# Zero-Knowledge proof, Deniability and Their Applications in Blockchain, E-Voting and Deniable Secret Handshake Protocols

A thesis submitted for the degree of Doctor of Philosophy in Computer Science

By

Somnath Panja
Applied Statistics Unit

Thesis Supervisor: Prof. Bimal Roy
Applied Statistics Unit


Indian Statistical Institute

203, B.T. Road, Kolkata-700108

February 2021

*Dedicated to my parents, my sisters and all the elders to me in the family.*

# Acknowledgements

It gives me immense pleasure to submit this thesis after years of meticulous research at Indian Statistical Institute, Kolkata. I take this opportunity to express my deep sense of gratitude to the persons associated with me without whom this task would not have been completed.

Firstly, I am grateful to my thesis supervisor Professor Bimal Roy for his esteemed guidance throughout my research life. His enduring inspiration, endless encouragement and handling the difficulties with patience have huge effect on my thesis and nourished my academic knowledge. His constant dedication to the field of cryptography and to teaching amaze me still every day. I owe an immense debt of gratitude for his top-notch advice along the way.

I would also like to express my sincere thanks to Professor Feng Hao, Department of Computer Science, University of Warwick, UK for his willingness to assist me whenever needed. He is also a collaborator of my research work. I am grateful to both the University of Warwick, UK, and Newcastle University, UK, for supporting me financially and providing me infrastructure facility during my multiple visits to the university for research collaboration hosted by Professor Feng Hao.

I also place on record my sincere thanks to Professor Kouichi Sakurai, Faculty of Information Science and Electrical Engineering, Kyushu University, Japan for his willingness to assist me during my research. He is a collaborator of my research work. I am grateful to Kyushu University for providing me financial support and infrastructure facility during my visit to the university hosted by Professor Kouichi Sakurai.

I am also deeply indebted to my beloved teachers Prof. Mridul Nandi, Prof. Kishan Chand Gupta, Prof. Debrup Chakraborty, Prof. Sushmita Ruj, Prof. Palash Sarkar, Prof. Subhamoy Maitra and all the faculty members of the Applied Statistics Unit and the Department of Computer Science, Indian Statistical Institute, Kolkata, for their immense help and support on different matters. I would like to thank the Dean and the Director of the Indian Statistical Institute. I am very much grateful to the Indian Statistical Institute, Kolkata, for supporting me financially through a fellowship and providing me infrastructure

facilities.

I would also like to thank Dr. Samiran Bag, University of Warwick, UK and Dr. Sabyasachi Dutta, University of Calgary, Canada, for their valuable suggestions and collaboration on my research. I would also like to thank Dr. Indranil Ghosh Ray, University of Warwick, UK for his help.

Finally, I would like to acknowledge the continuous support and encouragement of my parents, my sisters, all the elders to me in the family in completion of this research work. I thank all those who have helped me to complete this research work.

# Abstract

In this thesis, we propose a cryptographic technique for an authenticated, end-to-end verifiable and secret ballot election. Currently, almost all verifiable e-voting systems require trusted authorities to perform the tallying process except for the DRE-i and DRE-ip systems. We have shown a weaknesses in the DRE-ip system and proposed a solution. We have modified the DRE-ip system so that no adversary can create and post a valid ballot on the public bulletin board without detection. We provide security proofs to prove the security properties of the proposed scheme. We propose two methods to store these ballots using blockchain and cloud server. To the best of our knowledge, it is the first end-to-end verifiable Direct-Recording Electronic (DRE) based e-voting system using blockchain. We introduce an improved non-interactive zero-knowledge proof (NIZK) that boosts the efficiency of the system. We propose a method for publishing the final tally without revealing the tally from individual DRE machines using secure multi-party computation and NIZK proof. The experimental data obtained from our tests show the protocol's potential for real-world deployment. We also propose a secure and verifiable voter registration and authentication mechanism. The proposed system prevents ballot stuffing attack.

We also propose the first self-tallying decentralized e-voting protocol for a ranked-choice voting system based on Borda count. Our protocol does not need any trusted setup or tallying authority to compute the tally. The voters interact through a publicly accessible bulletin board for executing the protocol in a way that is publicly verifiable. Our main protocol consists of two rounds. In the first round, the voters publish their public keys, and in the second round they publish their randomized ballots. All voters provide Non-interactive Zero-Knowledge (NIZK) proofs to show that they have been following the protocol specification honestly without revealing their secret votes. At the end of the election, anyone including a third-party observer will be able to compute the tally without needing any tallying authority. We provide security proofs to show that our protocol guarantees the maximum privacy for each voter. We have implemented our protocol using Ethereum's blockchain as a public bulletin board to record voting operations as publicly verifiable transactions. The experimental data obtained from our tests show the protocol's potential for the real-world

deployment.

We then propose a deniable secret handshake protocol. The notion of deniability ensures that the transcript generated in an interactive protocol does not yield any evidence of the interaction. In the context of key-exchange protocols for secure message transmission, the notion of deniability is well-explored. On the other hand, a secret handshake protocol enables a group of authorized users to establish a shared secret key and authenticate each other. Recently, a framework for deniable secret handshake is proposed by Tian et al. in ISPEC 2018. We analyze the protocol, show three flaws and give solutions to prevent them.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Election is one of the most important process of a democratic government. Conducting a secure election is also one of the most challenging task as its requirements and constraints are remarkably strict. The idea of using cryptography in developing secure voting systems was first suggested by Chaum in 1981 in his highly influential paper [30] on untraceable electronic mail. Chaum described the possibility of conducting remote election using new cryptographic primitives. This idea proved to be popular in academic research community, and several voting systems were appeared in the literature over the next two decades, some of which resulted in actual prototype solutions [38, 63]. In the last two decades, research in developing secure electronic voting systems has received significant boost mainly for two reasons: First, the United States presidential election in 2000 caused much stir and cast a spotlight on United States voting infrastructures where voters faces much difficulties in casting their votes due to confusing ballot design and punch card voting machines [90]. This flurry of national attention led to allocation of a generous amount of funding for research and development of voting systems [36]. Secondly, several investigations on e-voting systems have raised the security issues related to both the vote secrecy and the integrity of the election [84, 61]. There have also been several documented instances of voting machine malfunctioning during live elections, altering candidate votes, adding and subtracting votes. This leads to extensive research on e-voting systems.

We now describe the security properties of e-voting systems.

## 1.1 Security Properties of Voting systems

In this section, we introduce some security properties of E2E verifiable voting systems. Many properties are related to each other whereas some other properties are in direct conflict. The merit of a system depends on what properties it satisfies.

### 1.1.1 Vote Privacy

The privacy of the vote is a fundamental human right. The reason behind it is that if an outsider gets to know the voter's choice, she may intimidate the voter or bribe the voter to vote in a certain way, ultimately corrupting the election. These fears led to the notion of secret ballot. Typically, voter privacy is maintained by providing the voter a private voting booth at the polling station. Her vote is cast in a ballot box and it does not bear any distinguishing marks. It is then mixed with all other ballots making it very difficult to determine her vote.

In early research literature, voting systems (for example, [35, 16, 11]) maintained a voter-ballot linkage by providing a receipt to the voter so that she can trace her vote on the public bulletin board. However, in 1994, Benaloh et al. [15] raised an issue and pointed out that this strategy enables a voter to prove how she voted to a third party, and thereby facilitating vote-buying and coercion. The authors introduced the notion of receipt-freeness. After that, due to this issue, several voting systems (such as [15, 103]) dispensed the receipt entirely and focused on integrity and transparency of the tallying process. However, in his highly influential paper in 2004, Chaum [31] introduced the use of receipt again in the e-voting system with an important distinction that the contents of the receipt are cryptographycally masked, and thereby it maintains privacy of the voter.

In 2005, Juels et al. [73] argued that a coercer may yet influence a voter's choice without any knowledge of a voter's choice. The authors mentioned three such way of coercion: a coercer may force the voter to abstain from voting, the coercer may seize her voter credentials, she may also force her to randomly vote for a candidate. Note that there is a difference between the receipt-freeness and coercion resistance: in receipt-freeness, it is assumed that the coercer is restricted to observing the election and the evidence given by a coerced voter.

However, in coercion resistance, the coercer is more powerful and may even talk to the voter in some way while she is voting.

Over the few years, the notion of vote privacy has been refined into following important properties:

- **Ballot Secrecy:** The voting system must keep the voter's choice of candidate secret.
- **Receipt-freeness:** The voting system should not provide any evidence to the voter so that she can prove to a third party how she voted.
- **Coercion-resistance:** The voter should be able to cast her vote for her choice of candidate even when she is appeared to be cooperating with the coercer.

Note that coercion-resistance implies receipt-freeness and receipt-freeness implies ballot-secrecy.

### 1.1.2 Vote verifiability

Generally, in real-world elections, the voting machines and the polling stuff need to be trusted for integrity of the election process. Over the last two decades, the academic research community attempt to use cryptography to reduce the trustworthiness on the voting machine and polling personnel for integrity of the election. Sako et al. [118] described the following two kinds of verifiability.

- **Individual verifiability:** A voter can verify that her vote has been included in the set of all cast ballots.
- **Universal verifiability:** Any observer can verify that all the cast votes has been correctly tallied.

The notion of E2E verifiable voting systems refers to the following three properties.

- **Cast-as-intended:** A voter can verify that her choice of candidate has been correctly marked on the ballot by the voting system.

- **Recorded-as-cast:** The voter can verify that her vote has been correctly recorded by the voting machine.
- **Tallied-as-recorded:** The voter can verify that her vote has been tallied as recorded.

These three properties can be ensured as follows: first, the voter verify that her vote is correctly encrypted by the voting machine. She can then verify that her ballot has been recorded on the bulletin board using her receipt to ensure her vote is recorded as cast. Then the voter or any observer can verify that the votes are tallied as recorded on the bulletin board. The voting system also provided proofs of its correct operations.

The above two kinds of verifiability are related to each other. The cast-as-intended and recorded-as-cast together ensure that they provide individual verifibility. The tallied-as-recorded property can be carried out by the voter or any observer, and hence it ensures universal verifiability.

### 1.1.3 Other important properties

We now describe some other important properties of a voting systems.

- **Eligibility verifiability:** Any observer of the election can verify that all the cast votes were cast by an eligible voter.
- **Usability:** The voter can cast her vote easily and effectively using the voting machine.
- **Accessibility:** The voting machine should be accessible to all voters including voters with disabilities, and still provides vote secrecy and verifiability.
- **Accountability:** If, at any stage of the election, the vote verification fails, the voter should be able to provide the proof or evidence of that failure to the relevant election authorities without compromising her ballot secrecy.
- **Robustness:** The voting machine should be robust i.e. it can provide correct result even when there are some small degree of malfunction or corruption without disrupting the election.

### 1.1.4 Conflict between different properties and challenges in developing voting system

There are some properties described above that are in direct conflict with each other. For example, vote privacy clashes with vote verifiability. If she can prove her candidate of choice to a third party, she can easily sell her vote and be coerced into voting for adversary's choice of candidate. The prevention of this conflict is described in [72]. The vote verifiability may also clash with usability since vote verifiability may require some extra steps for verification, which may be confusing to the voter [77] or make the voting system slightly complex. Similarly, accessibility may conflict with vote privacy. Providing human assistance or audio/visual aids to the voter with disabilities may disclose her vote to a third party. In case of internet voting, where voter casts her vote over the internet from her home, the voter is vulnerable to coercion and her ballot secrecy is not guaranteed.

Keeping such conflicts in mind, the researchers provide several technological remedies in their proposed voting systems. We now briefly describe some of the well-known voting systems proposed in the literature.

## 1.2 Some well-known voting systems

In this section, we briefly describe some of the well-known voting systems in the literature. These systems are categorised depending on cryptographic versus non-cryptographic ballots, and their ballot format, for physical ballots (i.e. paper based) versus electronic ballots, and their mode of deployment, for precinct-based voting (i.e. polling station based) as opposed to remote voting (i.e. internet voting).

### 1.2.1 Precinct-based voting with paper ballot

This category of voting systems consists of polling station based voting using paper ballots. The tally is computed from the paper ballots containing the voter's choice. These systems are descendent from Votegrity [31], and some of these systems are Scantegrity [33], Prêt à Voter [116].

#### 1.2.1.1 Votegrity

Votegrity [31] was the first E2E verifiable e-voting system. It was proposed by Chaum in 2004. It uses visual cryptography for the vote verification process. In Votegrity, an image is split into two shares using visual cryptography, such that individual shares do not yield any information about the original image and seems randomly generated. However, when two shares are super-imposed the original image is reconstructed. Vote processing is performed by a group of trusted authorities using decryption mixnet, invented by Chaum [30]. A system's security will be compromised only if all the trustees collude with each other.

On the election day, Alice goes to a polling station to cast her vote. She enters her choice of candidate on the voting machine. A printer prints pattern corresponding to her vote on two strips using visual cryptography. Her candidate choice is clearly visible when these strips are superimposed under a custom viewfinder. These two strips are generated using a pseudorandom function using a deterministic manner so that it appears random to an observer. The machine also prints some validating information and serial number on both the receipts. Alice chooses any one of the strips as her receipt and takes it home. The voting machine provides the chosen strips to the voter and shreds the other receipt. The machine saves a digital copy of the chosen receipt.

At the end of the voting phase, the machine publishes all the saved receipts on a public bulletin board. The voter can check that her receipt is correctly published on the bulletin board. If her chosen receipt does not match with those on the bulletin board, she can raise an issue with the election authority.

Although the receipts appear random, the printed information are digital encoding of data that enables the system to reconstruct both the receipts, and hence the vote itself. First the serial number is being detached from the saved receipt and then it is then passed through a set of decryption mixnet [30] servers and finally shows the vote without revealing any correspondence between the voter and her vote. The mixnet also publishes the proof of correct stuffing. Then any observer can compute the tally by observing these votes.

Votegrity is E2E verifiable and has proved to be immensely influential in literature. Several voting systems that followed Votegrity uses the same receipt and mixnet concepts,

such as Scantegrity [33], Prêt à Voter [116], which are deployed in real-world elections. Punchscan [49] and Scratch & vote [3] also takes inspiration from Votegrity and improve the system.

### 1.2.2 Precinct-based voting with electronic ballot

In this category of systems, voters cast their electronic votes in a polling station. These systems use various cryptographic primitives. Some highly influential systems in this category are MarkPledge [101] and the notion of Voter Initiated Auditing. Here we briefly discuss the notion Voter Initiated Auditing.

#### 1.2.2.1 Voter Initiated Auditing:

For most of the voting systems mentioned earlier, random checks are performed to check the security of the systems. There is no explicit guarantee that a voter's vote is correctly encrypted by the machine. For some systems such as Punchscan and Scantegrity that use paper ballots, the voter has no explicit assurance that the candidate randomization or marking options are encoded correctly. Instead, he gets that assurance from random audits conducted by multiple trustees. With these random verification, the voter can be assured that any significant changes will be detected. In 2006, Benaloh introduced the notion of Voter Initiated Auditing. In this system, the voter gets immediate assurance that her vote has been corrected cast.

In this paradigm, when a voter chooses her candidate on the voting machine, the machine encrypts her vote and provides a printed receipt consisting that encryption to the voter. The machine then gives the voter an option to either audit (i.e. challenge) or cast her vote. If she wishes to audit the vote (i.e. challenge the machine), the machine provides her another receipt containing her choice of candidate, the encrypted data and the randomness used to generate the encryption. With this information, the voter or anyone can regenerate the encryption to verify that the machine has correctly encrypted her vote. On the other hand, if she trust on the machine, she casts her voter. The machine then provides her a confirmation receipt.

The main difference here with other voting systems is that the machine first commits

to the encryption of the vote by printing it on a receipt before asking the voter whether to audit or cast her vote. If she wishes to audit, she can verify that the machine has correctly encrypted her vote. The voter can audit her vote as many times as she wishes before finally casting her vote. If the machine encrypts a vote for different candidate, the possibility that it will be detected increase with every audit performed by a voter. This technique ensures E2E verifiability.

Voter Initiated Auditing was not proposed as a new voting system, but as a vote casting procedure than can be augmented to existing voting systems to ensure E2E verifiability. Several well-known voting systems, such as Helios [1], VoteBox [119], Star-Vote [13] and DRE-ip [123], follow similar paradigm. Some other precinct based electronic voting systems are Bingo Voting [18], Wombat [9], DRE-i [59].

### 1.2.3 Remote voting over the Internet

Using this category of systems, a voter can cast her electronic vote over the Internet. Some of the prominent E2E verifiable voting machines in this category are JCJ/Civitas [73], Helios [1]. Some other well-known E2E verifiable remote voting systems are Adder [80], Pretty Good Democracy [117] and Remotegrity [147]. As discussed earlier in section 1.1.4, remote systems are vulnerable to coercion and the ballot secrecy is not guaranteed, and it restricts the use of remote voting systems in large scale elections.

### 1.2.4 Non-cryptographic verifiable voting systems

There are some voting systems that provides E2E verifiability without using cryptography. For instance, ThreeBallot [127], Twin and Aperio [47] voting systems ensures E2E verifiability in a polling station based paper ballot format. These systems have provided some new insights about the E2E voting systems.

## 1.3 Motivation of our work

This dissertation contributes to teaching, practice and the theory of cryptographic E2E verifiable e-voting systems using zero-knowledge proofs and blockchain. We also contribute

to the deniability of secret handshake protocols and discuss its usability in the e-voting systems. This thesis is based on the papers [109, 108, 106, 107].

### 1.3.1 Precinct-based Plurality Voting

In chapter 3, we propose a precinct-based E2E verifiable plurality voting system. This chapter is based on the papers [109, 108]. In a precinct-based voting system, each voter should receive assurance that her vote is *cast as intended*, *recorded as cast* and *tallied as recorded*. By contrast, in traditional paper-based voting system, a voter cannot verify how her vote is recorded and tallied in the voting process. As with traditional elections, voters go to their polling station, prove their eligibility for casting votes by presenting their identity card. The voter is given a token [84] that allows her to cast vote for her candidates of choice. Therefore, the system depends on trustworthy individual at the polling stations, thus leading to the introduction of automated paperless secure e-voting system. In this paper, we propose a secure authenticated DRE based E2E verifiable e-voting system without tallying authorities.

Hao et al. proposed a voting system, called DRE-i (DRE with integrity) [59], to achieve E2E verifiability without involving any tallying authorities (TAs). However, the pre-computation strategy requires that the pre-computed data is securely stored and accessed during the voting phase. This introduces the possibility for an adversary to break into the secure storage module and compromise the privacy of all ballots. To overcome this issue, Shahandashti et al. provided a voting system, called DRE-ip [123] (DRE-i with enhanced privacy). DRE-ip achieves E2E verifiability without TAs and simultaneously a significantly stronger privacy guarantee than DRE-i. However, both DRE-i and DRE-ip systems necessitate the requirement of a secure append-only public bulletin board (BB). If the BB or the voting machine or the private key of the signature is compromised, an attacker can change some ballots and add additional ballots as well in such a way that it cannot be detected by the DRE-ip tally verification algorithm. The private key of the signature might be compromised at the setup stage.

In his PhD thesis, Benaloh [11] assumes BBs with a secure append-only write operations, also stressing out that "implementing such bulletin boards may be problem unto itself".

Although the assumption that the BB is a trusted centralized entity is common in the literature, the importance of removing the BB as a single point of failure has been extensively discussed in the recent works of Culnane and Schneider [40], Chondros et al. [34] and Kiayias et al. [81]. In [81], Kiayias et al. show a weakness of the bulletin board proposed in [40] and improve the system. However, in [81], for $n$ peers, the minimum number of honest item collection peers that receive and store submitted items must be greater than $2n/3$ to ensure correct behavior of their bulletin board design. In [86], Küsters et al. raise awareness of an attack, which they call a clash attack, on the verifiability of some of the well-known e-voting systems (for example, ThreeBallot and VAV voting systems [127], a variant of the Helios voting system [1] and the Wombat Voting system [9]). Küsters et al. show that, if the voting machine and the bulletin board collaborate, a bulletin board can replace some ballots by its choice so that it cannot be detected. In [14], Benaloh et al. describe their Trash attack on some well-known verifiable e-voting systems if the bulletin board is compromised.

Instead of assuming a secure append-only public bulletin board, we have modified the DRE-ip algorithm to make it tamper-evident and proposed two methods (depending on how the election is arranged) to store the ballots. We have measured the costs in terms of Ethereum Gas (and US dollars) to verify and store each ballot on Ethereum blockchain [142]. In this case, all the ballots and public keys of the system remain tamper-resistant. This system prevents coercion even when voters are willing to be influenced. For example, voters may collude with an adversary to vote in favour of adversary's choice of candidate and may intend to prove their choice of vote after the voting process. Our proposed system uses the exponential ElGamal cryptosystem to encrypt a vote. The system generates two distinct generators of the group whose logarithmic relationship is unknown. The system securely deletes the random variable and the vote ('confirmed' vote) for each voter. Consequently, the voter cannot prove her choice of candidate to the adversary. Thus, this kind of voter coercion can be avoided. In addition, since each DRE machine normally covers voting process for small regions, revealing the tally from each DRE machines discloses the voter's distribution in small regions. This is also a breach of voter's privacy to some extent. Disclosure of voter's distribution may impact the financial investments, development and social security of those small regions. We apply a secure multi-party computation method with non-interactive

zero-knowledge (NIZK) proof to compute the final tally correctly while keeping the tally from each DRE machine secret.

