006).

[5] Chaum, D. L. Untraceable electronic mail, return address, and digital pseudonyms.

[6] David Chaum, Aleksander Essex, R. C. J. C. S. P. A. T. S. P. V. Scantegrity: End-to-end voter verifiable optical-scan voting. In *IEEE Security and privacy* (2008).

[7] David Chaum, peter Y.A Ryan, S. A. S. A practical, voter-verifiable election scheme. In *Proceedings of the 10th European conference on research in computer security* (2005).

[8] David Chaum, T. P. P. Wallet databases with observers. In *Proceeding of the 12th annual international cryptology conference on advanced Cryptology* (1992).

[9] ElGamal, T. A public key cryptosystem and a signature scheme based on descrete logarithms. In *IEEE transactions on information theory.*

[10] Jivanyan, A. New receipt free e-voting schemend self-proving mix net as a new paradigm . IACR.

[11] Josh Benaloh, D. T. Receipt-free secret-ballot elections. In *Proceedings of 26th ACM symposium on theory of computing* (2001).

[12] Josh D. Cohen, M. J. A robust and verifiable cryptographically secure election scheme. In *Proceeding of the 26th annual symposium on foundation of computer science* (1985).

[13] Kazue Sako, J. K. Receipt free mix-type voting scheme. In *Advances in EUROCRYPT* (1995).

[14] Kazue Sako, J. K. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *EUROCRYPT* (1995).

[15] Kyle MacNamara, L. l. A survey of electronic voting schemes.

[16] Labeeb Ahmed Qubati, Sheriff Khattab, I. F. Survey on end-to-end verifiable cryptographic voting schemes. In *International journal of computer applications* (2014).

[17] Laure Fouard, Mathilde Ducluse, P. L. A survey of electronic voting schemes.

[18] Martin Hirt, K. S. Efficient receipt free voting based on homomorphic encryption. In *Advances in EUROCRYPT* (2000).

[19] Otamoto, T. An electronic voting scheme. In *Proceedings of IFIP* (1996).

[20] Rjaskova, Z. Electronic voting schemes. PhD Thesis, Comenius University, Bratislava.

[21] Ronald Cramer, Rosario Gennaro, B. S. A secure and optimally efficient multi-authority elections scheme. In *Advances in EUROCRYPT* (1997).

[22] Ronald L Rivest, W. D. S. Three voting protocols: Threeballot, vav, and twin. In *In USENIX/ACCURATE Electronic Voting Technology Workshop* (2007).

[23] Yiannis Tsiounis, M. Y. On the security of elgmalal based encryption.