We also propose a novel method for voter registration and authentication in a verifiable manner using Fuzzy Vault algorithm [99]. As opposed to the traditional biometric based authentication systems, we do not store the biometric template (fingerprint) of individual voters. There are some privacy and security advantage of our proposed voter registration and authentication system. First, we store a biometrically encrypted key corresponding to an individual voter from which neither the biometric nor the key can be retrieved. The secret key itself is independent of the biometric and can be changed or modified. Secondly, we do not rely on fingerprint alone for authentication since people leave fingerprint everywhere inadvertently. In addition, there may be Mafia-owned businesses that collect fingerprint data in large quantities if there is any exploit path. Thirdly, we present a two factor authentication scheme relying on fingerprint of the voter and a smart card (containing a secret key) given to the voter. Fourthly, our scheme is publicly verifiable. We show that the correctness of our system is verifiable by the public. As a result, the system thwarts ballot stuffing attack.

### 1.3.2 Internet-based Borda Count Voting

In chapter 4, we propose an internet-based verifiable Borda Count voting system. This chapter is based on the paper [106]. In a Borda count voting system, the voters cast their vote by ranking the candidates according to the order of preference. Each candidate obtains some points according to her position in the ranking done by a particular voter. In the end all the points obtained by her from all the voters are summed up and on the basis of this sum the winner is selected. For example, the least preferred candidate may get 0 point, the next one may get 1 point and so on. The Borda count voting system has been employed in the elections in Nauru [114], Slovenia [51] and in Kiribati [114]. In Ireland, a modified version of the Borda count system has been used by the Green party to elect its president [46]. Unlike the plurality voting system, the Borda count systems are designed to gather more information from the voter regarding her predilection toward more than one candidate.

Borda count voting using traditional paper ballot is not only time consuming, but also prone to human errors. Hence, there is a desirable need to advance toward using an electronic

voting system. However, with the advent of e-voting systems comes the need to ensure privacy and integrity of the voting system. An electronic system can be vulnerable to several attacks like intrusion, software alteration/modification, eavesdropping etc.

The lack of assurance on the integrity of e-voting systems has encouraged researchers to devise e-voting systems that provide end-to-end verifiability and that are proven to be secure. The research on end-to-end verifiable e-voting systems started with the pioneering work by Chaum [31]. Chaum's scheme uses visual cryptography to protect the privacy of voters. Every voter is issued with two strips of paper corresponding to a vote. Each one of the two strips does not divulge any secret on their own. When the two strips are superposed on one another under a custom viewfinder, the vote is revealed. The voter retains one strip and the other one is digitized before getting destroyed. Once the polling station voting has finished, all the saved voter receipts are published on the bulletin board, so that the voters can verify that their ballots are not discarded. This work first highlighted the notion of end-to-end verifiability in voting. A voting system is called end-to-end verifiable if it ensures that 1) every vote is cast as intended, 2) every vote is recorded as cast, and 3) every vote is tallied as recorded. Some other notable research works in this area are MarkPledge [101], Prêt à Voter [116], Punchscan [49], Scantegrity [29], Scantegrity II [32], Scratch & Vote [3], STAR-Vote [13], Adder [80], and Helios [1]. These systems use either mix-net [31] or homomorphic encryption [2], but they all involve a set of trustworthy tallying authorities (TAs) to perform the decryption and tallying process in a publicly verifiable way.

A major difficulty of implementing the above schemes is to find and manage a set of trustees who perform complex cryptographic operations as tallying authorities. Threshold control schemes can be applied to distribute the trust among TAs. Nonetheless, if a sufficient number of trustees collude, they can trivially breach the privacy of the e-voting system.

Hao, Ryan and Zieliński proposed a decentralized online voting scheme in [60]. This scheme allows a finite number of voters to conduct voting without requiring the help of any tallying authorities. This scheme is called Open-Vote network (OV-net). OV-Net consists of two rounds. In the first round each voter publishes her public key on the public bulletin board. In the second round each voter publishes her randomized ballot that is generated using the public keys of other voters, the secret key of that voter and her secret vote. Once

all the encrypted ballots are available on the bulletin board, anyone can easily calculate the tally from them. This scheme relies on non-interactive zero-knowledge proofs for proving the well-formedness of each ciphertext. The scheme offers the maximum possible privacy guarantee as each voter learns nothing more than the tally and their own vote. A public observer learns nothing more than the tally from the bulletin board.

The OV-Net scheme is designed to support plurality voting where every voter gets to vote for a single candidate. In this paper, we propose a decentralized Borda count e-voting scheme by extending OV-Net to support ranking-based voting. Similar to the OV-Net scheme, our Borda count scheme also has two rounds. In the first round, the voters publish their public keys, and in second round they publish their encrypted ballots under a public key re-constructed by combining every other voter's public keys. Each encrypted ballot comes with a Non-interactive Zero-Knowledge (NIZK) proof to prove the well-formedness of the ballot. Once all the voters submit their ballots, the tally can be computed from the published information available on the bulletin board. Anyone can compute the tally and verify the correctness of all operations with the help of the NIZK proofs. Our scheme offers strong privacy guarantee. A probabilistic polynomial time adversary learns nothing other than the tally and whatever she can interpret from the tally. We have implemented the Borda count e-voting scheme on Ethereum's [142] platform and have evaluated the efficiency of our scheme.

### 1.3.3 Deniable secret handshake protocol

In chapter 5, we propose a deniable secret handshake protocol. This chapter is based on the paper [107]. Privacy concerns in modern day electronic communications triggers the need of examining familiar security services such as *authentication* and *key agreement*. The paramount importance of protecting user's security and privacy, especially when two users want to communicate between themselves in a hostile network condition, initiated the study of *secret handshake* protocols. Consider the situation when two parties $A$ and $B$ want to identify each other as member of a secret agency and then communicate. However, the situation is really hostile and they do not know each other. Consequently, $A$ wants to make sure that if $B$ is not a member of the group then he will learn nothing about $A's$ identity

once the protocol is run. If only $B$ is a member of the group then he can identify $A$ as a member of the group, otherwise not. In a secret handshake protocol there is a central authority (CA) who creates a group of authorized users and is responsible for generating keys, certificates etc. for the users. Due to the deniability property, this deniable secret handshake protocol could be used as a coercion resistant internet-based voting system. We mention that this could be a potential future work.

# Chapter 2

# Background

Protocols for secure E2E verifiable e-voting systems rely on various cryptographic primitives or building blocks. In this chapter, we review those cryptographic primitives. We first review zero-knowledge proofs, a component used for E2E verifiable e-voting. We then discuss the mixnet protocol, another component of universal verifiable voting. We also discuss secure-multi party computation protocol. Then we briefly describe blockchain. Finally, we discuss biometric encryption protocols that will be required for registration and authentication of a voter using her biometric.

Over the years, researchers in the field of e-voting literature use various public key cryptosystems to make their system secure and verifiable. A comprehensive review of various public key cryptosystems and their security properties could be found in the book by Stinson [130]. We use ElGamal encryption and its variation in our proposed e-voting systems.

## 2.1 Zero-knowledge proof

In many verifiable e-voting systems, zero-knowledge proof has been used as a component to perform the verification process. Using a zero-knowledge proof paradigm, a prover $P$ communicates with a verifier $V$ to prove that an assertion is true without revealing its secret. If the prover possesses the secret information and the assertion is valid, then the verifier should accept this proof. However, if the assertion is invalid and the prover does not posses the secret information i.e. the prover is dishonest, then the verifier should reject this proof with noticeably high probability. In the end, after verifying the proof, the verifier should not learn anything more than the truth of the assertion. In other words, no matter

what knowledge the verifier gains after executing the protocol, it could have got the same knowledge without interacting with the prover. Thus, although the verifier can verify the validity of the proof while interacting with the prover, it can not gain any new information and, in particular, he can not turn around and regenerate the proof on his own.

Using Zero-knowledge proof protocol, a prover proves an assertion of the form '$x$ is in language $\mathcal{L}$' to a verifier, where $x$ is a string and $\mathcal{L}$ is a language, generally an NP language. The inputs to the prover are $x$ and an witness $w$ for $x$ such that $\mathcal{R}_{\mathcal{L}}(x, w) = 1$. The verifier is only given $x$ as input. The protocol is zero-knowledge if, after execution of the protocol, the verifier knows nothing about the witness $w$, except what it can deduce from the input $x$.

The definition of a perfect zero-knowledge proof is given below.

Perfect zero-knowledge proof: An interactive proof system $< P, V >$ for language $\mathcal{L}$ is said to be a perfect zero-knowledge proof if there exists a negligible function $\eta(.)$ such that the following three properties hold:

- Completeness: $\forall x \in \mathcal{L}, Pr[OUTCOME_V(< P(x, w) >, V(x)) = 1] > 1 - \eta(k)$, where $OUTCOME_V(< P(x, w) >, V(x)) = 1$ means that output of the verifier $V$ is success (i.e. the assertion is true) after execution of the protocol between the prover $P$ and the verifier $V$. The prover $P$ takes $x$ and the corresponding witness $w$ as input. The verifier $V$ only gets $x$ as input.

- Soundness: For any prover $P^*$ (other than $P$) that does not know the witness $w$, $\forall x \notin \mathcal{L}, Pr[OUTCOME_V(< P^*(x) >, V(x)) = 1] < \eta(k)$

- Zero-knowledgeness: There exists a probabilistic polynomial time simulator $S$ such that for all verifier $V^*, \forall x \in \mathcal{L}, S(x) = VIEW_{V^*} < P(x, w), V^*(x) >$, where $VIEW_{V^*} < P(x, w), V^*(x) >$ is a record of the interactions between the prover $P$ and the verifier $V^*$.

### 2.1.1 Variants of zero-knowledge:

There are some variants of zero-knowledge exist in the literature.

- Statistical zero-knowledge: A protocol is called statistical zero-knowledge if the distributions produced by the simulator $S$ and the proof protocol $< P(x, w), V^* >$ are statistically close i.e. the statistical difference between the two distributions is a negligible function.

- Computational zero-knowledge: A protocol is called computational zero-knowledge if no efficient algorithm can distinguish between the distributions produced by the simulator $S$ and the proof protocol $< P(x, w), V^* >$. In this paradigm, the verifier $V$ and $V^*$ are assumed to be probabilistic polynomial time. However, a surprisingly powerful verifier $V^*$ or $V$ might be able to extract some more knowledge (about the witness $w$) from the execution of a computational zero-knowledge protocol than the simulator $S$.

- Zero-knowledge argument: The prover $P$ is assumed to be probabilistic polynomial time on the security parameter $k$. However, the prover may spend significant amount of time preparing the execution of the protocol.

- Honest-verifier zero-knowledge: The verifier $V$ is expected to be honest (i.e. TO follow the protocol correctly) during execution of the protocol. In particular, according to the zero-knowledge proof protocol, the verifier is expected to send a random challenge to the prover. An honest verifier will always choose a random challenge and will not choose it depending on the prover's messages. In honest-verifier zero-knowledge proof protocol, the simulator simulates the transcript of the interactions between the prover and the verifier rather than simulating anything that the verifier could output. An interactive honest-verifier zero-knowledge proof can be made non-interactive by using Fiat-Shamir heuristics. In Fiat-Shamir heuristics, the verifier generates the random challenge by using the hash of the previous messages generated during the execution of the protocol. This hash function is modeled as a random oracle.

Generally in a verifiable voting system, zero-knowledge proofs are used to prove that the voting system follows its actions correctly in a verifiable manner without revealing the secret vote and the randomness used to generate the encryption.

## 2.2 Blockchain

Since the invention of Bitcoin, research on bitcoin [98] and blockchain has become a thriving field. One other blockchain that has become highly influential in research community is Ethereum [142]. In this section, we briefly describe the Blockchain. Blockchain uses its peer-to-peer network to accept and store the transactions in a decentralized fashion. The network stores the transactions by hashing the into the ongoing blockchain. While adding a transactions into the blockchain, it uses a hash-based proof-of-work mechanism to form a record that can not be changes without redoing the proof-of-work. The longest chain in the blockchain network serves as the proof of sequence of events witnessed. As long as majority of the CPU power in the network is controlled by the honest nodes, they will generate the longest chain outpacing the dishonest nodes. In a blockchain's network, messages are broadcast, nodes can leave or rejoin the network at their will, accepting the longest chain as a proof of what happened while they were gone.

- **Transaction:** Each transaction bears the digital signature of its owner. Transactions are broadcast and all the nodes agree to accept the longest chain of blocks as the current record of the blockchain. Transactions are stored in the order in which they were received into the blockchain. The structure of a transaction is defined by the corresponding blockchain.

- **Blocks and hashing:** A block may contain several transactions arranged in a Merkle tree fashion. The transactions are hashed into a Merkle tree and only the root of the tree is included in the block's hash. A cryptographic collision resistant hash function is used to take hash of the block and published widely. Each hash includes the previous hash in its hash to form a chain with each additional hash reinforcing the previous one, and hence any modification in the chain will be detected by checking the hash values. Each node checks the hashes of all blocks in the blockchain and accept the longest chain whose hashes match correctly. The network avoids double inclusion of the same transaction or block that has already been added earlier.

- **Proof-of-work:** A proof-of-work is a computationally expensive task that will be

performed by a miner to include a transaction or a block into the blockchain. As the blocks are added one after the another into the blockchain after performing proof-of-work and the previous hashes are included into the current block, to modify a prior block in the blockchain, an attacker has to redo the proof-of-work computation for that block as well as all the blocks after it in the blockchain so that it can readjust all the hashes in the blockchain from that block. The exact proof-of-work algorithm depends on the corresponding blockchain. In Bitcoin, the proof-of-work system is similar to Adam Back's hashcash [4]. The proof-of-work involves searching for a value that when hashed begins with a predefined number of zero bits. Therefore, to perform this task, the average work to be done is exponential with the number of predefined zero bits; however, it can be verified performing only a single hash computation. In Bitcoin, this proof-of-work is performed by incrementing a nonce in the block until a value is found such that the block's hash begins with the required number of zero bits. Once a block is appended into the blockchain after performing this proof-of-work, it cannot be changed without redoing the proof-of-work. As new blocks are appended one after the another into the blockchain, modifying a block requires to perform the proof-of-work task again for that block as well as all the blocks appended after it in the blockchain.

- **Network:** The network is similar to Bitcoin's peer-to-peer network [98].

## 2.3 Mixnet

Consider a set of messages sent by various senders who wish to shuffle the messages without reveling the secret permutation. This functionality was first introduced by Chaum [30] in 1981 and called the protocol as mixnets. Since then, several mixnet protocols are proposed in the literature based on different definitions and constructions. These mixnets can be classified into two types: heuristics-based mixnets and robust mixnets.

In heuristics-based mixnets, the shuffling is usually done for low latency applications and executed almost synchronously, for instance, web browsing. This kind of mixnet protocols maintain some level of privacy; however, some mixnet servers may be dropped or corrupt messages, although the impact of this may be not serious. In this case, a different set of

mixnet server can be used to perform proper mixing.

In robust mixnets, the requirements are very strict such as inputs messages can not be modified or dropped. These mixnets can be used in applications like voting. The shuffling may take significant time like hours or even a day since the mixing is done in large batches. The privacy of the shuffled permutation should also be preserved and should be provably secure and protected.

Over the years, various mixnet protocols have been proposed. Some interesting attacks have been discovered and fascinating technique proposed to improve efficiency. Figure 2.1 depicts a mixnet diagram.

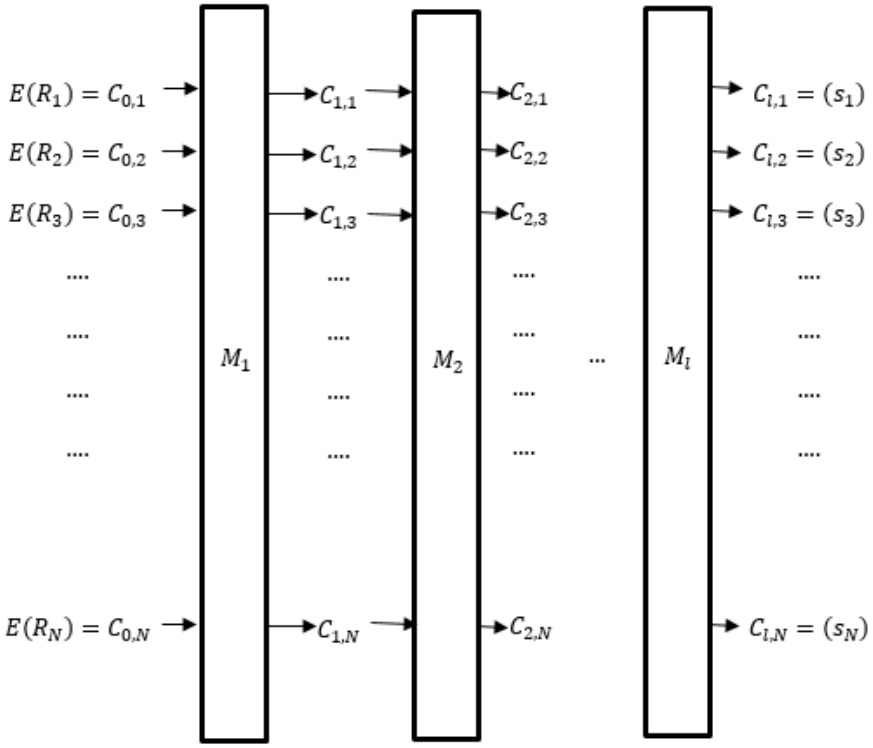


Figure 2.1: An example of mixnet with $N$ messages and $l$ mixnet server. $R_i$ represents the $i$-th message. The encryption function is denoted by $E$. $M_k$ are mixnet servers $\forall k \in \{1, 2, ..., l\}$. $C_{k-1,j}, \forall j \in \{1, 2, ..., N\}$ are the inputs to the mixnet $M_k$, and its outputs are $C_{k,j}, \forall j \in \{1, 2, ..., N\}$.The last mixnet server's output $(s_1, s_2, ..., s_N)$ is a permutation of $(R_1, R_2, ..., R_N)$.

### 2.3.1 Categorization of mixnets based on security provided

For mixnets that provide proof of correctness, two important security properties needs to be considered: privacy and soundness of the protocol. The mixnets provide interactive proofs that it performs the permutation correctly.

**Privacy:** In a mixnet, it is assumed that the adversary is computationally bounded. Since the inputs to the mixnets and its outputs are ciphertexts, a computationally unbounded adversary can decrypt the messages and get the corresponding permutation. Several mixnets are proposed in the literature with different levels of privacy.

- In some mixnets, all input to output permutations are possible and the permutations remains secret to a computationally bounded adversary. If, from some other sources, some correspondence of a permutation are revealed, it does not leak any additional information about the rest of the correspondence of the permutation. Clearly, these types of mixnets are more interesting than other types of mixnets.

- In some mixnets, although any input can go to any output, some permutations are not possible. If, from some other sources, some correspondence of a permutation are revealed, it may leak some additional information about the rest of the correspondence of the permutation.

- Some mixnets provides the proofs of their permutation in such a way that the proof itself narrows down the possible permutation.

**Soundness:** The soundness property proves that under what condition a mixnet server can cheat the proof protocol. Different mixnet protocol provides different levels of soundness. There are mainly three different types of mixnets based on the soundness property.

- Some mixnet protocols provide overwhelming soundness property i.e. even a computationally unbounded adversary has a negligible chance to cheat the proof protocol. Clearly, this kind of mixnet protocols are more interesting.

- In some mixnet protocols, a computationally bounded adversary has a negligible chance to cheat the proof protocol. This kind of mixnet protocols are used in many

practical purposes.

- In some mixnet protocols, a prover has a small but not negligible chance to cheat the proof protocol and provide a seemingly correct suffle.

### 2.3.2 The public bulletin board

Most verifiable robust mixnet protocols use a public bulletin board (BB) so that the public can check the proof protocol to verify its correctness. All the entries of the mixnet servers are posted on this public bulletin board. There are several algorithms for implementation of a public BB proposed in the literature. For instance, In [81], Kiayias et al. propose a public bulletin board. However, in [81], for $n$ peers, the minimum number of honest nodes must be greater than $2n/3$ to ensure correct behavior of their bulletin board protocol.

### 2.3.3 Early Mixnets:

Chaum first introduced the mixnet protocol in 1981 in his highly influential paper on untraceable electronic mail [30]. Chaum used RSA encryption (i.e. RSA onions) with random padding. Let us assume that there are $l$ number of mixnet servers. Let us denote the $i$-th mixnet server as $M_i$, its public key as $pk_i$ and the corresponding secret key as $sk_i$. Let the random number used for the $j$-th message $m_j$ and the $k$-th mixnet server be $r_{k,j}$. All the public keys of the mixnet servers are announced publicly. For the $j$-th message $m_j$, the input to the first mixnet server will be of the form $C_{0,j} = E(r_{1,j}, E(r_{2,j}, E(r_{3,j}, ...E(r_{l,j}, m_j)....)))$. After receiving the input from the $(i-1)$-th mixnet server, the $i$-th mixnet server decrypts outer layer of the input ciphertext (onion) using her secret key $sk_i$ and removes the random padding $r_{i,j}$. It then outputs the reduced onions in lexicographic order.

**Attacks on Chaum's mixnets:** Pfitzmann and Pfitzmann analyzed this Chaum's mixnet and showed a potential attack on it. They use the multiplicative homomorphic property of the RSA and the randomness of the padding [112] to attack the mixnet protocol. They also provide possible countermeasure to prevent that attack.

In 1993, Park et al. [110] proposed the first reencryption mixnet protocol. In reencryption mixnets, each mixnet server rerandomizes the ciphertexts with fresh random values that get combined with the existing random values. The authors used ElGamal cryptosystem in

their proposed mixnet protocol.

### 2.3.4 Universally verifiable cryptosystems

In 1990, Sako and Kilian [118] proposed a new kind of mixnet protocol with a new property, called universally verifiable mixnet. The proof of correct shuffling provided by the mixnets can be verifiable by the public or any observer of the protocol. Since the generation of the proof for the correct shuffling takes a significant amount of time, the efficiency of this mixnet protocol decreased. Generally, these kind of universally verifiable mixnet protocols publishes all its inputs, outputs and the proof of correct shuffling on a public BB so that anyone can verify the correctness shuffling. Neff [100] proposed a universally verifiable mixnet protocol that requires $8N$ exponentiations (not counting bulk modexp optimisations).

To preserve the privacy and the integrity of the system, it is assumed that not all mixnet servers cooperate with the adversary. This means that, to preserve the privacy and the integrity, at least one mixnet server must be honest who does not reveal its secret key and the random permutation to the adversary.

## 2.4 Secure multi-party computation

In 1986, Yao [144] proposed garbled circuit representation to perform any secure multi-party computation. In this method, decomposition of the computation is performed bit-by-bit basis using several gates. Although Yao's protocol can compute any multi-party computation securely preserving the secrecy of their input, it is inefficient in practice. In 1987, Goldreich et al. [53] proposed the GMW protocol to perform secure multi-party computation with honest majority. Ben-Or et al. [10] proposed a secure multi-party computation protocol, called BWG protocol. BWG protocol defines how to compute addition and multiplication on secret shares and is often used with Shamir secret sharing schemes. A comprehensive study on secure multi-party computation can be found in [54].

## 2.5 Biometric Encryption

Biometric technologies add a new level of authentication to various applications; however, there are always risks and challenges related to privacy and security of the biometric. Some

of technical challenges in biometric authentication algorithms include accuracy, reliability and security of the biometric. The main privacy and security risk related to the biometric technologies is creating a digital artifact of the biometric from the stored biometric template data in such a way that I will match with the original biometric data, called masquerade attack. Some other security and privacy risks include spoofing attack, replay attack, over-writing YES/NO result attack, tempering attack, Trojan horse attack, substitution attack, misuse of the biometric image (data theft), unauthorised secondary uses of the biometric etc. These kinds of risks limits the uses of biometric technologies in practical applications. Biometric encryption technologies can enhance both the security and privacy of the biometric data.

In 1996, Tomko et al. [134] first introduced the the concept of biometric encryption. A comprehensive review of biometric encryption technologies can be found in papers [67, 137, 26, 27].

Biometric encryption is a technique that either binds a randomly generated key with the biometric or generates a key from the biometric. The resulting data is called biometrically encrypted key or biometrically encrypted data. This biometrically encrypted key is then stored. This key can be regenerated only when a correct fresh biomtric is presented. These algorithms are fuzzy in nature since the biometric image may slightly vary each time. The major challenges in developing a biometric encryption algorithm is that how to generate the same key on verification even if the biometric sample may be slightly different each time. After a successful verification, the key can be regenerated from the biometrically encrypted key, and then it can be used in other cryptographic algorithms or as a password. Biometric encryption can be done in two methods: key binding with biometric and key generation from biomtric. Some biometric encryption is designed to work on both the methods. For instance, the Fuzzy Vault [99] and Fuzzy commitment [76] works on both the key binding and key generation mode.

In key binding method, a randomly generated key is securely bound to the biometric. The resulting biometrically encrypted data is then stored from which neither the biometric nor the key can be derived. The random number is generated by algorithm on enrollment in such a way that neither the user nor anybody knows it. This random number is different

on every different enrollment process and is independent of the biometric. At the end of the enrollment, both the random number and the biomentric are securely deleted. On verification process, a fresh biometric is presented which when applied on a legitimate biometrically encrypted key (or biometrically encrypted data), the same random number which was used during enrollment will be regenerated. The biometric encryption algorithm is designed in such a way that it can work on an acceptable variety of the fresh biometric image during verification process. At the end of the verification process, the biometric template is discarded again. The generated key can then be used in other cryptographic algorithms, or it can be used as a password.

In key generation method, a key is generated from the biometric during enrollment. This key is then stored. At the end of the enrollment, the biometric template is securely deleted. On verification, when a fresh biometric is presented, the algorithm will generate the same enrolled biometric template from the stored key.

Most of the biometric encryption algorithms use a suitable Error Correcting Code (ECC). Error Correcting Codes are used in applications where error can occur such as communication and data storage. For example, a binary block ECC, denoted by $(n, k, d)$, encodes $k$ bit data in $n$ bit data (where $n > k$) by adding some redundancy in it. $k$ is the length of the key. These $n$ bit strings are called codewords. The distance (generally Hamming distance) between any two codewords must be greater than or equal to $k$. This ECC can correct up to $(d-1)/2$ bits among $n$ bits. On verification, if the fresh biometric is taken from a legitimate user, the number of error bits will be within the ECC bound and the original codeword will be generated. For an imposter, it is highly likely that the number of error bits will be out of ECC bound and the original key will not be recovered.

Some biometric system applies a non-invertible transformation in the middle to improve their security of their system. This transformation is kept secret.

Some advantages of the biometric encryption based systems are:

- Unlike conventional biometric systems, in biometric encryption systems, the biometric images need not be stored, and it can not be retrieved from the saved biometrically encrypted data.

- For different applications, the biometrically encrypted data can be different for the same biometric. The biometrically encrypted data can be revoked or modified.
- Different biometric encrypted data for the same biometric can not be linked.
- After successful verification, the generated key can be used in cryptographic algorithms or as a password.
- These systems provide stronger security than the conventional biometric systems by binding the key with the biometric.
- These systems thwarts Trojon horse attack, overriding YES/NO result attack, substitution attack and lessen the possibility of masquerade attack.

These advantages of the biometric encryption systems makes it attractive in biometric based authentication systems as well as cryptographically secure authentication applications. Some of the well known biometric encryption systems proposed in the literature are Mytec1 [134], Mytec2 [129], Fuzzy Vault [99], Fuzzy commitment [76] etc.

## 2.6 Related work

We now focus on the related works done in literature.

### 2.6.1 Precinct-based Plurality Voting

There has been extensive research on e-voting system over the past two decades. Researchers have proposed a number of E2E verifiable schemes and some of these are used in practice. Notable E2E e-voting system include Votegrity [31] (proposed by Chaum), Markpledge [101], Prêt à Voter [116], STAR-Vote [13], Punchscan [49], Scratch & vote [3], Scantegrity , Scantegrity II [33], Helios [1], Bingo Voting [18], Wombat [9], DRE-i [59], DRE-ip [123]. A review of these systems can be found in [61]. Many other schemes follow similar approaches, in particular, a variant of Prêt à Voter, vVote, has been used in 2014 state election in Victoria, Australia [39]. Scantegrity [33] was trialled in local elections in Takoma Park, Maryland, USA [24]. Helios [1] was used to elect Universitè catholique de Louvain in 2009 and it has been used in universities and associations (IACR and ACM). Other schemes that have been

used in internal university or party elections include Punchscan [49], Bingo Voting [18], Wombat [9] and DRE-i [59]. However, almost all DRE based E2E verifiable systems require a secure bulletin board. Our system relaxes the requirement of secure BB and provides efficient solution using blockchain and cloud server. All of the above systems consider voter registration and authentication outside of their scope. In our proposed system, we have incorporated a secure voter registration and authentication mechanism using biometric in a verifiable manner.

There are few online internet voting systems based on blockchain. In [149], authors propose a voting system using Bitcoin [98]. In their voting system, the vote does not need to be encrypted and decrypted. Random numbers are used to hide the ballot that are distributed using zero-knowledge proof. There are few other cryptocurrency based voting systems (such as, [131]). These systems are based on the payments that she receives from the voter. The problem with these systems is that malicious voters may refuse to "pay" the candidate to retain the money. Furthermore, a centralized trusted authority who coordinates between the candidates and voters must exists. There are smart contract based internet voting systems [96], which only support two candidates("YES"/"NO" voting ) and voting is restricted to limited (approximately 50) participants. Tivi and Followmyvote ([50],[141]) are commercial internet voting systems that use the blockchain as ballot box. They claim to achieve verifiability and accessibility anytime and anywhere; however, the voter's privacy in these systems are hard to evaluate.

### 2.6.2 Internet-based Borda Count Voting

In most of the e-voting systems, trustworthy election authorities are required to preserve voter's privacy, to decrypt the vote and to compute the tally in a verifiable manner. Generally, threshold cryptography is used to distribute this trust among multiple tallying authorities; see, for example, Helios [1]. However, while using threshold cryptography, if the tallying authorities collude among themselves altogether, voter's privacy will be lost.

Several researchers have proposed e-voting systems based on blockchain. In [149], Zhao and Chan propose a voting system using Bitcoin. In their voting system, random numbers and zero-knowledge proofs are used to hide the vote. In [131], Tarasov and Tewari propose

an e-voting system based on cryptocurrency. In this system, a centralised trusted authority exists to coordinate the election. Tivi [141], Followmyvote [50] and The Blockchain Voting Machine [64] are Internet voting systems that use blockchain as a ballot box. These systems depend on trusted authorities to achieve voter's privacy. In Tivi, the trusted authority shuffles the encrypted votes before decrypting and computing the tally. In Followmyvote, the trusted authority obfuscates the link between a voter's identity and her voting key before the voter casts her vote. In our proposed protocol, the voter's privacy and the tally procedure do not depend on trusted election authorities. We implement the proposed protocol using smart contract in such a way that the Ethereum bockchain's consensus mechanism enforces the execution of the voting protocol. Recently, the Abu Dhabi Securities Exchange [65] has launched a blockchain-based voting service. In Estonia, blockchain-based voting systems [19] have been proposed for the internal elections of political parties and shareholder voting. The possibility of using blockchain in e-voting is also discussed in a report [19] by the Scientific Foresight Unit of the European Parliamentary Research Service. Recently, in [5], Bag et al. propose an end-to-end verifiable Borda count voting system. However, their scheme is on a centralised setting where a central facility (i.e., a touch-screen voting machine) is used to directly record votes from voters. In such a setting, it is inevitable that the touch-screen machine learns the voter's choice. In this paper, we propose the first self-tallying decentralized Borda count voting protocol, in which voters cast votes using their own devices in a distributed manner. No third-party entity can learn the voter's input unless all other voters are compromised (i.e., in a full-collusion attack).

The first self-tallying voting protocol was proposed by Kiayias and Yung [82] for board-room voting. Their protocol has the following three attractive features: it is self-tallying; it provides the maximum voter privacy; and it is dispute-free. We discuss these properties in detail in a later section. Their protocol executes in three rounds. However, the computational load for each voter is heavy and it increases linearly with the number of voters. A subsequent protocol by Groth [57] improves the computational complexity. The computational load for each voter is less than the Kiayias and Yung's protocol [82] and remains constant with the number of voters; however, the protocol trades off round efficiency for less computation. It requires $(n + 1)$ rounds, where $n$ is the number of voters. The round

efficiency is worse than the Kiayias and Yung's protocol [82]. Hao, Ryan and Zieliński investigated the computation complexity and proposed the Open Vote Network (OV-Net) protocol [60]. Their protocol significantly improves computational complexity. Their protocol executes in only two rounds. In fact, their protocol is a generalization of the anonymous veto network (AV-net) protocol [62] with the added self-tallying function. McCorry et al. [96] provided the first implementation of the OV-Net protocol using the Ethereum blockchain.

### 2.6.3 Deniable secret handshake protocol

The concept of secret handshake was first introduced by Balfanz et al. [6]. The security of the scheme was based on the hardness of *bilinear Diffie-Hellman* problem in *random oracle* model. The scheme was constructed from a pairing-based key agreement scheme. Castelluccia et al. [25] gave a more efficient scheme under the *computational* Diffie-Hellman assumption in the random oracle model. Tsudik-Xu [136] proposed a protocol where in a single run any number of members can authenticate themselves to other members in the group. Yamashita-Tanaka [143] gave a scheme with multiple groups that can accommodate change of memberships. That is, if a member is added to a new group or is deleted from one, it is not necessary to change his other memberships. This work was forwarded by Kawai et al. [78] to achieve a monotonicity condition. Apart from anonymity (which is inherent) of the participant, few other properties e.g. affiliation-hiding [68], unlinkability [70], and user untraceability [94] are also explored in the literature of secret handshake protocol. Tian et al. [133] proposed a solution to achieve full-unilinkability based on a $k$-size one-time certificate set. Untraceable secret handshake allows authorized users (participants) to remain untraceable if the issuing authority is not trusted. To cope against untrusted CA, Manulis et al. proposed two solutions [94, 95].

Deniability in the context of authentication was formally introduced by Dwork et al. [45]. Raimondo et al. [113] explored the notion of deniability for secure key-exchange protocols and put forward the notion of strong deniability and partial deniability. Jiang-Naini [71] gave an efficient key exchange protocol that achieves full deniability in the public random oracle model. Public random oracle model was defined by Pass [111], which is a weaker assumption than the random oracle. Unger-Goldberg [138] studied deniable authenticated key exchanges

in the generalized universal composability framework. They gave two protocols offering forward secrecy and both full and strong deniability. Later, they extended their study [139] to propose three strongly deniable key exchange protocols- deniable authenticated key exchange with zero-knowledge, zero-knowledge Diffie-Hellman and extended zero-knowledge Diffie-Hellman. Yao-Zhao [145] proposed a provably secure internet key exchange protocol which provides simultaneous strong deniability for both the participants. Their construction achieves deniability in the restricted random oracle model (defined in [146]). Recently, Tian et al. proposed a framework for deniable secret handshake protocol (DSH) [132]. They have given a generic construction of a DSH protocol from any forward-secure secret handshake protocol.

# Chapter 3

# A secure end-to-end verifiable e-voting system using blockchain and cloud server

## 3.1 Introduction

This chapter is based on papers [109, 108]. In this chapter, we propose a secure E2E verifiable e-voting system based on blockchain and cloud server. We have shown one weakness in the DRE-ip system [123]. An attacker can post ballots in such a way that it cannot be detected by the tally verification process. We have proposed a solution to prevent this attack. We provide security proofs to show that the proposed protocol is end-to-end verifiable as well as preserves each voter's privacy and the integrity of the system. We prove that the efficient NIZK proof algorithm proposed by Lin et al. [91] is not correct since it does not satisfy the properties of a zero-knowledge proof. We propose an efficient 1-out-of-n NIZK algorithm (i.e. the prover Algorithm 3 and the verifier Algorithm 4 described in section 3.5.4) involving conjunction and disjunction of multiple assertions. The security proofs of the proposed efficient NIZK proof algorithm are given in section 3.5.4.4. Depending on how the election is organised, we propose two methods to store the ballots on a public bulletin board. We have measured the costs of verifying and storing the ballots on the Ethereum blockchain using the proposed efficient 1-out-of-n NIZK proof. We have also measured the computation time of the proposed efficient 1-out-of-n NIZK proof. To the best of our knowledge, it is the first end-to-end verifiable DRE based e-voting system using blockchain. We also propose a method for publishing the final tally when multiple DRE machines are used in a regional

zone keeping the result (i.e. tally) from each DRE machine secret. By a regional zone, we mean a constituency from where a candidate is to be elected such as a district instead of the whole country. Thus, our system hides the voter's distribution in small areas where DRE machines are used. In addition, we propose a secure and verifiable voter registration and authentication mechanism using voter's biometric information (fingerprint). The proposed system prevents the well-known ballot stuffing attack.

**Components of the proposed system** 1) A fingerprint scanner with fingerprint pulse at the sensor, a smart card reader must be attached to a device that will verify the eligibility of the voter. It generates a token that will be presented to the DRE machine to proceed with the voting process. It also requires a public bulletin board to display the voter registration data. 2) The proposed voting system consists of all the devices required for the DRE-ip [123] system. Therefore, it requires a DRE machine with a printer attached to it and a public bulletin board to show the recorded ballots in public. The bulletin board can be a publicly accessible web site.

## 3.2 Preliminaries

In this section, we focus on the trust requirements and cryptographic assumptions based on which we prove the security properties of our proposed protocol.

### 3.2.1 Trust requirements

We describe our trust requirements that our scheme is expected to meet.

- **Integrity.** We assume that the voting machine or the BB may alter the voter's vote or change the tallying results. It may happen by accident (for example, due to some software bugs) or by malice (for example, due to some adversarial attack). However, we require that any such changes will be detected even when the machine is completely controlled by an adversary.

- **Vote secrecy.** When a voter selects her choice of candidate on the touch screen, the DRE machine learns her vote by definition. This is inevitable. We assume that the

DRE machine keeps the voter's choice secret. However, we require that when the DRE machine is completely compromised by an adversary, the adversary will only learn the partial tally at the time of compromise, but nothing beyond that.

### 3.2.2 Cryptographic assumption

We first describe some notations that we use throughout our paper.

**Notation.** We use the same notations that are used in the DRE-ip system. These notations were introduced by Camenisch and Stadler [23]. We use $P_K\{\lambda\colon \Gamma = \gamma^\lambda\}$ to denote a non-interactive *proof of knowledge* of a secret $\lambda$ such that $\Gamma = \gamma^\lambda$ for publicly known $\Gamma$ and $\gamma$. We shorten the notation to $P_K\{\lambda\}$ where context is clear. We use $P_{WF}\{A : X, ..., Y, Z\}$ to denote a *proof of well-formedness* of $A$ with respect to $X, ..., Y, Z$. We shorten the notation to $P_{WF}\{A\}$ where context is clear.

Zero-knowledge proof was first introduced by Goldwasser, Micali, and Rackoff [56] to prove the truth of a statement without conveying any other information. Subsequently, Bellare and Goldreich [7] refined the definition of zero-knowledge proofs to distinguish them from proofs of knowledge. We use Schnorr's proofs of knowledge of discrete logarithm [122]. We then apply the technique proposed by Cramer, Damgård Schoenmakers [37] to construct proof of disjunctive, conjunctive and combination of both. Fiat-Shamir heuristic [48] is applied to make the constructed proof non-interactive. The security proofs are in random oracle model [8]. The index $i$ of the transaction is embedded as input to the hash function to bind the proof to the transaction. In this paper, we propose an efficient NIZK proof. The proposed prover (Algorithm 3) and the verifier (Algorithm 4) algorithms are described in section 3.5.4.

**Cryptographic setup.** Our proposed system works over an ECDSA like group setting or a DSA like multiplicative cyclic group setting where the decisional Diffie-Hellman (DDH) assumption holds. In particular, we can choose two large primes $p$ and $q$ such that $q$ divides $(p-1)$. Then we choose the subgroup $\mathbb{G}_q$ of order $q$ of the group $\mathbb{Z}_p^*$ and assume that $g$ is the generator of $\mathbb{G}_q$. $q$ must be greater than the number of voters. The decisional Diffie-Hellman assumption [42] is given below. We use the DDH assumption to prove the security properties of our proposed protocol.

Assumption 1. (DDH) The two probability distribution $\{(g^a, g^b, g^{ab}) : a, b$ are uniformly and independently chosen from $\mathbb{Z}_q^*\}$ and $\{(g^a, g^b, g^c) : a, b, c$ are uniformly and independently chosen from $\mathbb{Z}_q^*\}$ are computationally indistinguishable in the security parameter $n = log(q)$.

## 3.3 A weakness of the DRE-ip system

In this section, we recall the DRE-ip [123] system, present one weakness of the system and provide a countermeasure. We describe the DRE-ip algorithm almost verbatim as given in [123].

### 3.3.1 The DRE-ip system

We describe the DRE-ip algorithm for the case when there are only two candidates contesting in the election. The DRE-ip algorithm does not require any tallying authority. Let $v_i$ represents the vote for the $i$-th ballot then we have $v_i \in \{0, 1\}$. During the setup phase, the algorithm chooses two generators $g_1, g_2$ of the corresponding group $\mathbb{G}_q$ such that their logarithmic relationship is unknown. The DRE keeps track of the partial sum $s = \Sigma r_i$ for the random numbers $r_i$ generated on the fly and the running tally $t = \Sigma v_i$ for the cast (confirmed) vote $v_i$. The system incorporates Benaloh-style voter-initiated auditing [12] in which the voter gets option to audit their vote generated by the DRE to get confidence that the DRE generates the ballot according to her choice of vote. An audited ballot cannot be used to cast a vote. At the end of the voting phase, the set of total ballot $\mathbb{B}$ will be comprised of the set of all audited ballot $\mathbb{A}$ and the set of all confirmed ballot $\mathbb{C}$ i.e. $\mathbb{B} = \mathbb{A} \cup \mathbb{C}$.

*Voting phase:* This phase involves the DRE, voter and the bulletin board:

1. The voter enters the booth, starts voting and keys in her vote $v_i \in \{0, 1\}$.

2. The DRE generates random $r_i \in \mathbb{Z}_q$ and computes $U_i = g_1^{r_i}, V_i = g_2^{r_i} g_2^{v_i}, P_{WF}\{V_i : g_1, g_2, U_i\}$.

The DRE provides a signed receipt including the unique ballot index $i$ and contents of the ballot $U_i, V_i, P_{WF}\{V_i\}$ to the voter.

3. The voter notices that the first part of the receipt is provided, then she chooses to either audit or confirm her vote.

In case of audit:

4. The DRE adds $i$ to the set of audited ballot $\mathbb{A}$ and provides a signed receipt to the voter. The receipt is clearly marked as *audited* including $r_i$ and $v_i$.

5. The voter takes and keeps the receipt. The voter verifies that $v_i$ reflects her choice. If the verification succeeds, the voting continues from step 1; otherwise if the verification fails, the voter should raise a dispute immediately.

In case of confirmation:

4. The DRE adds $i$ to the set of confirmed ballot $\mathbb{C}$. The DRE updates the tally and the sum: $t = \sum_{j \in \mathbb{C}} v_j$ and $s = \sum_{j \in \mathbb{C}} r_j$.

The DRE provides a signed receipt, clearly marked as *confirmed* to the voter. The DRE securely deletes $r_i$ and $v_i$.

5. The voter leaves the polling booth with her receipts.

6. The DRE posts all receipts provided to the voter on the bulletin board.

7. The voter matches her receipts with those on the bulletin board to verify that her receipts are posted on the bulletin board.

*Tallying phase:* This phase involves the DRE, the bulletin board and the public:

1. The DRE posts the final tally $t$ and the sum $s$ on the bulletin board.

2. The public:

- verify that all the well-formedness proofs on the bulletin board are correct (well-formedness verification);

- verify that, for all audited ballots on the bulletin board, $U_i$ and $V_i$ included in the first part of the receipt are consistent with $r_i, v_i$ provided in the second part (along with the system parameters $g_1, g_2$) (audit consistency verification);

- verify that following equations hold (tally verification):

$\prod_{j \in \mathbb{C}} U_j = g_1^s$ and $\prod_{j \in \mathbb{C}} V_j = g_2^s g_2^t$.

If at any point during voting phase or tallying phase, any of the verification by the voter or the public fails, the election stuff should be notified. These includes step 5 (for the audited votes), step 7 of the voting phase and step 2 of the tallying phase. The proof of well-formedness $P_{WF}\{V_i : g_1, g_2, U_i\}$ can be implemented as $P_{WF}\{V_i\} = P_K\{r_i : ((U_i = g_1^{r_i}) \wedge (V_i = g_2^{r_i})) \vee ((U_i = g_1^{r_i}) \wedge (V_i/g_2 = g_2^{r_i}))\}$. $P_{WF}\{V_i\}$ is a non-interactive zero-

knowledge proof of well-formedness of the ballot. The DRE-ip system records each audited and confirmed vote on the BB in a tabulated form given in Table 3.1.

| Initial: $g_1, g_2$ | |
|---|---|
| $i$: $U_i, V_i, P_{WF}\{V_i\}$ | audited, $r_i$, $v_i$ |
| ... | ... |
| $j$: $U_j, V_j, P_{WF}\{V_j\}$ | confirmed |
| Final: $t, s$ | |

Table 3.1: DRE-ip bulletin board. $V_j \in \{g_2^{r_j}, g_2^{r_j} g_2\}$, $s$ is the sum of all random variables of confirmed ballots and $t$ is the final tally.

The Theorem 1 in [123] analyzes the security of the DRE-ip algorithm. The Theorem 1 states that, in DRE-ip, assuming that all proofs of well-formedness are proofs of knowledge, if the public well-formedness and the tally verification succeed, then the reported tally is the correct tally of all confirmed votes on the bulletin board.

In [123], Shahandashti et al. also consider an intrusive adversary that apart from the ability to determine an arbitrary number of votes, gets read access to the DRE storage for a period during the voting phase. The authors consider that the adversary can control arbitrary number of voters, hence in effect she can cast an arbitrary number of votes. During the access period, the adversary is able to observe the votes cast and read the partial tally $t$ and partial sum $s$.

The Theorem 2 in the paper [123] analyzes the ballot secrecy in case of an intrusive attack. It states that, in DRE-ip, assuming that all proofs of well-formedness are zero-knowledge, if the DDH assumption holds, then an adversary that determines an arbitrary number of votes and gets temporary read access to the DRE storage cannot get any information about the non-adversarial votes cast before and after the adversarial access period other than their partial tallies.

### 3.3.2 Analysis of the protocol:

1) **Weakness.** If the voting machine and the election authority collaborate, the following attack is possible. The election authority who is responsible for publishing the total number of voters cast their votes in the election needs to increase the total count of cast votes. The

attacker can add some valid ballots in favour of her choice of candidate at the end of the bulletin board in such a way that it cannot be detected by the DRE-ip tally verification algorithm.

Suppose an attacker would like to add some ballots in favour of her candidate. The attacker can choose some random numbers $r_1, r_2, ..., r_k$ from $\mathbb{Z}_q$ such that $\sum_{i=1}^{k} r_i \equiv 0 \pmod{q}$. The attacker generates encrypted ballots of the form $((j+i\colon U_i, V_i, P_{WF}\{V_i\}), confirmed)$, where $U_i = g_1^{r_i}, V_i = g_2^{r_i} g_2^{v}, P_{WF}\{V_i\} = P_{WF}\{V_i : g_1, g_2, U_i\} = P_K\{r_i : ((U_i = g_1^{r_i}) \wedge (V_i = g_2^{r_i})) \vee ((U_i = g_1^{r_i}) \wedge (V_i/g_2 = g_2^{r_i}))\}$ and $v$ ($v \in \{0,1\}$) is the vote in favour her candidate. Since the attacker knows $r_i$ and $v$, she can generate the non-interactive zero-knowledge proof $P_{WF}\{V_i\}$.

| Initial: $g_1, g_2$ | |
|---|---|
| $j+1\colon U_1, V_1, P_{WF}\{V_1\}$ | confirmed |
| $j+2\colon U_2, V_2, P_{WF}\{V_2\}$ | confirmed |
| ... | ... |
| $j+k\colon U_k, V_k, P_{WF}\{V_k\}$ | confirmed |
| Final: $t + k.v, s$ | |

Table 3.2: Additional ballots added by an adversary

If an attacker posts encrypted ballots in favour of her candidate in the above form (Table 3.2) with zero-knowledge proof of well-formedness $P_{WF}\{V_i\}$ and changes the final tally from $t$ to $(t + k.v)$ as soon as the DRE publishes the final tally $t$ on the BB, then such attack cannot be detected by the tally verification procedure of the DRE-ip system. The attacker does not need to change the sum $s$. This is because $g_1^{\sum_{i=1}^{k} r_i} = g_2^{\sum_{i=1}^{k} r_i} = g_2^0 = e$, the identity element of the group. The attacker can increase the final tally to $(t + k.v)$. The tally verification procedure verifies all zero-knowledge proofs and checks the following equations: $\prod_{j\in\mathbb{C}} U_j = g_1^s, \prod_{j\in\mathbb{C}} V_j = g_2^s.g_2^{t+k.v}$, which will succeed in this case. The first of the two tally verification equations: $\prod_{j\in\mathbb{C}} U_j = g_1^{s+\sum_{i=1}^{k} r_i} = g_1^s g_1^{\sum_{i=1}^{k} r_i} = g_1^s$ and the second of the two tally verification equations: $\prod_{j\in\mathbb{C}} V_j = g_2^{s+\sum_{i=1}^{k} r_i} g_2^{t+k.v} = g_2^s g_2^{\sum_{i=1}^{k} r_i} g_2^{t+k.v} = g_2^s g_2^{t+k.v}$. This is a weakness of the system. The above mentioned ballots can be added at the end of the bulletin board. Note that the proof of well-formedness of the above mentioned confirmed ballots are correct, and hence the tally verification procedure of DRE-ip algorithm

[123] satisfies. The DRE-ip algorithm is developed to eliminate requirement of the tallying authority. However, as described above, an attacker can mount the above attack in the following scenario:

($i$) The bulletin board is secure and append-only; however, the adversary gets access to the DRE machine to get the signature of above ballots. Additionally, if the election authority colludes with the adversary, an attacker can post the above mentioned ballots at the end of the bulletin board. The election authority who has colluded with the adversary needs to increase the total count of cast votes. The signature key can be compromised at the setup stage, or a malicious DRE-ip software can generate such ballots.

***Possible countermeasure.*** To prevent the weakness 1, we assume that the names of voters who have cast their votes are not published on the bulletin board since it has a disadvantage: everybody learns who abstained from voting. Such information in some cases might be illegal to be revealed (see, for example, [2]). We have provided a secure and verifiable voter registration and authentication procedure that will generate a token. This token is presented to the DRE machine. The DRE checks the validity of the token. If the verification succeeds, the DRE allows the voter to cast her vote. We have described this in section 3.4.

To prevent this kind of attack, in our proposed algorithm described in section 3.5.1.1, we include hash of the previous ballot and a NIZK proof of a valid token number (generated by the voter authentication procedure) with each ballot. The hash of the previous ballot is included in the signature (for example, using Digital Signature Algorithm) of the current ballot. This creates an append-only list of ballots that cannot be modified by an adversary. The hash of the last ballot in the election is published with the final tally message along with two NIZK proofs: one of these two NIZK proofs proves the knowledge of the partial sum $s$; the other NIZK proof proves the knowledge of $s.prev\_hash$.

Note that the idea of a running hash is not new in the e-voting system. In 2007, Sandler and Wallach described the idea of hash linking of votes to ensure the integrity of their system [120] and later applied in the VoteBox system [119]. In 2011, Benaloh and Lazarus proposed a similar idea to mitigate their Trash attack [14], and in 2013, Bell et al. used a similar idea in their Star-Vote system [13]. We extend a similar idea to our proposed e-voting system.

## 3.4 Voter registration and authentication

In this section, we propose a secure and verifiable voter registration and authentication algorithm. We use a biometric encryption algorithm to bind a secret key with her biometric data (fingerprint). This secret key is one of the secret keys used for verifying the authenticity of the voter. We first describe the voter registration algorithm, and then we describe the voter authentication algorithm.

### 3.4.1 Voter registration

In this phase, voters will provide their personal information including voter identification number and fingerprint to register for the election. The voter registration process may occur throughout a year before the election. The voter registration data corresponding to each voter are displayed on a public bulletin board so that the public can verify the one-person-one-vote requirement. The following steps are performed to register a voter for the election.

1. The $i$-th voter provides her foundational identity card (for example, election ID card or Passport) to the voter registration official at the registration centre. The officer verifies the election ID card. Then the voter has her fingerprint scanned at the registration centre.

2. We use a biometric encryption algorithm proposed by Nandakumar et al. [99], called Fuzzy Vault, to bind a randomly generated key into the voter's fingerprint data. As shown in Figure 3.1 (a), a random number $r_{i1}$ is generated on enrollment uniformly and independently from $\mathbb{Z}_q$ so that neither the voter nor anybody knows it. This random number acts as one of the secret keys used in the registration process. The random number $r_{i1}$ itself is independent of the biometric, and hence it can be changed or modified. The random number $r_{i1}$ and the fingerprint template are securely deleted at the end of the enrollment.

In Fuzzy vault algorithm [99], the secret random key $r_{i1}$ is represented as coefficients of a polynomial in a Galois Field, for example, $GF(2^{16})$. This scheme is designed to secure a key of length $16n$ bits, where $n$ is the degree of the encoding polynomial. For example, if $n = 8$, we can secure a key of size 128 bits. The 16 bit x-coordinate value of the polynomial comprises the fingerprint minutiae location and the angle, and the corresponding Y-coordinate

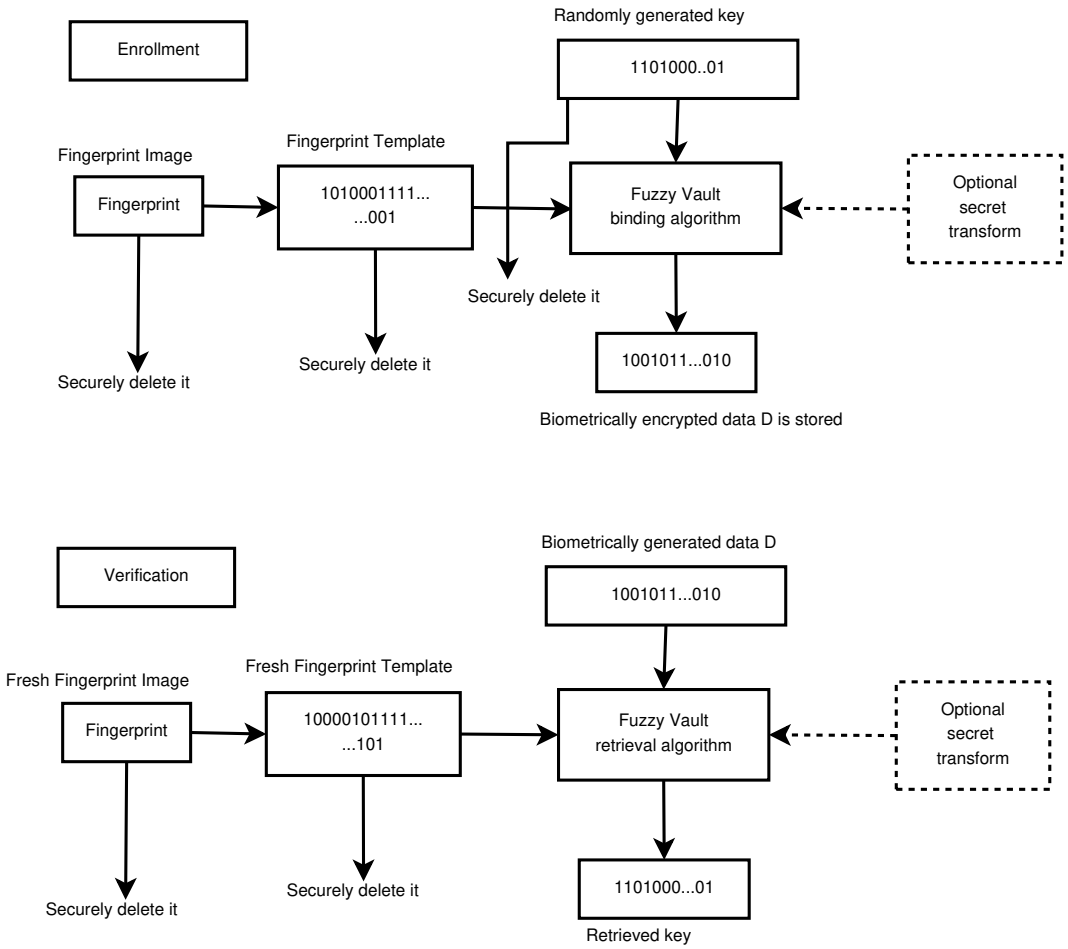


Figure 3.1: High level diagram of the Fuzzy Vault process to bind a key ($r_{i1}$ in the description) with the fingerprint. (a) Enrollment process that binds a randomly generated key ($r_{i1}$ in the description); (b) Verification process that retrieves the same key ($r_{i1}$ in the description).

is computed from the polynomial at the point x. Both the values x and y are stored with chaff points to hide the real minutiae. It also stores fingerprint alignment information. On verification, if a sufficient number of minutiae points coincide with some genuine recorded points, the full polynomial can be constructed using an error-correcting code (for example, Reed-Solomon error-correcting code) or Lagrange interpolation. The secret will be correctly decrypted only if the polynomial is correctly constructed. The performance results show that FRR (False Rejection Rate) is about 6% to 17% and FAR (False Acceptance Rate) is about 0.02%. The FRR is due to poor samples that are fact of life in biometric systems. However, the FRR could be significantly reduced through reties.

Since Fuzzy Vault actually stores real minutiae even though they are buried inside the

chaff points, this may become a source of potential vulnerability [67, 26, 27]. As a countermeasure, in [67], Jain et al. proposed a transform-in-the-middle approach (shown in the dashed-square in Figure 3.1) in which the fingerprint minutiae data are permuted based on a secret user's password.

3. The machine at the registration centre generates another random number $r_{i2}$ uniformly and independently from $\mathbb{Z}_q$. Then it computes $E(g^{H(r_{i1}||r_{i2})})$, where $g$ is the generator of the group $\mathbb{G}_q$, $E(g^{H(r_{i1}||r_{i2})})$ is the encryption of $g^{H(r_{i1}||r_{i2})}$ and '||' is the concatenate operation. We discuss this encryption procedure $E$ later in this section. Then the machine provides a signed receipt consisting $(VOTER\_ID_i, E(g^{H(r_{i1}||r_{i2})}))$ to the voter.

4. The random number $r_{i2}$ is given to the voter in a smart card or token. The voter keeps the smart card safely with her. It will be required during the voter authentication phase.

5. It publishes $(VOTER\_ID_i, E(g^{H(r_{i1}||r_{i2})}))$ on the public bulletin board. We assume that the device sends all the data to the BB over an authenticated channel using digital signature. The biometrically encrypted data $D_i$ along with $VOTER\_ID_i$ is securely stored by the government in a database or locally (for example, a token or smart card). The biometrically encrypted data $D_i$ will be used for verification of the voter during the voter authentication phase. It securely deletes the random numbers $r_{i1}, r_{i2}$ and the fingerprint template.

6. The voter leaves the registration centre, and checks that her receipt is recorded on the public bulletin board.

7. When the enrollment process finishes, all the receipts $(VOTER\_ID_i, E(g^{H(r_{i1}||r_{i2})}))$ are published on the public bulletin board. The bulletin board uses some mixnet servers proposed by Neff et al. [100] to permute the set of data $g^{H(r_{i1}||r_{i2})}$. The input to the mixnet are $(E(g^{H(r_{i1}||r_{i2})})), \forall i \in \{1, 2, ..., N\}$, where the encryption algorithm $E$ depends on the mixnet servers [100], $N$ is the total number of voters. It outputs $g^{H(r_{i1}||r_{i2})}, \forall i \in \{1, 2, ..., N\}$ in a verifiable manner. Figure 3.2 depicts the mixnet servers. The mixnet [100] servers provide NIZK proofs to prove a permutation between the input data $(E(g^{H(r_{i1}||r_{i2})})), \forall i \in \{1, 2, ..., N\}$ and output $g^{H(r_{i1}||r_{i2})}, \forall i \in \{1, 2, ..., N\}$ without revealing the exact permutation. We assume that at least one mixnet server is honest, who does not reveal its secret keys and the per-

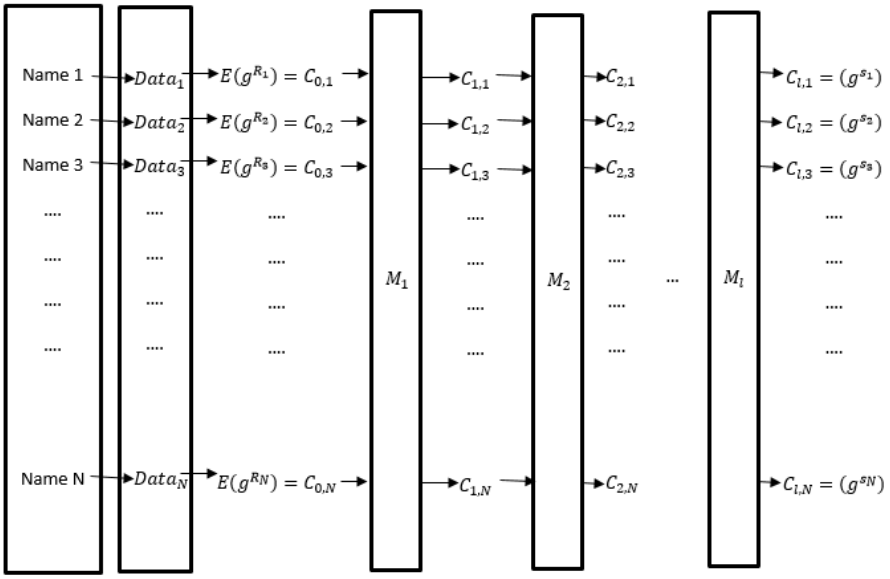


Figure 3.2: Public bulletin board displaying voter registration data of $N$ voters along with $l$ mixnet servers. Here, Name $i$ represents the name of the $i$-th voter. $Data_i$ represents the $i$-th voter's data $(VOTER\_ID_i, E(g^{H(r_{i1}||r_{i2})}))$. $R_i$ represents $H(r_{i1}||r_{i2})$. $M_k$ are mixnet servers $\forall k \in \{1, 2, ..., l\}$. $C_{k-1,j}, \forall j \in \{1, 2, ..., N\}$ are the inputs to the mixnet $M_k$, and its outputs are $C_{k,j}, \forall j \in \{1, 2, ..., N\}$.The last mixnet server's output $(g^{s_1}, g^{s_2}, ..., g^{s_N})$ is a permutation of $(g^{R_1}, g^{R_2}, ..., g^{R_N})$.

mutation.

### 3.4.2 Voter authentication during the voting phase

In this section, we discuss the voter authentication procedure using the biometrically encrypted data collected during the voter registration phase. The voter authentication is performed in parallel with the voting phase. The following steps are performed to verify the authenticity of a voter. After a voter's verification is successful, it generates a token number that will be used to cast her vote. All the voter's biometrically encrypted data $D_i$ along with $VOTER\_ID_i$ that were collected during the voter registration phase are presented (for example, in a smart card) here at this phase for verification of voters.

1. The voter goes to the polling station and provides her $VOTER\_ID_i$, fingerprint, the smart card containing $r_{i2}$ to verify her eligibility to vote.

2. We use the verification procedure of the Fuzzy Vault algorithm proposed by Nandakumar et al. [99] to verify her eligibility to vote. Figure 3.1 (b) illustrates the verification

procedure of the Fuzzy Vault algorithm [99]. On verification, the user provides her fresh fingerprint data which, when applied to the legitimate biomerically encrypted data $D_i$, will let the Fuzzy Vault algorithm recreate the same key $r_{i1}$. At the end of verification, the algorithm securely deletes the fingerprint template once again. The Fuzzy Vault algorithm is designed to account for an acceptable variations of the input fingerprint. On the other hand, an imposter with different enough fingerprint will not be able recreate the key.

3. It provides $token_i = H(r_{i1}||r_{i2})$ as a token number to the voter. The token number $token_i$ is given to the voter in an encrypted format using a symmetric key encryption. At the end of verification procedure, it securely deletes $r_{i1}, r_{i2}, token_i$ and the fingerprint once again.

4. The voter goes to the DRE machine and presents her encrypted token.

Voters must observe the public bulletin board (Figure 3.2) throughout the election stage to ensure that their registration data are not being changed. If, at any point between the voter registration and the final tallying phase, any of the entry on the public bulletin board is changed based on her receipt, the voter must raise an issue with the election official. Note that the proposed voter registration and authentication scheme can also be incorporated into other verifiable e-voting systems.

### 3.4.3 Performance analysis of voter registration and authentication

The voter registration phase is conducted well before the actual election. For each voter, the voter registration procedure includes execution of the Fuzzy Vault algorithm and computation of one encryption $E(g^{H(r_{i1}||r_{i2})})$, where the encryption algorithm $E$ depends on the mixnet server [100]. The mixnet [100] requires $8N$ exponentiations, where $N$ is the total number of voters registered for the election. Voter authentication includes execution of the Fuzzy Vault key retrieval algorithm and computation of a symmetric key encryption of $H(r_{i1}||r_{i2})$.

## 3.5 Voting system

In this section, we discuss the proposed voting scheme, analyse its security properties, propose efficient NIZK proofs, measure its performance and finally compare it with some well-known DRE-based voting systems.

### 3.5.1 Proposed voting scheme

We now describe the algorithm for the voting phase and the tallying phase.

#### 3.5.1.1 Voting and Tallying Phase

We have modified the DRE-ip system to prevent the weakness and the ballot stuffing attack. The system requires a publicly accessible bulletin board (BB). We assume that the BB can be insecure. However, in our system, if the adversary tries to modify the content of the BB, it could be detected by the public because of the use of hashchain.

We assume that one DRE machine is used to elect a candidate in a regional zone. Extension to multiple DRE machines is discussed in section 3.5.1.2. We assume that the DRE sends recorded ballots to BB over an authenticated channel using standard technique such as digital signatures. We describe the case where there are only two candidates i.e. if $v_i$ represents the vote for the $i$-th ballot, we have $v_i \in \{0, 1\}$. Extension to multiple candidates is discussed in subsection 3.5.1.3. An audited ballot is not used to cast a vote. Let $\mathbb{A}$, $\mathbb{C}$ and $\mathbb{B}$ are the set of all audited ballots, confirmed ballots and all ballots respectively. Thus, at the end of the voting phase, $\mathbb{B} = \mathbb{A} \cup \mathbb{C}$. We use the exponential ElGamal cryptosystem to encrypt the votes in which no party knows the decryption key. Figure 3.3 depicts a public bulletin board of our proposed e-voting system.

***Key Generation Phase.*** 1. The system generates an efficient description of a cyclic group $\mathbb{G}_q$ of order $q$ with two distinct generators $g_1, g_2$ whose logarithmic relationship is unknown.

2. The DRE publishes description of the group $(\mathbb{G}_q, q, g_1, g_2, Pk_{sign})$, where $Pk_{sign}$ is the public key of the digital signature scheme (for example, DSA) used by the DRE machine..

The group descriptions are shared between DRE and BB and it is published on the

BB. Initially, $t = 0$ and $s = 0$. The DRE calculates hash of this transaction and stores it in $prev\_hash$. The hash value $prev\_hash$ will be included in the next ballot. This hash must be verified by public during the tallying phase to ensure that this transaction remains unaltered.

***Voting Phase.*** This phase involves the voter, the DRE and the BB.

1. The voter goes to the DRE machine and presents her encrypted token. The DRE decrypts (using symmetric key) it to get the *token* (generated during the voter authentication phase). The DRE checks for an entry $g^{token}$ on the voter registration bulletin board (Figure 3.2) corresponding to her token. It also verifies that this token has not been already used to cast a vote, by checking all confirmed ballots on its own public bulletin board (that displays all ballots, Figure 3.3). If the verification succeeds, it allows the voter to proceed to the next step.

2. The voter initiates the voting and keys in her vote $v_i \in \{0, 1\}$.

3. The DRE generates random $r_i \in \mathbb{Z}_q^*$, and evaluates

$U_i = g_1^{r_i}, V_i = g_2^{r_i} g_2^{v_i}$,

$P_{WF}\{V_i : g_1, g_2, U_i\} = P_K\{r_i : ((U_i = g_1^{r_i}) \wedge (V_i = g_2^{r_i})) \vee ((U_i = g_1^{r_i}) \wedge (V_i/g_2 = g_2^{r_i}))\}$

$= P_K\{r_i : (U_i = g_1^{r_i}) \wedge ((V_i = g_2^{r_i}) \vee (V_i/g_2 = g_2^{r_i}))\}$,

and $P_K\{token : (W_i = g^{token})\}$.

Here, $P_{WF}\{V_i : g_1, g_2, U_i\}$ is a NIZK proof to show that the ballot is an encryption of either $v_i = 0$ or $v_i = 1$ (i.e. the ballot is well-formed). $P_K\{r_i\}$ is a NIZK proof of knowledge of $r_i$. Algorithm 3 (resp. Algorithm 4) and Algorithm 7 (resp. Algorithm 8) describe the creation (resp. verification) of the NIZK proof $P_{WF}\{V_i\}$ and $P_K\{token\}$ respectively. The DRE machine provides a signed receipt including the unique ballot index $i$ and the ballot content $(i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\}, P_K\{token\}, sign)$ to the voter, where $sign$ is the signature of $(i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\}, P_K\{token\})$.

4. The voter receives the first part of the receipt and chooses to either audit or confirm her vote.

In case of audit:

5. The DRE adds $i$ to $\mathbb{A}$. The DRE provides a signed receipt of the audit, marked as 'audited', including $r_i$, $v_i$ to the voter.

6. The voter takes and keeps the receipt. She verifies her choice of vote $v_i$. If the verification succeeds, it continues to execute next step, otherwise the voter should raise a dispute.

7. The DRE merges both parts into a single part $((i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\}, P_K\{token\}), audited, r_i, v_i, sign)$, where $sign$ is the signature of $((i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\}, P_K\{token\}), audited, r_i, v_i)$. The DRE posts this receipt (or transaction) on the BB. The DRE computes hash of this transaction and updates $prev\_hash$ value. This $prev\_hash$ will be attached to the next ballot (or transaction). The DRE repeats the process from step 2.

In case of confirmation:

5. The DRE adds $i$ to $\mathbb{C}$, updates the tally, the sum and evaluates:

$$t = \sum_{j \in \mathbb{C}} v_j, s = \sum_{j \in \mathbb{C}} r_j. \tag{3.5.1}$$

The DRE provides a signed receipt, marked as 'confirmed' to the voter. Then the DRE securely deletes both $r_i$ and $v_i$.

6. The voter leaves the booth with her receipts.

7. The DRE merges both parts into a single part, $((i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\}, P_K\{token\}), confirmed, sign)$, where $sign$ is the signature of $((i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\}, P_K\{token\}), confirmed)$. The DRE posts this transaction on the BB. The DRE computes hash of this transaction and stores it in $prev\_hash$. This $prev\_hash$ will be attached to the next ballot (or transaction).

8. The voter verifies that all her receipts match with those on the BB. The voter should raise an issue if her receipts do not match with those on the BB.

***Verification by bulletin board (cloud or blockchain).*** This phase involves the DRE and the underlying BB. The BB can be implemented using cloud server, Ethereum blockchain (method 1) or a combination of both blockchain and cloud server (method 2). The BB verifies the ballot well-formedness proof and the signature of the transaction before adding it into the BB.

***Tallying Phase.*** This phase involves the DRE, the BB and the public.

1. The DRE evaluates: $\Gamma_1 = g_1^s$, $\Gamma_2 = g_2^s$, $\Gamma_3 = g_1^{s.prev\_hash}$ and $\Gamma_4 = g_2^{s.prev\_hash}$. Then the DRE posts the final tally $t$, $\Gamma_1$, and $\Gamma_2$ on the BB with the zero-knowledge proofs $P_K\{s : (\Gamma_1 = g_1^s) \wedge (\Gamma_2 = g_2^s)\}$ and $P_K\{s.prev\_hash : (\Gamma_3 = g_1^{s.prev\_hash}) \wedge (\Gamma_4 = g_2^{s.prev\_hash})\}$. The Algorithm 9 (resp. Algorithm 10) provided in section 3.5.4.5 describes the procedure to create (resp. verify) these NIZK proofs. Let the tuple $(t, \Gamma_1, \Gamma_2, P_K\{s\}, P_K\{s.prev\_hash\}))$ be denoted by 'MESSAGE'. To post $i$-th message, say 'MESSAGE', on the BB, the following procedure is adopted.

i) The DRE creates a transaction consisting of the data $(i, prev\_hash, MESSAGE, final, sign)$, where $sign$ is the signature of $(i, prev\_hash, MESSAGE, final)$. The DRE posts this transaction on the BB.

2. The public:

i) verify that the hash of each transaction matches with the $prev\_hash$ value of the next ballot (or transaction).

ii) verify that the zero-knowledge proofs provided in step 1 of this tally phase are correct, provided the hash of the last ballot on the bulletin board. This step involves verification of the two NIZK proof: $(a)$ verification of $P_K\{s\}$, given $\Gamma_1$ and $\Gamma_2$; $(b)$ verification of $P_K\{s.prev\_hash\}$, provided $\Gamma_3$ $(= \Gamma_1^{prev\_hash})$ and $\Gamma_4$ $(= \Gamma_2^{prev\_hash})$, where $prev\_hash$ is the hash of the last ballot on the bulletin board.

iii) verify that all the well-formedness proofs of each transactions on the BB (well-formedness verification) are correct.

iv) verify that for all the audited ballots $((i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\}, P_K\{token\}), audited, r_i, v_i, sign)$ on the BB: the first part of the receipt $(i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\}, P_K\{token\})$ are consistent with $r_i$ and $v_i$.

v) verify that the signature of each transaction is valid.

vi) verify that all the NIZK proofs provided by the mixnet servers (Figure 3.2 in section 3.4.1) are correct.

vii) verify that all $g^{token}$ in all confirmed ballots on the BB (Figure 3.3) are different and there exists only one entry with value $g^{token}$ on the BB generated during the voter registration phase ( Figure 3.2 in section 3.4.1).

viii) verify that all the NIZK proof $P_K\{token\}$ are correct.

ix) verify that all the following equations hold (tally verification):

$$\prod_{j\in\mathbb{C}} U_j = \Gamma_1, \text{ and } \prod_{j\in\mathbb{C}} V_j = \Gamma_2.g_2^t. \tag{3.5.2}$$

During the voting and tallying phase, if any of the verification carried out by the voter or the public fails, the election authority should be notified. We assume that there are procedures in place to deal with such verification failures. In practice, a truncated hash function may be used to calculate short digest e.g. 32 bit long for each part of the receipt so that a voter can easily compare their receipt with those on the bulletin board. In this case, voters are expected to verify their receipts with those on the bulletin board. We assume that there are facilities provided for them to do so in the polling station.

If sufficient resources are available, there can be another optional module that takes a transaction from the proposed algorithm and checks whether it has been added to the BB.

#### 3.5.1.2 Extension to multiple DRE machines

If multiple DRE machines are used in a regional zone to elect a candidate, then instead of disclosing the tally from each DRE machine and then adding them together to get the final tally, we publish the final tally by combining the tallies from all DRE machines so that the tally from each DRE machine remains secret. The correctness of the procedure are realized by producing the corresponding zero-knowledge proof. This is particularly done to avoid revealing voter's distribution in small areas (where DRE machines are used to cast the vote) of a regional zone. Let the total number of DRE machines used in a regional zone be $\eta(\eta \in \mathbb{N})$. Let $t_{(i)}$ and $s_{(i)}$ represent the tally $t$ and sum of random variables $s$ respectively for the $i$-th DRE machine.

1. Each DRE performs following tasks.

i) The $i$-th DRE posts $\Gamma_{2(i)} = g_2^{t_{(i)}+s_{(i)}}$ on the BB instead of total tally $t_{(i)}$ with $P_K\{t_{(i)} + s_{(i)} : \Gamma_{2(i)} = g_2^{t_{(i)}+s_{(i)}}\}$, the non-interactive zero-knowledge proof of knowledge of the sum of tally $t_{(i)}$ and $s_{(i)}$ for this DRE machine. The DRE also posts $\Gamma_{1(i)} = g_1^{s_{(i)}}$ with $P_K\{s_{(i)} : \Gamma_{1(i)} = g_1^{s_{(i)}}\}$, the zero-knowledge proof of the sum of random variables $s_{(i)}$ for all confirmed ballots.

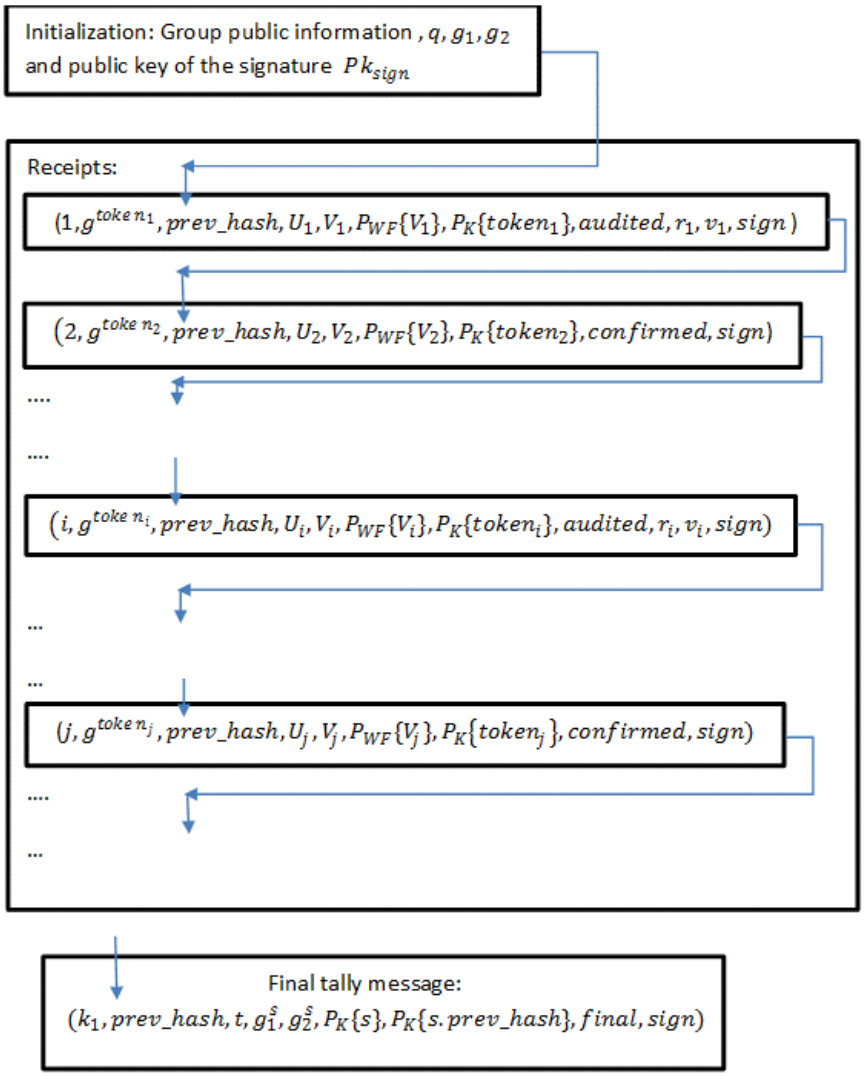


Figure 3.3: A public bulletin board of our system.

ii) Provided above information, the public perform all tasks stated in step 2 of the *Tallying Phase*.

2. (a) Let the group information $g_1$ and $g_2$ are same for every DRE machine. If they are different for each DRE machine, we perform step (b) instead of this step.

i) Now all DRE machines perform two secure multi-party computations ([53],[54],[10]) to evaluate $t_{final} = \Sigma_{i=1}^{\eta} t_{(i)}$ and $s_{final} = \Sigma_{i=1}^{\eta} s_{(i)}$. It publishes $t_{final}$ and $s_{final}$ as the final tally and the final sum of random variables.

ii) The public verifies $\prod_{i=1}^{\eta} \Gamma_{2(i)} = g_2^{t_{final}} g_2^{s_{final}}, \prod_{i=1}^{\eta} \Gamma_{1(i)} = g_1^{s_{final}}$.

(b) Let $g_{1(i)}$ and $g_{2(i)}$ represent the group information $g_1$ and $g_2$ respectively for the $i$-th

DRE machine.

i) Since, for every $i$-th DRE, $g_{1(i)}$ is public information, each DRE can send their $g_{1(i)}$ to each other to compute $g_a = \prod_{i=1}^{\eta} g_{1(i)}, g_b = \prod_{i=1}^{\eta} g_{2(i)}$. We assume that DRE machines can communicate with each other over an authenticated channel using digital signatures. It evaluates $\Gamma_{b(i)} = g_b^{t_{(i)}+s_{(i)}}$, $\Gamma_{2(i)} = g_{2(i)}^{t_{(i)}+s_{(i)}}$, $\Gamma_{a(i)} = g_a^{s_{(i)}}$, $\Gamma_{1(i)} = g_{1(i)}^{s_{(i)}}$,.

ii) Each DRE posts $(\Gamma_{b(i)}, \Gamma_{2(i)}), (\Gamma_{a(i)}, \Gamma_{1(i)})$ on the BB with $P_K\{t_{(i)} + s_{(i)} : \Gamma_{b(i)} = g_b^{t_{(i)}+s_{(i)}} \wedge \Gamma_{2(i)} = g_{2(i)}^{t_{(i)}+s_{(i)}}\}$ and $P_K\{s_{(i)} : \Gamma_{a(i)} = g_a^{s_{(i)}} \wedge \Gamma_{1(i)} = g_{1(i)}^{s_{(i)}}\}$. The public must verify these proofs.

iii) Now all DRE machines perform two secure multi-party computations ([53],[54],[10]) to evaluate $t_{final} = \Sigma_{i=1}^{\eta} t_{(i)}$ and $s_{final} = \Sigma_{i=1}^{\eta} s_{(i)}$. It publishes $t_{final}$ and $s_{final}$ as the final tally and the final sum of random variables.

iv) The public verifies $\prod_{i=1}^{\eta} \Gamma_{b(i)} = g_b^{t_{final}} g_b^{s_{final}}, \prod_{i=1}^{\eta} \Gamma_{a(i)} = g_a^{s_{final}}$.

***Secure multi-party computations to compute*** $\Sigma t$***.*** Here we briefly describe the secure multi-party computation for summation of their private inputs. Suppose three parties have private inputs $t_1, t_2, t_3$ respectively. The first party chooses random $t_{11} \in_R \mathbb{Z}_q$, and $t_{12} \in_R \mathbb{Z}_q$ uniformly and independently. Then it computes $t_{13} = (t_1 - t_{11} - t_{12})$. The first party sends $t_{12}$ to the second party and $t_{13}$ to the third party in encrypted format over an authenticated channel. Similarly, the second chooses random $t_{21} \in_R \mathbb{Z}_q$, and $t_{22} \in \mathbb{Z}_q$ uniformly and independently. Then it computes $t_{23} = (t_2 - t_{21} - t_{22})$. The second party sends $t_{21}$ to the first party and $t_{23}$ to the third party in encrypted format over an authenticated channel. Similarly, the third party chooses random $t_{31} \in_R \mathbb{Z}_q$, and $t_{32} \in \mathbb{Z}_q$ uniformly and independently. Then it computes $t_{33} = (t_3 - t_{31} - t_{32})$. The third party sends $t_{31}$ to the first party and $t_{32}$ to the second party in encrypted format over an authenticated channel. Now, after receiving $t_{21}$ and $t_{31}$ from the second and the third party respectively, the first party computes $T_1 = t_{11} + t_{21} + t_{31}$. Similarly, the second party computes $T_2 = t_{12} + t_{22} + t_{32}$ and the third party computes $T_3 = t_{13} + t_{23} + t_{33}$. Now all the three parties send their $T_i$'s to a fourth party who computes $T = T_1 + T_2 + T_3$, which is the sum of $t_1, t_2, t_3$. Note that the private input $t_i$ of each party remains secret after computation of their sum for all $i \in \{1, 2, 3\}$. In a similar fashion, untrusted parties can compute the sum of their private inputs without revealing it. The zero-knowledge proofs, verification by the public and the

final tally verification checks (step 2 of the *Tallying Phase*) collectively prove that all parties follow the protocol faithfully.

#### 3.5.1.3 Extension to multiple candidates

If there are $n(n >= 3)$ candidates contesting in the election, we will consider an upper bound, say $N$, on the number of voters and will encode the vote for the $j$-th candidate as $v = N^{j-1}$. The $i$-th ballot in that case will of the form $((i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\}, P_K\{token\}), audited, r_i, v_i, sign)$ in case of audit or $((i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\}, P_K\{token\}), confirmed, sign)$ in case of confirmed vote, where $V_i = g_2^{r_i} g_2^{N^{j-1}}$. The well-formedness proof $P_{WF}\{V_i\}$ will be 1-out-of-n disjunctive proof and can be stated as:

$P_{WF}\{V_i : g_1, g_2, U_i\} = P_K\{r_i : \vee_{j=1}^{n}((U_i = g_1^{r_i}) \wedge (V_i/g_2^{N^{j-1}} = g_2^{r_i}))\}$

$= P_K\{r_i : (U_i = g_1^{r_i}) \wedge (\vee_{j=1}^{n}(V_i/g_2^{N^{j-1}} = g_2^{r_i}))\}$.

The well-formedness proof $P_{WF}\{V_i\}$ is a NIZK proof to show that the ballot is an encryption of $v_i$, where $v_i \in \{1, N, N^2, ..., N^{n-1}\}$.

### 3.5.2 Storing recorded ballots

Depending on how the election is arranged, there can be several methods to store the ballots. In our modified DRE-ip system, any ballot is linked to the previous ballot by using the hash of the previous ballot in the digital signature of the ballot. This creates an append-only list that can be maintained by anyone. In this section, we highlight one existing method and propose two new methods to store these ballots.

**Using cloud server.** The cryptographic group information, the public keys of the ElGamal encryption and digital signature, all the ballots and the final tally messages can be stored in cloud server. The public bulletin board accesses the cloud storage to show the ballots. As stated previously, the hash of the previous ballot is included in the signature of the current ballot. This makes an append-only list of ballots that cannot be modified by an adversary without being detected by the public. Multiple cloud servers could be used to remove a single point of failure (see, for example, the public BB proposed by Palngipang et. al. [105]).

**Using public blockchain (method 1).** The cryptographic group information, the public keys of ElGamal encryption and digital signature, all the ballots and the final tally messages can be stored on a public blockchain such as Ethereum. The public and the individual voters can verify that their ballots are included into the blockchain. Once a ballot is posted on the blockchain, it remains tamper-proof. In our proposed system, each ballot includes a ballot well-formedness proof. Casting a ballot using our proposed protocol involves verifying the ballot well-formedness by the blockchain and storing it into the blockchain. We have improved the performance of the 1-out-of-n NIZK proof by reducing the number of exponentiations to almost half as compared to the NIZK proof described in [123]. The experimental results in section 3.5.6.1 shows the financial costs to mine each ballot in the blockchain. A small cartel of miners ($< 51\%$) may delay transactions from being accepted into the blockchain by using selfish mining or feather forking. Such ability of miners to delay a transaction is a fundamental problem for every smart contract. Figure 3.4 (resp. Figure 3.5) highlights how the cost in gas (resp. in US dollar) for storing and verifying each ballot on Ethereum blockchain vary with different number of candidates contesting in the election using the proposed efficient 1-out-of-n NIZK proof. We discuss the performance analysis (both cost and timing measurements analysis) of the procedure in detail in section 3.5.6.

**Using both the cloud server and the public blockchain (method 2).** Another method to implement a public bulletin board is to use both the blockchain and the cloud server. The cryptographic group information and the public keys of ElGamal encryption and digital signature are stored on a blockchain such as Ethereum in a single transaction before beginning of the polling phase. According to our proposed protocol, the hash of this transaction is included in the first ballot as well as in the signature of the first ballot. This transaction is the first transaction of the hash chain of ballots. During the tallying phase, the final tally message is also stored on a public blockchain such as Ethereum. The DRE sends the final tally message to the blockchain and verifies that it is included into the blockchain. According to our proposed protocol, the hash of the last valid cast ballot is included in the final tally message as well as in the signature of the final tally message. However, all the audited and confirmed ballots during the voting phase can be stored in a cloud server. Thus, if any of the already recorded ballot in the cloud is modified, it can

be detected by the public by observing the final tally message transaction and the group information transaction stored in the blockchain. It also prevents a malicious bulletin board from successfully adding a new ballot or deleting a recorded ballot in the cloud without detection. Since, for each DRE machine, we add only two transactions (the first and the last transaction of the hashchain) into the blockchain, the total financial cost towards mining these ballots is at most 16 million gas. Therefore, in this case, the cost of mining is almost negligible. In section 3.5.6.2, we analyse the timing measurements for posting each ballot.

### 3.5.3 Security analysis of the proposed system

In this section, we discuss the security properties of the proposed voter registration and authentication system. We also show that the proposed voting system is end-to-end verifiable, and it preserves ballot secrecy under both the intrusive and non-intrusive attacks.

#### 3.5.3.1 Security analysis of the voter registration and authentication procedure

We first discuss the correctness and the verifiability of the voter authentication procedure.

**One-Person-One-Vote without revealing any correspondence between the person and her ballot (encrypted vote).** Only one vote per one voter is ensured by the fact that an entry $(VOTER\_ID_i, E(g^{H(r_{i1}||r_{i2})}))$ on the public bulletin board (Figure 3.2) corresponds to only one entry $g^{H(r_{i1}||r_{i2})}$ on the output of the last mixnet server without revealing the exact correspondence via mixnet servers [100]. The correctness of the shuffling procedure can be verified by checking all the NIZK proofs provided by the mixnet servers [100]. This hides any relation between the voter's voter id $VOTER\_ID_i$ (or voter's name) and $g^{H(r_{i1}||r_{i2})}$. This means that it does not reveal any correspondence between the voter and her token number $H(r_{i1}||r_{i2})$ $(say, token_i)$. Figure 3.2 illustrates this fact. Now since $g^{token_i}$ and $P_K\{token_i\}$ are included in her ballot on the BB (Figure 3.3), one vote per one person is ensured by verifying the fact that $g^{token_i}$ in all confirmed ballots (described in section 3.5.1.1) on the BB (Figure 3.3) are different and there exists only one entry $g^{token_i}$ on the BB (Figure 3.2) generated during the voter registration phase (section 3.4.1) and by verifying the correctness of the NIZK proof $P_K\{token_i\}$. This can be verified by the public.

**Legitimacy of each voter.** On successful verification of a voter, the voter authentication procedure proves three things: the entry corresponds to the fingerprint of the legitimate voter, it can retrieve the correct random number $r_{i1}$ from her biometrically encrypted data $D_i$ and the voter has presented the correct random number $r_{i2}$ by providing her smart card. Therefore, the voter has provided all her secret keys and her fingerprint correctly. Hence, the legitimacy of the voter is ensured.

An imposter has to know two the secret keys $r_{i1}$ and $r_{i2}$ to generate a correct token number $H(r_{i1}||r_{i2})$ . The random number $r_{i1}$ is biometrically encrypted with the voter's fingerprint. The biometrically encrypted data $D_i$ is kept securely by the government. The random number $r_{i2}$ is kept by the voter in a smart card. Therefore, the probability to generate a correct token is $(1/2)^{\kappa}$, where $\kappa = min\{l_4, l_1 + l_2\}$, $l_4$ is the number of bits in the output of the hash function $H$, $l_1, l_2$ are the number of bits in $r_{i1}$ and $r_{i2}$ respectively. Thus, the ballot stuffing attack is prevented.

#### 3.5.3.2 End-to-End verifiability and integrity of the voting system.

We show that the proposed system achieves end-to-end verifiability. We also discuss the integrity of the election tally in our system. We show how voter-initiated auditing ensures that the votes are cast as intended and recorded as cast. We also prove that, assuming all well-formedness proofs are proof of knowledge, if all public verification succeed, votes are tallied as recorded. The number of voters is assumed to be less than the size of the group $q$.

Voter initiated auditing performs three checks. First, the voter observes that the first part of the receipt is provided before deciding whether to audit or confirm a ballot. Second, if the voter chooses to audit a ballot, she is provided with another receipt reflecting her choice of vote $v_i$ and randomness $r_i$. Thus the voter can verify that her choice of vote $v_i$ is correctly captured by the DRE. Third, the voter matches her receipts with those on the BB. Thus, the voter makes sure that her receipts are recorded by the BB without any modification. The public verification of the consistency of the audited ballots guarantees that the DRE has been successful to respond to the challenges made by the voter. Hence, the individual verification (step 6 of the voting phase in case of audit and step 8 of the voting phase) and the public audit consistency verification (step 2(iv) and 2(v) of the Tallying phase)

collectively guarantee that the votes are cast as intended and recorded as cast.

In the following theorem, we show that if the ballot well-formedness, signature and tally verification succeed, and hash of each transaction matches with the *prev_hash* field of the next transaction, the proposed system achieves *tallied as recorded* property.

**Theorem 3.5.1.** *In the proposed system, assuming that all proofs of well-formedness are proofs of knowledge, if the public ballot well-formedness, signature and token number and tally verifications succeed, and hash of each transaction matches with the prev_hash field of the next transaction, then the reported tally t is the correct tally of all confirmed votes on the bulletin board.*

The proof of the above theorem is rather straightforward and hence omitted here. In the proposed algorithm, any ballot is linked to its previous ballot by using the hash of the previous ballot in the signature of the ballot. This creates an append-only list of valid ballots. Having ballots linked to its previous ballot prevent a malicious bulletin board from adding new ballots. It can be shown how all the public verification, proof of well-formedness and the first tally verification checks (i.e. first of the two in equation 3.5.2) collectively guarantee that the second tally verification (i.e. the second of the two in equation 3.5.2) holds if and only if $t = \sum_{i \in \mathbb{C}} v_i$, where $\mathbb{C}$ is the set of all confirmed votes. Hence, if hash of the previous ballot matches with the *prev_hash* field of the current ballot and well-formedness, signature, token number and tally verification succeed, the reported tally $t$ is the correct tally of all confirmed ballots on the BB (Figure 3.3).

If the adversary does not get access to both the secret key of the underlying signature algorithm and a valid unused token number *token*, she cannot post a valid ballot on the BB. However, if the adversary gets access to the secret key of the underlying signature algorithm, a valid unused token number *token* and gets read access to the DRE storage variables (for example, partial sum $s$, partial tally $t$ and *prev_hash*), then she can post adversarial ballot on the BB; however, it will be detected immediately since the DRE will not be able to post the next ballot on the BB. The voter will not be able to match her receipts with those on the BB. Therefore, this attack can be detected immediately. The invalid transactions (adversarial ballots) that are causing this inconsistency can also be determined immediately

by observing the *prev_hash* field of the current ballot generated by the DRE. If facilities are available, the bulletin board may remove those invalid transactions from the end of the bulletin board to add the current valid ballot.

Note that, in DRE-ip, if an adversary gets access to the secret key of the underlying signature algorithm, the adversary can post ballots on the bulletin board in such a way that it cannot be detected by the tally verification process (including the public verification and tally verification process) in the tallying phase. A malicious DRE-ip system can also add such valid ballots on the BB. This has been described earlier in section 3.3.

#### 3.5.3.3 Ballot secrecy and the voter's privacy.

In this section, we prove that our scheme is secure against all probabilistic polynomial time adversary that try to deduce a particular voter's vote. We show that the public bulletin board does not reveal any information about a voter's vote except what a tally normally does. In an election with $n_1$ voters, if an attacker colludes with some $m_1$ number of voters, she will learn the partial tally of the remaining $(n_1 - m_1)$ voters; however, she will not be able to deduce any honest voter's vote. Note that the attacker can compute the partial tally by subtracting the colluding voter's votes from the final tally.

We assume that the discrete logarithm between $g_1$ and $g_2$ is either not known to any party or deleted securely (see, for example, [58] for secure deletion) during the voting phase. In addition, we assume secure deletion of the random value $r_i$ and the vote $v_i$ after each vote is cast.

We consider intrusive attacks to the DRE machine in which the adversary gets read access to the DRE storage for a certain period during the voting phase. The adversary is able to observe the vote cast during that access period and also gets read access to the running tally $t$, partial sum $s$ and the running hash *prev_hash*. The adversary can also read the publicly available information on the bulletin board. We prove that if an adversary can make temporary access to the DRE machine at a certain time $T$, she will only learn the partial tally of all cast votes from the start of the election to the time $T$ and from the time $T$ to the end of the election.

For the purpose of this proof, we consider an abridged bulletin board where the zero-

knowledge proofs of well-formedness are simulated. Assuming that the zero-knowledge proofs are secure, the adversary will only have negligible advantage while dealing with them. In the rest of the proof, we'll not mention the zero-knowledge proofs with each ballot; however, it is provided with each ballot.

**Lemma 3.5.1.** *In our proposed scheme, if an attacker colludes with some $m_1$ ($m_1 < n_1$) number of voters and gets read access to the DRE storage after $n_1$ voters cast their votes, she will only learn the partial tally of $(n_1 - m_1)$ uncompromised voters, not the individual votes of the uncompromised voters.*

*Proof.* Without any loss of generality, we assume that the indices of the uncompromised (honest) voters are $\{1, 2, ..., n_1 - m_1\}$, and that of the colluding voters are $\{n_1 - m_1 + 1, n_1 - m_1 + 2, ..., n_1\}$. The DRE stores only four variables: the sum of all randomness $s$ generated so far for creating the ballots; the running tally $t$ of all cast (confirmed) votes, the running hash $prev\_hash$ (hash of the previous ballot) and the unique ballot index $i$. Since $s$, $t$ are initialized with 0 at the beginning of the election, if the attacker gets read access to the DRE storage after $n_1$ voters have cast their votes, she will learn the partial tally of $n_1$ voters and partial sum of randomness used to create their ballots and the running hash $prev\_hash$ after $n_1$ ballots. The attacker will know all the colluding voters' vote; however, the randomness used to create their ballots will remain secret to the attacker. This is because, after a vote is cast, the DRE securely deletes the corresponding random number and the vote used to compute the confirmed ballot. Each ballot is of the form $(U_i, V_i)$ along with the unique ballot index $i$, running hash $prev\_hash$,$g^{token}$, $P_K\{token\}$, NIZK proof of ballot well-formedness and the signature of the ballot, where $U_i = g_1^{r_i}$ and $V_i = g_2^{r_i} g_2^{v_i}$ $\forall i \in \{1, 2, ..., n\}$. The attacker will know the colluding voters' votes $v_{n-m+1}, v_{n-m+2}, ..., v_n$. Hence, she can compute the partial tally of uncompromised voters' vote by subtracting these votes from the overall tally $t$. However, the randomness $r_i$ will remain secret to the attacker $\forall i \in \{1, 2, ..., n - m\}$. The randomness of an uncompromised voter, $r_i = s - \{r_1 + r_2 + ... + r_{i-1} + r_{i+1} + ... + r_n\}$. Now all $r_i$'s are uniformly and independently distributed over $\mathbb{Z}_q$. The partial sum $s$ is known to the attacker. Now if at least one of the random values $r_j$ where $j \in \{1, 2, ..., i-1, i+1, ..., n\}$ is unknown to the attacker, $r_i$ will be a random value unknown to the attacker. Hence, for

all uncompromised voters, the value $r_i$ to compute the ballot $(U_i, V_i)$ is random, uniformly distributed over $\mathbb{Z}_q$ and unknown to the attacker even if the attacker knows the partial sum $s$ and the running hash $prev\_hash$. Moreover, according to the protocol, the secret key of the encryption i.e. the discrete logarithm between $g_1$ and $g_2$ is either not known to any party or securely deleted during the setup stage. Hence, to deduce an uncompromised voter's vote from the ballot $(U_i, V_i)$, the attacker has to solve a discrete logarithm problem (DLP). Therefore, if the discrete logarithm problem (DLP) is hard to solve in the group $\mathbb{G}_q$, an attacker will not be able deduce an uncompromised voter's vote. □

The adversary can control arbitrary number of voters, and in effect she can cast arbitrary number of votes. We call the votes cast or observed by an adversary adversarial votes. The adversary can compute the tally of non-adversarial votes cast before and after the adversarial access period by using the knowledge of adversarial votes, final tally and the partial tally during the adversarial access period.

**Theorem 3.5.2.** *Let the elections begins at time $T_{begin}$ and finishes at time $T_{end}$. If the attacker who determines arbitrary number of votes and gets temporary read access to the DRE storage during a certain period $[T_0, T_1] \subset [T_{begin}, T_{end}]$ will only learn the partial tally of non-adversarial votes cast between the time $T_{begin}$ to $T_0$ and between $T_1$ to $T_{end}$, but not the individual non-adversarial votes.*

*Proof.* The proof of this theorem follows from Lemma 3.5.1. □

We have proved the theorem for one adversarial access period only. However, it can also be extended to multiple adversarial access period.

***Receipt-freeness.*** We consider a definition of receipt-freeness which requires that a voter cannot produce a receipt to prove that she votes in a particular way (i.e. for a particular candidate). Its purpose is to protect against vote buying. This definition originates from Benaloh [15]. The proposed system provides a receipt to the voter; however, the voter cannot prove that she votes in a particular way since her vote is encrypted. The public key $g_1$ of the ElGamal encryption algorithm can be generated from the group generator $g_2$ using a hashing algorithm in such a way that the discrete logarithm (secret key) relationship is not

known to any party including the DRE (or this secret key is securely deleted during the setup stage). Furthermore, after encrypting each confirmed vote, the DRE securely deletes corresponding random variable $r_i$ and the vote $v_i$. These information are not provided to the voter or stored in the DRE machine. Therefore, a voter using her receipts provided by the DRE cannot prove that she voted for a particular candidate. Hence, the the proposed scheme is receipt-free.

### 3.5.4 Zero-knowledge proofs

In this section, we present the NIZK proof algorithms that is required for well-formedness proof of the ballot. We first recall the NIZK proof algorithms used in DRE-ip [123]. We then recall the efficient NIZK proof algorithm proposed by Lin et al. in [91]. Subsequently, we analyze the efficient NIZK proof protocol proposed in [91] and show that it does not satisfy the required security properties of a zero-knowledge proof. In particular, the efficient NIZK proposed in [91] does not satisfy the completeness and the witness indistinguishability properties of zero-knowledge proof. Thereafter, we propose a NIZK proof algorithm that is more efficient than the NIZK proof presented in DRE-ip [123]. We prove that our proposed efficient NIZK proof algorithms satisfy all the required security properties of a NIZK proof. The security proofs of our proposed efficient NIZK proof algorithms are given in section 3.5.4.4.

Assume that there are $n$ candidates contesting in the election. Let us assume that the vote $v_i$ in the $i$ -th ballot is given to the $j$-th candidate, where $j \in \{1, 2, ..., n\}$. As discussed in section 3.5.1.3, we encode the vote for the $j$-th candidate as $v = N^{j-1}$, where $j \in \{1, 2, ..., n\}$. In this case, the $i$-th ballot will be of the form $((i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\},$ $P_K\{token\}), audited, r_i, v_i, sign)$ in case of audit or $((i, g^{token}, prev\_hash, U_i, V_i, P_{WF}\{V_i\},$ $P_K\{token\}), confirmed, sign)$ in case of confirmed vote, where $U_i = g_1^{r_i}$ and $V_i = g_2^{r_i} g_2^{N^{j-1}}$. The well-formedness proof $P_{WF}\{V_i\}$ is a 1-out-of-n disjunctive proof and can be stated as:

$P_{WF}\{V_i : g_1, g_2, U_i\} = P_K\{r_i : \vee_{j=1}^{n}((U_i = g_1^{r_i}) \wedge (V_i/g_2^{N^{j-1}} = g_2^{r_i}))\}$

$= P_K\{r_i : (U_i = g_1^{r_i}) \wedge (\vee_{j=1}^{n}(V_i/g_2^{N^{j-1}} = g_2^{r_i}))\}$.

Some notions are defined in Table 3.3.

| Notation | Description |
|---|---|
| $N$ | An upper bound on total number of voters |
| n | The number of candidates contesting in the election, $n(n >= 2)$ |
| $v_i$ | The vote corresponding to $i$-th ballot, the value $N^{j-1}$, where $j \in \{1, 2, ..., n\}$ |
| $\lambda$ | The secret $r_i$ corresponding to the $i$-th ballot |
| $U_i$ | For the $i$-th ballot, $g_1^{r_i}$ |
| $V_i$ | For the $i$-th ballot, $g_2^{r_i}.g_2^{N^{j-1}}$, where $j \in \{1, 2, ..., n\}$ |
| $\Gamma'$ | $U_i$ (i.e. $g_1^{r_i}$) corresponding to the $i$-th ballot |
| $\gamma'$ | It represents $g_1$ |
| $\Gamma''_l$ | $V_i/g_2^{N^{l-1}}$ corresponding to the $i$-th ballot, where $l \in \{1, 2, ..., n\}$ |
| $\gamma''_l$ | It represents $g_2, \forall l \in \{1, 2, ..., n\}$ |

Table 3.3: Notations

#### 3.5.4.1 Revisiting the 1-out-of-n NIZK proof used in DRE-ip

Algorithm 1 (resp. Algorithm 2) represents the prover algorithm (resp. verifier algorithm) for generation (resp. verification) of the 1-out-of-n NIZK proof used in DRE-ip. This 1-out-of-n NIZK proof is extended from the 1-out-of-2 NIZK proof given in [123]. Algorithm 1 (resp. Algorithm 2) is written to prove (resp. verify) a proposition of the form $\vee_{l=1}^{n}((\Gamma' = \{\gamma'\}^{\lambda}) \wedge (\Gamma''_l = \{\gamma''_l\}^{\lambda}))$.

**Algorithm 1:** A prover with identifier $ID$ generates a proof of knowledge of a secret $\lambda$ such that $\vee_{l=1}^{n}((\Gamma' = \{\gamma'\}^{\lambda}) \wedge (\Gamma''_l = \{\gamma''_l\}^{\lambda}))$ for known $ID, n, \Gamma', \gamma', (\Gamma''_l, \gamma''_l)_{l=1}^{n}$, where the vote is given to the $j$-th candidate, $j \in \{1, 2, ..., n\}$.

**Input :** $ID, n, \Gamma', \gamma', (\Gamma''_l, \gamma''_l)_{l=1}^{n}, \lambda, j$ such that $\Gamma' = \{\gamma'\}^{\lambda}$ and $\Gamma''_j = \{\gamma''_j\}^{\lambda}$
**Output:** $\Pi = P_K\{\lambda : \vee_{l=1}^{n}((\Gamma' = \{\gamma'\}^{\lambda}) \wedge (\Gamma''_l = \{\gamma''_l\}^{\lambda}))\}$
**begin**
  **choose** random $w, r_1, c_1, r_2, c_2, ..., r_{j-1}, c_{j-1}, r_{j+1}, c_{j+1}, ..., r_n, c_n \in \mathbb{Z}_q$
  **calculate** $t_{11} = \{\gamma'\}^{r_1}\{\Gamma'\}^{c_1}, t_{12} = \{\gamma''_1\}^{r_1}\{\Gamma''_1\}^{c_1}, t_{21} = \{\gamma'\}^{r_2}\{\Gamma'\}^{c_2}, t_{22} = \{\gamma''_2\}^{r_2}\{\Gamma''_2\}^{c_2}, ...., t_{j-11} = \{\gamma'\}^{r_{j-1}}\{\Gamma'\}^{c_{j-1}}, t_{j-12} = \{\gamma''_{j-1}\}^{r_{j-1}}\{\Gamma''_{j-1}\}^{c_{j-1}}, t_{j1} = \{\gamma'\}^{w}, t_{j2} = \{\gamma''_j\}^{w}, t_{j+11} = \{\gamma'\}^{r_{j+1}}\{\Gamma'\}^{c_{j+1}}, t_{j+12} = \{\gamma''_{j+1}\}^{r_{j+1}}\{\Gamma''_{j+1}\}^{c_{j+1}}, ..., t_{n1} = \{\gamma'\}^{r_n}\{\Gamma'\}^{c_n}, t_{n2} = \{\gamma''_n\}^{r_n}\{\Gamma''_n\}^{c_n}$
  **calculate**
  $c = H(ID, (\gamma', \Gamma', \gamma''_l, \Gamma''_l)_{l=1}^{n}, (t_{l1}, t_{l2})_{l=1}^{n})$,
  **calculate**
  $c_j = c - (c_1 + c_2 + .... + c_{j-1} + c_{j+1} + ... + c_n)$
  **calculate** $r_j = w - c_j\lambda$
  **return** $\Pi = (c_1, c_2, ...., c_{j-1}, c_j, c_{j+1}, ..., c_n, r_1, r_2, ..., r_{j-1}, r_j, r_{j+1}, ..., r_n)$
**end**

**Algorithm 2:** Verification of proof $\Pi$ generated by Algorithm 1 given $ID, n, \Gamma', \gamma', (\Gamma''_l, \gamma''_l)_{l=1}^n$. However, the verifier does not know to which candidate (i.e. $j$) the vote is given.

**Input :** $ID, n, \Gamma', \gamma', (\Gamma''_l, \gamma''_l)_{l=1}^n, \Pi = (c_1, c_2, ..., c_n, r_1, r_2, ..., r_n)$
**Output:** success or failure
**begin**
  **calculate**
  $t_{11} = \{\gamma'\}^{r_1}\{\Gamma'_1\}^{c_1}, t_{12} = \{\gamma''_1\}^{r_1}\{\Gamma''_1\}^{c_1}, t_{21} = \{\gamma'\}^{r_2}\{\Gamma'_2\}^{c_2}, t_{22} = \{\gamma''_2\}^{r_2}\{\Gamma''_2\}^{c_2}, ...,$
  $t_{n1} = \{\gamma'\}^{r_n}\{\Gamma'_n\}^{c_n}, t_{n2} = \{\gamma''_n\}^{r_n}\{\Gamma''_n\}^{c_n}$
  **calculate**
  $c' = H(ID, (\gamma', \Gamma', \gamma''_l, \Gamma''_l)_{l=1}^n, (t_{l1}, t_{l2})_{l=1}^n)$
  **if** $c' = (c_1 + c_2 + ... + c_n)$ **then**
    **return** success
  **else**
    **return** failure
  **end**
**end**

#### 3.5.4.2 Revisiting the efficient 1-out-of-n NIZK proof proposed by Lin et al. [91]

Lin et al. have proposed a 1-out-of-n NIZK proof [91] and claimed that it is more efficient than the original NIZK proof. We describe the 1-out-of-n NIZK proof proposed by Lin et al. [91] almost verbatim. We then analyze this NIZK proof and show that it does not satisfy the properties of zero-knowledge proof. We describe the 1-out-of-n NIZK proof in our case.

***1-out-of-n NIZK proof by Lin et al. [91].*** The prover and the verifier are divided into two parts respectively depending on the parity of $j$ (even or odd), where the vote is given to the $j$-th candidate.

The prover chooses a random number $w$. If $j$ is even, the prover chooses random numbers $r_\beta, d_\beta, \forall \beta \in \{2, 4, ..., n\}$; otherwise, if $j$ is odd, the prover chooses random numbers $r_\eta, d_\eta, \forall \eta \in \{1, 3, ..., n-1\}$. Thereafter, the prover calculates the vote $(U_i, V_i) = (g_1^{r_i}, g_2^{r_i} g_2^{N^{j-1}})$ and $(t_{j1}, t_{j2}) = (g_1^w, g_2^w)$. The prover then calculates $(t_{l1}, t_{l2}) = (g_1^{r_l} U_i^{d_l}, g_2^{r_l}\{V_i / g_2^{N^{l-1}}\}^{d_l})$, where $l = 2, 4, ..., n$ if $j$ is even or $l = 1, 3, ..., n-1$ if $j$ is odd. For non-interactiveness, $c_{all} = H(r_i||U_i||V_i)$. Then the prover computes $c_{even} = H(r_i||U_i||V_i||\{t_{l1}, t_{l2}\}_{l=2(l\in even)}^n)$ if $j$ is even or $c_{odd} = H(r_i||U_i||V_i||\{t_{l1}, t_{l2}\}_{l=1(l\in odd)}^{n-1})$ if $j$ is odd.

Then the prover computes the following parameters based on the parity of $j$.

If $j$ is even, the prover calculates $(t_{\beta 1}, t_{\beta 2}) = (g_1^{r_\beta} U_i^{d_\beta}, g_2^{r_\beta}\{V_i/g_2^{N^{\beta-1}}\}^{d_\beta}), c_{odd} = c_{all} - c_{even}, d_j = c_{even} - \sum_\beta d_\beta, r_j = w - r_i d_j$.

If $j$ is odd, the prover calculates $(t_{\eta 1}, t_{\eta 2}) = (g_1^{r_\eta} U_i^{d_\eta}, g_2^{r_\eta}\{V_i/g_2^{N^{\eta-1}}\}^{d_\eta}), c_{even} = c_{all} - c_{odd}, d_j = c_{odd} - \sum_\eta d_\eta, r_j = w - r_i d_j$.

Finally, the verifier sends $(\{t_{l1}, t_{l2}, d_l, r_l\}_{l=2(l\in even)}^{n}, U_i, V_i, c_{odd})$ to the verifier if $j$ is even or $(\{t_{l1}, t_{l2}, d_l, r_l\}_{l=1(l\in odd)}^{n-1}, U_i, V_i, c_{even})$ to the verifier if $j$ is odd.

The verifier verifies the correctness of $c_{even}$ or $c_{odd} = \sum_l d_l$, $c_{all} = c_{even} + c_{odd}$ and $(t_{l1}, t_{l2}) = (g_1^{r_l} U_i^{d_l}, g_2^{r_l}\{V_i/g_2^{N^{l-1}}\}^{d_l})$, where $l = \{2, 4, ..., n\} or \{1, 3, ..., n-1\}$. If the verifier verifies these conditions successfully, it returns success; otherwise, it returns failure.

***Analysis of this 1-out-of-n NIZK proof.*** According to the protocol, the verifier has to compute $H(r_i||U_i||V_i)$ or $H(r_i||U_i||V_i||\{t_{l1}, t_{l2}\}_{l=2(l\in even)}^{n})$ if $j$ is even or $H(r_i||U_i||V_i||\{t_{l1}, t_{l2}\}_{l=1(l\in odd)}^{n-1})$ if $j$ is odd to verify the conditions: $c_{even}$ or $c_{odd} = \sum_l d_l$, $c_{all} = c_{even} + c_{odd}$. However, the verifier cannot compute these hash functions since she does not know the secret value $r_i$ used in the argument of the hash function. Therefore, this 1-out-of-n NIZK proof does not satisfy the completeness property of the zero-knowledge proof.

Moreover, according to the protocol, the verifier has to verify the conditions $(t_{l1}, t_{l2}) = (g_1^{r_l} U_i^{d_l}, g_2^{r_l}\{V_i/g_2^{N^{l-1}}\}^{d_l})$, where $l = \{2, 4, ..., n\} or \{1, 3, ..., n-1\}$. To verify these conditions, the verifier has to know whether $j$ is even or odd. This means that the verifier has to know whether the voter has given her vote to an even numbered candidate or an odd numbered candidate. In other words, if the verifier verifies it successfully when $l = \{2, 4, ..., n\}$, the verifier will know that the voter has given her vote to an even numbered candidate (since $j$ is even). Similarly, if the verifier verifies it successfully when $l = \{1, 3, ..., n-1\}$, the verifier will know that the voter has given her vote to an odd numbered candidate (since $j$ is odd). Therefore, this 1-out-of-n NIZK proof does not satisfy the witness indistinguishability property of the zero-knowledge proof.

#### 3.5.4.3 Our proposed efficient 1-out-of-n NIZK proof

We propose an efficient 1-out-of-n NIZK proof. Our proposed 1-out-of-n NIZK proof satisfy all the required security properties of the zero-knowledge proof. The security proofs of

**Algorithm 3:** A prover with identifier $ID$ generates a proof of knowledge of a secret $\lambda$ such that $(\Gamma' = \{\gamma'\}^{\lambda}) \wedge (\vee_{l=1}^{n}(\Gamma''_l = \{\gamma''_l\}^{\lambda}))$ for known $ID, n, \Gamma', \gamma', (\Gamma''_l, \gamma''_l)_{l=1}^{n}$, where the vote is given to the $j$-th candidate, $j \in \{1, 2, ..., n\}$.

**Input :** $ID, n, \Gamma', \gamma', (\Gamma''_l, \gamma''_l)_{l=1}^{n}, \lambda, j$ such that $\Gamma' = \{\gamma'\}^{\lambda}$ and $\Gamma''_j = \{\gamma''_j\}^{\lambda}$
**Output:** $\Pi = P_K\{\lambda : (\Gamma' = \{\gamma'\}^{\lambda}) \wedge (\vee_{l=1}^{n}(\Gamma''_l = \{\gamma''_l\}^{\lambda}))\}$
**begin**
  **choose** random $w, r_1, c_1, r_2, c_2, ..., r_{j-1}, c_{j-1}, r_{j+1}, c_{j+1}, ..., r_n, c_n \in \mathbb{Z}_q$
  **calculate**
  $w_1 = w + (r_1 + r_2 + ... + r_{j-1} + r_{j+1} + ... + r_n) + (c_1 + c_2 + ... + c_{j-1} + c_{j+1} + ... + c_n)\lambda$

  **calculate** $t_1 = \{\gamma'\}^{w_1}, t_{12} = \{\gamma''_1\}^{r_1}\{\Gamma''_1\}^{c_1}, t_{22} = \{\gamma''_2\}^{r_2}\{\Gamma''_2\}^{c_2}, ...., t_{j-12} = \{\gamma''_{j-1}\}^{r_{j-1}}\{\Gamma''_{j-1}\}^{c_{j-1}}, t_{j2} = \{\gamma''_j\}^{w}, t_{j+12} = \{\gamma''_{j+1}\}^{r_{j+1}}\{\Gamma''_{j+1}\}^{c_{j+1}}, ..., t_{n2} = \{\gamma''_n\}^{r_n}\{\Gamma''_n\}^{c_n}$
  **calculate**
  $c = H(ID, \gamma', \Gamma', (\gamma''_l, \Gamma''_l)_{l=1}^{n}, t_1, (t_{l2})_{l=1}^{n})$,
  **calculate**
  $c_j = c - (c_1 + c_2 + .... + c_{j-1} + c_{j+1} + ... + c_n)$
  **calculate** $r_j = w - c_j\lambda$
  **return** $\Pi = (c_1, c_2, ...., c_{j-1}, c_j, c_{j+1}, ..., c_n, r_1, r_2, ..., r_{j-1}, r_j, r_{j+1}, ..., r_n)$
**end**

**Algorithm 4:** Verification of proof $\Pi$ generated by Algorithm 3 given $ID, n, \Gamma', \gamma', (\Gamma''_l, \gamma''_l)_{l=1}^{n}$. However, the verifier does not know to which candidate (i.e. $j$) the vote is given.

**Input :** $ID, n, \Gamma', \gamma', (\Gamma''_l, \gamma''_l)_{l=1}^{n}, \Pi = (c_1, c_2, ..., c_n, r_1, r_2, ..., r_n)$
**Output:** success or failure
**begin**
  **calculate**
  $t_1 = \{\gamma'\}^{r_1+r_2+...+r_n}\{\Gamma'_1\}^{c_1+c_2+...+c_n}, t_{12} = \{\gamma''_1\}^{r_1}\{\Gamma''_1\}^{c_1}, t_{22} = \{\gamma''_2\}^{r_2}\{\Gamma''_2\}^{c_2}, ..., t_{n2} = \{\gamma''_n\}^{r_n}\{\Gamma''_n\}^{c_n}$
  **calculate**
  $c' = H(ID, \gamma', \Gamma', (\gamma''_l, \Gamma''_l)_{l=1}^{n}, t_1, (t_{l2})_{l=1}^{n})$
  **if** $c' = (c_1 + c_2 + ... + c_n)$ **then**
    **return** success
  **else**
    **return** failure
  **end**

**end**

the proposed 1-out-of-n NIZK proof are given in section 3.5.4.4. We have modified the zero-knowledge proof involving the conjunction and disjunction of predicates to improve its performance. The 1-out-of-n (Algorithm 3 and Algorithm 4) proofs presented here are

efficient than the proofs presented in DRE-ip [123]. Algorithm 3 (resp. Algorithm 4) is written to prove (resp. verify) a proposition of the form $\vee_{l=1}^{n}((\Gamma' = \{\gamma'\}^{\lambda}) \wedge (\Gamma_l'' = \{\gamma_l''\}^{\lambda}))$ which is equivalent to $(\Gamma' = \{\gamma'\}^{\lambda}) \wedge (\vee_{l=1}^{n}(\Gamma_l'' = \{\gamma_l''\}^{\lambda}))$. As we assumed previously, the vote is given to the $j$-th candidate. We assume that the simultaneous multiple exponentiation (SME) [97] technique is used to optimize the computation of a term of the form $g^x h^y$. The computation cost of the term $g^x h^y$ is around 1.2 exponentiation using SME technique. The prover algorithm (Algorithm 3) requires $(1.2(n-1)+2)$ exponentiations; however, the prover algorithm (Algorithm 1) presented in DRE-ip requires $(2.4(n-1)+2)$ exponentiations. To verify such zero-knowledge proof, the verifier algorithm (Algorithm 4) requires $1.2(n+1)$ exponentiations; however, the verification algorithm (Algorithm 2) presented in DRE-ip requires $2.4n$ exponentiations. In Table 3.4 , we theoretically analyse the cost of execution of the prover and the verifier of the 1-out-of-n NIZK proof and our proposed efficient 1-out-of-n NIZK proof. Since exponention operations are the most time consuming operations, we only include the number of exponentiations in theoretical analysis (Table 3.4).

| Scheme | Prover | Verifier |
|---|---|---|
| 1-out-of-n NIZK proof | $(2.4(n-1)+2)e$ | $2.4ne$ |
| Proposed 1-out-of-n NIZK proof | $(1.2(n-1)+2)e$ | $1.2(n+1)e$ |

Table 3.4: Computation complexity of the 1-out-of-n NIZK and the proposed 1-out-of-n NIZK proof. $e$ represents the exponentiation operation.

These algorithms can be extended to prove any proposition of the form $\wedge_{i=1}^{k}\varphi_i \wedge (\vee_{l=1)}^{n}\psi_l)$ for a set of assertions $\{\varphi_1, \varphi_2, ..., \varphi_k, \psi_1, \psi_2, ..., \psi_n\}$, where the number $k$ and $n$ are known to both the prover and the verifier. To generate such zero-knowledge proof, the prover algorithm (extended version of Algorithm 3) requires $(1.2(n-1)+k+1)$ exponentiations; however, the prover algorithm (extended version of Algorithm 1) presented in DRE-ip requires $(1.2(k+1)(n-1)+k+1)$ exponentiations. To verify such zero-knowledge proof, the verifier algorithm (extended version of Algorithm 4) requires $1.2(n+k)$ exponentiations; however, the verification algorithm (extended version of Algorithm 2) presented in DRE-ip requires $1.2n(k+1)$ exponentiations. Therefore, the proposed NIZK proofs are almost $(k+1)$ times more efficient than the NIZK proofs presented in [123].

#### 3.5.4.4 Security properties of zero-knowledge proof of the prover Algorithm 3 and the verifier Algorithm 4.

**Algorithm 5:** A prover with identifier $ID$ generates a proof of knowledge of a secret $\lambda$ such that $(\Gamma' = \{\gamma'\}^{\lambda}) \wedge ((\Gamma''_1 = \{\gamma''_1\}^{\lambda}) \vee (\Gamma''_2 = \{\gamma''_2\}^{\lambda}))$.

**Input :** $ID, \Gamma', \gamma', (\Gamma''_l, \gamma''_l)_{l=1}^2, \lambda$ such that $\Gamma' = \{\gamma'\}^{\lambda}$ and $\Gamma''_1 = \{\gamma''_1\}^{\lambda}$
**Output:** $\Pi = P_K\{\lambda\colon (\Gamma' = \{\gamma'\}^{\lambda}) \wedge ((\Gamma''_1 = \{\gamma''_1\}^{\lambda}) \vee (\Gamma''_2 = \{\gamma''_2\}^{\lambda}))\}$
**begin**
**choose** random $w, r_2, c_2 \in \mathbb{Z}_q$
**calculate** $w_1 = w + r_2 + c_2\lambda$
**calculate**

$$t_1 = \{\gamma'\}^{w_1}, t_{12} = \{\gamma''_1\}^{w}, t_{22} = \{\gamma''_2\}^{r_2}\{\Gamma''_2\}^{c_2} \tag{3.5.3}$$

**calculate**
$c = H(ID, \gamma', \Gamma', (\gamma''_l, \Gamma''_l)_{l=1}^2, t_1, (t_{l2})_{l=1}^2)$,

$$c_1 = c - c_2 \tag{3.5.4}$$

**calculate**

$$r_1 = w - c_1\lambda \tag{3.5.5}$$

**return** $\eta = (c_1, c_2, r_1, r_2)$
**end**

**Algorithm 6:** Verification of proof $\Pi$ generated by *Algorithm* 5 given $ID, \Gamma', \gamma', (\Gamma''_l, \gamma''_l)_{l=1}^2$. The discrete logarithmic relationships for the pairs $(\gamma', \gamma''_1)$ and $(\gamma', \gamma''_2)$ are unknown.

**Input :** $ID, \Gamma', \gamma', (\Gamma''_l, \gamma_l)_{l=1}^2, \Pi = (c_1, c_2, r_1, r_2)$
**Output:** successful or failure
**begin**
**calculate**
$t_1 = \{\gamma'\}^{r_1+r_2}\{\Gamma'\}^{c_1+c_2}$, $t_{12} = \{\gamma''_1\}^{r_1}\{\Gamma''_1\}^{c_1}$ and
$t_{22} = \{\gamma''_2\}^{r_2}\{\Gamma''_2\}^{c_2}$
**calculate**
$c' = H(ID, \gamma', \Gamma', (\gamma''_l, \Gamma''_l)_{l=1}^2, t_1, (t_{l2})_{l=1}^2)$
**if** $c' = c_1 + c_2$ **then**
  **return** successful
**else**
  **return** failure
**end**
**end**

We have proved the security properties of the proposed efficient NIZK proof for two candidates i.e. for 1-out-of-2 NIZK proof. The security proofs can be easily extended for $n$

candidates i.e. for 1-out-of-n NIZK proof. Algorithm 5 (resp. Algorithm 6) is presented to prove (rep. verify) a proposition of the form $\varphi' \wedge (\varphi_1 \vee \varphi_2)$ for three assertions $\varphi'$, $\varphi_1$ and $\varphi_2$, where the prover knows discrete logarithms for the pair $(\varphi', \varphi_1)$. Here $\varphi'$, $\varphi_1$ and $\varphi_2$ are assertions $(\Gamma' = \{\gamma'\}^{\lambda}), (\Gamma''_1 = \{\gamma''_1\}^{\lambda})$ and $(\Gamma''_2 = \{\gamma''_2\}^{\lambda})$ respectively. In order for prover $P$ and verifier $V$ to achieve the security properties, we must restrict the computational power of $V$ or any attacker so that it is bounded by a polynomial in the size of common input. Clearly, without this restriction we need not talk about zero-knowledge since $V$ of an unbounded computational power can find $P$'s private input hidden behind common input. The discrete logarithm for the pairs $(\gamma', \gamma''_1)$ and $(\gamma', \gamma''_2)$ are unknown. Note that this assumption is also made in [123] for construction of 1-out-of-n NIZK. Otherwise, an attacker can find out the set corresponding to the witness i.e. the witness indistinguishability property will be lost.

*Completeness:*

By direct observation of the protocol, it is straightforward to see that the completeness property is preserved. This means that, if the prover generates $(c_1, c_2, r_1, r_2)$ and follow the protocol instruction, the honest verifier will always accept it.

*Soundness:*

We need to find soundness error probability.

Let us assume that for the same commitment $(t_1, t_{12}, t_{22})$ with fixed $r_2, c_2, w$, two different response viz. $(c_1, c_2, r_1, r_2)$ and $(c_1^1, c_2^1, r_1^1, r_2^1)$ are generated, where $c_1 \neq c_1^1$. Now we can compute a witness for $\lambda$ i.e. $\lambda = (r_1 - r_1^1)/(c_1^1 - c_1)$. Therefore, the prover $P$ knows the witness $\lambda$. Similarly, let us assume that for the same commitment $(t_1, t_{12}, t_{22})$ with fixed $r_1, c_1, w$, two different response viz. $(c_1, c_2, r_1, r_2)$ and $(c_1^1, c_2^1, r_1^1, r_2^1)$ are generated, where $c_2 \neq c_2^1$. Now we can compute a witness for $\lambda$ i.e. $\lambda = (r_2 - r_2^1)/(c_2^1 - c_2)$. Therefore, in this case also, the prover $P$ knows the witness $\lambda$.

Suppose a prover, $P^*$, is a cheater, i.e., he does not know the correct discrete logarithm value for any of the pair $(\varphi', \varphi_1)$ or $(\varphi', \varphi_2)$.

For a commitment $(t_1, t_{12}, t_{22})$ he chooses in equation 3.5.3, the verifier is waiting for a response $(c_1, c_2, r_1, r_2)$ such that

$$c_1 + c_2 = H(ID, \gamma', \Gamma', (\gamma''_l, \Gamma''_l)_{l=1}^2, \{\gamma'\}^{r_1+r_2}\{\Gamma'\}^{c_1+c_2}, \{\gamma''_1\}^{r_1}\{\Gamma''_1\}^{c_1}, \{\gamma''_2\}^{r_2}\{\Gamma''_2\}^{c_2}). \quad (3.5.6)$$

Since the prover, $P^*$, does not know the correct discrete logarithm, the best known strategy to compute such response is to guess $(r_1, r_2)$ and any one of $c_1$ or $c_2$ first as follows:

1. picking random $r_1 \in_R \mathbb{Z}_q$ and $r_2 \in_R \mathbb{Z}_q$ uniformly;

2. picking either of $c_1$ or $c_2$ uniformly from the image space of the hash function $H$ i.e. picking $c_2 \in_R Image(H)$ (assume $H$ has the same large output space as $\mathbb{Z}_q$)

3. computing

$c_1 = H(ID, \gamma', \Gamma', (\gamma_l'', \Gamma_l'')_{l=1}^2, \{\gamma'\}^{r_1+r_2}\{\Gamma'\}^{c_1+c_2}, \{\gamma_1''\}^{r_1}\{\Gamma_1''\}^{c_1}, \{\gamma_2''\}^{r_2}\{\Gamma_2''\}^{c_2}) - c_2$. (or $c_2 = H(ID,$
$\gamma', \Gamma', (\gamma_l'', \Gamma_l'')_{l=1}^2, \{\gamma'\}^{r_1+r_2}\{\Gamma'\}^{c_1+c_2}, \{\gamma_1''\}^{r_1}\{\Gamma_1''\}^{c_1}, \{\gamma_2''\}^{r_2}\{\Gamma_2''\}^{c_2}) - c_1$, if he picks $c_1 \in_R$ $Image(H)$ in step 2).

This is a well-known computationally hard problem to find such a $c_1$ since $H$ is a random oracle and $g^x$ is a one-way function. Now since he does not know the discrete logarithm corresponding to any of the pair $(\varphi', \varphi_1)$ or $(\varphi', \varphi_2)$ and $H$ is a random oracle and $g^x$ is a one-way function, the soundness error probability is $(1/2^n)$, where $n$ is the security parameter $log(q)$.

*Zero-knowledgeness:*

We'll show that although the algorithm is not a full zero-knowledge, the algorithm does not reveal any information about $P$'s private input $\lambda$. Note that the Schnorr signature scheme (NIZK) and the 1-out-of-n zero-knowledge proof described in [123] are also not a full zero-knowledge but they does not reveal any information about $P$'s private input.

For a response $(c_1, c_2, r_1, r_2)$ to be valid and accepted by the verifier $V$, they must satisfy the equation

$$c_1 = H(ID, \gamma', \Gamma', (\gamma_l'', \Gamma_l'')_{l=1}^2, \{\gamma'\}^{r_1+r_2}\{\Gamma'\}^{c_1+c_2}, \{\gamma_1''\}^{r_1}\{\Gamma_1''\}^{c_1}, \{\gamma_2''\}^{r_2}\{\Gamma_2''\}^{c_2}) - c_2. \quad (3.5.7)$$

Viewed by a third party, equation 3.5.7 means either of the following two cases:

1. the equation is constructed by $P$ using her private input, hence $P$ discloses that she has been in interaction with verifier $V$, or

2. an attacker or a verifier has successfully broken the random oracle hash function $H$ of large output space $\mathbb{Z}_q$ and another one-way function $g^x$, because she has constructed the

equation 3.5.7.

This is a well-known hard problem since $H$ is a random oracle and $g^x$ is a one-way function.

Since an attacker or verifier is polynomially bounded, the third party will of course believe that (1) is the case. The response $(c_1, c_2, r_1, r_2)$ is precisely a signature under Schnorr's signature scheme. Since only $P$ could have issued such signature, the third party has made correct judgement.

A simulator cannot generate such response $(c_1, c_2, r_1, r_2)$ without knowing the discrete logarithm corresponding to any of the pair $(\varphi', \varphi_1)$ or $(\varphi', \varphi_2)$.

Therefore, it is not a full zero-knowledge. However, we'll show that the proof transcript $(c_1, c_2, r_1, r_2)$ does not reveal any information about $P$'s private input (discrete logarithm).

Let us consider the equation 3.5.5 calculated by prover $P$ i.e.

$r_1 = w - c_1\lambda$. Here $w$ is chosen uniformly from $\mathbb{Z}_q$ independent from all previous instances and $\lambda$ is $P$'s private input. After receiving a valid response $(c_1, c_2, r_1, r_2)$, $c_1$ and $r_1$ are known to the verifier $V$ and any attacker.

Note that $c_2$ and $r_2$ are chosen uniformly from $\mathbb{Z}_q$ independent from all previous instances. Let $\tilde{C_2}$ and $\tilde{R_2}$ denote random variables corresponding to $c_2$ and $r_2$ respectively.

$c$ being the output of the hash function $H$ is also uniformly distributed over the image space of $H$. Let us assume the image space of $H$ is $\mathbb{Z}_q$. Let $\tilde{C}$ denote the random variable corresponding to $c$.

We assume that there is a probability distribution for $\lambda$ which determines a random variable $\tilde{\lambda}$. Let $\tilde{W}$ denotes the random variable corresponding to $w$. The three random variables viz. $\tilde{C_1}$, $\tilde{W}$ and $\tilde{\lambda}$ determine a random variable $\tilde{R}$ over $\mathbb{Z}_q$ representing $r_1$, where $r_1 = w - c_1\lambda$.

Let $\tilde{t_1}$, $\tilde{t_{12}}$ and $\tilde{t_{22}}$ denote the random variables representing $t_1 = \{\gamma'\}^{w_1}$, $t_{12} = \{\gamma_1''\}^w$ and $t_{22} = \{\gamma_2''\}^{r_2}\{\Gamma_2''\}^{c_2}$ respectively, where $w_1 = w + r_2 + c_2\lambda$.

Now, since $r_2$ and $c_2$ are chosen uniformly and independently from $\mathbb{Z}_q$, the random variables $\tilde{C_2}$ and $\tilde{t_{22}}$ are also independent.

Also since $r_2$, $c_2$ and $w$ are chosen uniformly and independently from $\mathbb{Z}_q$, the random variables $\tilde{C_2}$ and $\tilde{t_1}$ are also independent.

Hence the random variables $\tilde{C_2}$ and $\tilde{C}$ are also independent, where the random variable $\tilde{C}$ represents $c = H(ID, \gamma', \Gamma', (\gamma_l'', \Gamma_l'')_{l=1}^2, t_1, (t_{l2})_{l=1}^2)$.

Consider the equation 3.5.4 i.e. $c_1 = c - c_2$. Here $c_2$ is uniformly distributed and independent of $c$. Therefore, the random variable $\tilde{C_1}$ is also uniformly distributed and independent of $\tilde{C}$. This follows from following well-known result from probability theory.

Two random variables $X_1$ and $X_2$ are such that $(X_1, X_2) \in \mathbb{G} \times \mathbb{G}$, where $(\mathbb{G}, +)$ is a group. Given that (1) $X_1$ and $X_2$ are independent and (2) $X_1$ is uniform over $\mathbb{G}$. Let $X_3 = X_1 + X_2$, then (1) $X_2$ and $X_3$ are independent and (2) $X_3$ has uniform distribution over $\mathbb{G}$.

Since $\tilde{C_1}$ is independent of $\tilde{C}$, $\tilde{C_1}$ is also independent of $\tilde{W}$ and $\tilde{\lambda}$.

Therefore, three random variables $\tilde{W}$, $\tilde{C_1}$ and $\tilde{\lambda}$ are uniformly distributed over $\mathbb{Z}_q$ and independent of each other. Let $n$ denotes the security parameter $log(q)$. Now, $Pr[\tilde{R} = r_1]$ $= \Sigma_{w \in \mathbb{Z}_q} \Sigma_{c_1 \in \mathbb{Z}_q} Pr[\tilde{W} = w] Pr[\tilde{C_1} = c_1] Pr[\tilde{\lambda} = (w - r_1){c_1}^{-1}] = (1/2^n) \Sigma_{w \in \mathbb{Z}_q} \Sigma_{c_1 \in \mathbb{Z}_q} Pr[\tilde{C_1} = c_1] Pr[\tilde{\lambda} = (w - r_1){c_1}^{-1}] = (1/2^n)^2 \Sigma_{w \in \mathbb{Z}_q} \Sigma_{c_1 \in \mathbb{Z}_q} Pr[\tilde{\lambda} = (w - r_1){c_1}^{-1}] = (1/2^n)^2 \Sigma_{w \in \mathbb{Z}_q} 1$ $= (1/2^n)^2 . (2^n) = (1/2^n)$.

$Pr[\tilde{R} = r_1 | \tilde{\lambda} = \lambda] = \Sigma_{c_1 \in \mathbb{Z}_q} Pr[\tilde{C_1} = c_1] Pr[\tilde{W} = r_1 + c_1\lambda] = (1/2^n) \Sigma_{c_1 \in \mathbb{Z}_q} Pr[\tilde{W} = r_1 + c_1\lambda] = (1/2^n)$.

Therefore, $Pr[\tilde{\lambda} = \lambda | \tilde{R} = r_1] = (Pr[\tilde{\lambda} = \lambda] Pr[\tilde{R} = r_1 | \tilde{\lambda} = \lambda] / Pr[(\tilde{R} = r_1]) = (Pr[\tilde{\lambda} = \lambda].(1/2^n))/(1/2^n)$

$$= Pr[\tilde{\lambda} = \lambda]. \tag{3.5.8}$$

Therefore, the $r_1$ does not reveal any information about $P$'s private input $\lambda$. $r_1$ forms a one-time pad (shift cipher) encryption of $P$'s private input $\lambda$, which provides information-theoritic quality of security i.e. it has prefect secrecy.

Although we have used the same $\lambda$ for constructing $t_1 = \{\gamma'\}^{w_1}$, where $w_1 = w + r_2 + c_2\lambda$. Due to hardness of the discrete logarithm problem, an attacker or a verifier cannot find $w_1$ from $\{\gamma'\}^{w_1}$. Also, the logarithmic relationship for the pairs $(\gamma', \gamma_1''), (\gamma', \gamma_2'')$ are unknown. $t_1$ does not reveal any more information about $P$'s private input $\lambda$ than that has already been revealed by their common input $\Gamma' = \{\gamma'\}^{\lambda}$.

Therefore, the algorithm does not reveal any more information about $P$'s private input

$\lambda$ than that has been revealed by their common inputs $\Gamma'$, $\Gamma''_1$, and $\Gamma''_2$.

*Witness Indistinguishability:*

We have to show that the distribution of the conversation is independent of the qualified set $A$ corresponding to $P$'s private input $\lambda$. Since the prover generates a proof of knowledge of a secret $\lambda$ such that $(\Gamma' = \{\gamma'\}^{\lambda}) \wedge ((\Gamma''_1 = \{\gamma''_1\}^{\lambda}) \vee (\Gamma''_2 = \{\gamma''_2\}^{\lambda}))$. A verifier and an attacker already know that the assertion $(\Gamma' = \{\gamma'\}^{\lambda})$ is in the qualified set $A$. Our aim is to prevent an attacker or a verifier to know that which one among $(\Gamma''_1 = \{\gamma''_1\}^{\lambda})$ and $(\Gamma''_2 = \{\gamma''_2\}^{\lambda}))$ corresponds to the $P$'s private input $\lambda$. Let $\varphi' = (\Gamma' = \{\gamma'\}^{\lambda})$ and $\varphi_l = (\Gamma''_l = \{\gamma''_l\}^{\lambda})$, where $l = 1, 2$.

We use the same notation and random variables described in the previous section.

Since, $w$, $c_2$ and $r_2$ are chosen uniformly and independently from $\mathbb{Z}_q$(i.e. $t_1 \in_R \mathbb{G}_q$ and $t_{12} \in_R \mathbb{G}_q$), $\tilde{t_1}$, $\tilde{t_{12}}$ are independent from the qualified assertion $\varphi_1$. Since the logarithmic relationships for the pairs $(\gamma', \gamma''_1)$ and $(\gamma', \gamma''_2)$ are unknown to attacker, $\tilde{t_1}$, $\tilde{t_{12}}$, $\tilde{t_{22}}$ are also independent from the qualified assertion $\varphi_1$.

Therefore, $\tilde{C}$ is also independent from the qualified assertion $\varphi_1$.

Since $c_2$ is chosen uniformly and independently from $\mathbb{Z}_q$, then the distribution of $(c_1, c_2)$ is also independent of the qualified assertion $\varphi_1$.

Since $c_1$ is uniformly distributed and independent from $w$, from equation 3.5.5 we can conclude that $\tilde{R}$ is independent from $\tilde{W}$ and hence $\tilde{R}$ is independent of the assertion $\varphi_1$.

Therefore, the conversation $(c_1, c_2, r_1, r_2)$ is independent of the qualified assertion $\varphi_1$ and hence the algorithm is witness-indistinguishable. □

#### 3.5.4.5 Other NIZK proofs used in section 3.5.1.1.

In this section, we present the NIZK proof algorithms that are required in section 3.5.1.1. Algorithm 7 (resp. Algorithm 8) represents the prover algorithm (resp. verifier algorithm) for generation (resp. verification) of the NIZK proof $P_K\{token : \Gamma' = g^{token}\}$ required in the voting and tallying phase in section 3.5.1.1 for an encrypted vote $(U_i, V_i)$. Algorithm 9 (resp. Algorithm 10) represents the prover algorithm (resp. verifier algorithm) for generation (resp. verification) of the NIZK proof $P_K\{s : (\Gamma_1 = g_1^s) \wedge (\Gamma_2 = g_2^s)\}$ and $P_K\{s.prev\_hash : (\Gamma_3 = g_1^{s.prev\_hash}) \wedge (\Gamma_4 = g_2^{s.prev\_hash})\}$ required in the voting and tallying phase in section

**Algorithm 7:** A prover with identifier $ID$ generates a NIZK proof of knowledge of a secret $x$ such that $(\Gamma' = g^x)$ for known $ID, \Gamma', g$ and the encrypted vote $(U_i, V_i)$.

**Input :** $ID, \Gamma', g, x, U_i, V_i$ such that $(\Gamma' = g^x)$
**Output:** $\eta = P_K\{x\colon (\Gamma' = g^x)\}$
**begin**
  **choose** random $w \in \mathbb{Z}_q$
  **calculate**
  $t_1 = g^w$.
  **calculate**
  $c = H(ID, U_i, V_i, g, \Gamma', t_1)$
  **calculate** $r = w - cx$
  **return** $\eta = (c, r)$
**end**

**Algorithm 8:** Verification of proof $\eta$ generated by Algorithm 7 given $ID, \Gamma', g$ and the encrypted vote $(U_i, V_i)$.

**Input :** $ID, \Gamma', g, U_i, V_i, \eta = (c, r)$
**Output:** success or failure
**begin**
  **calculate**
  $t_1 = g^r \Gamma'^c$
  **calculate**
  $c_1 = H(ID, U_i, V_i, g, \Gamma', t_1)$
  **if** $c_1 = c$ **then**
    **return** success
  **else**
    **return** failure
  **end**
**end**

**Algorithm 9:** A prover with identifier $ID$ generates a NIZK proof of knowledge of a secret $x$ such that $((\Gamma_1 = g_1^x) \wedge (\Gamma_2 = g_2^x))$ for known $ID, \Gamma_1, g_1, \Gamma_2, g_2$.

**Input :** $ID, \Gamma_1, g_1, \Gamma_2, g_2, x$ such that $((\Gamma_1 = g_1^x) \wedge (\Gamma_2 = g_2^x))$
**Output:** $\eta = P_K\{x\colon ((\Gamma_1 = g_1^x) \wedge (\Gamma_2 = g_2^x))\}$
**begin**
  **choose** random $w \in \mathbb{Z}_q$
  **calculate**
  $t_1 = g_1^w, t_2 = g_2^w$.
  **calculate**
  $c = H(ID, g_1, \Gamma_1, g_2, \Gamma_2, t_1, t_2)$
  **calculate** $r = w - cx$
  **return** $\eta = (c, r)$
**end**

**Algorithm 10:** Verification of proof $\eta$ generated by Algorithm 9 given $ID, \Gamma_1, g_1, \Gamma_2, g_2$.

**Input :** $ID, \Gamma_1, g_1, \Gamma_2, g_2, \eta = (c, r)$
**Output:** success or failure
**begin**
  **calculate**
  $t_1 = g_1^r \Gamma_1^c, t_2 = g_2^r \Gamma_2^c$
  **calculate**
  $c_1 = H(ID, g_1, \Gamma_1, g_2, \Gamma_2, t_1, t_2)$
  **if** $c_1 = c$ **then**
    **return** success
  **else**
    **return** failure
  **end**
**end**

3.5.1.1.

### 3.5.5 Comparison

In this section, we'll discuss how our proposed system compares with other DRE-based verifiable e-voting systems. In particular, we compare with Chaum's Votegrity [31], Neff's MarkPledge [101], VoteBox [119], Star-Vote [13], DRE-i [59], vVote [39], and DRE-ip [123]. All of these systems consider voter registration and voter authentication outside of their scope and assume that they are performed securely and correctly. We propose a voter registration and authentication mechanism. All of the above mentioned systems rely on a secure bulletin board. In our proposed system, if the BB is insecure, it will be detected by the public or individual voters during the voting phase or tallying phase. We use either the cloud server or the blockchain (method 1) or both the blockchain and cloud server (method 2) to store the ballots. A comparison of these systems in terms of their underlying security assumptions is given in Table 3.5 (also see DRE-ip [123] for security assumptions of these systems).

We now look at the computational complexity of different DRE-based e-voting systems and compare it with our proposed system. We do not consider Votegrity, MarkPledge, and vVote since they use mixnets and the computational complexity depends on the implementation of those mixnets. Here, we have computed all calculations based on a two-candidate

| system | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| Votegrity | ● | ● | ● | ● | ● | ● | ○ | ● | ● | ● |
| MarkPledge | ● | ● | ● | ● | ● | ● | ○ | ● | ● | ● |
| VoteBox | ● | ● | ● | ● | ● | ● | ○ | ● | ● | ● |
| STAR-Vote | ● | ● | ● | ● | ● | ● | ○ | ● | ● | ● |
| DRE-i | ○ | ● | ● | ● | ● | ● | ● | ○ | ● | ● |
| vVote | ● | ● | ● | ● | ● | ● | ○ | ● | ● | ● |
| DRE-ip | ○ | ● | ● | ● | ● | ● | ○ | ○ | ● | ● |
| Proposed system using cloud server | ○ | ● | ○ | ● | ● | ● | ○ | ○ | ○ | ○ |
| Proposed system using blockchain (method 1) | ○ | ● | ○ | ● | ● | ● | ○ | ○ | ○ | ○ |
| Proposed system using both cloud and blockchain (method 2) | ○ | ● | ○ | ● | ● | ● | ○ | ○ | ○ | ○ |

Table 3.5: Security assumptions for some DRE-based verifiable e-voting systems. Columns are represented as - A: Reliable Tallying authorities, B: Sufficient Voter-initiated auditing, C: Protection against malicious bulletin board, D: Secure setup, E: Secure random number generator, F: Secure Deletion, G: Secure Ballot Storage, H: Trust-worthy tallying authorities, I: Secure computation (with proof of correctness) of the final tally without revealing the results from each DRE machine when multiple DRE machines are used, J: voter registration and authentication. ●: assumption is required, ○: assumption is not required.

election, encryption based on ElGamal cryptosystem and one tallying authority (TA) if present. If the number of TAs increases, the complexity of tally calculations and verification of the systems requiring tallying authorities also increases. We assume that the TA, if present, provides proof of correctness as required by the end-to-end verifiability. We assume that the simultaneous multiple exponentiation (SME) [97] technique is used for optimization. Using this technique, the computation cost of the term $g^x h^y$ is equivalent to around 1.2 exponentiations. The voter authentication process and the voting phase can be performed in pipeline. Table 3.6 summarizes the computational complexity of different systems (also see DRE-ip [123] for performance comparison of these systems).

The ElGamal encryption for a ballot takes around 2 exponentiations. The zero-knowledge proof $P_{WF}\{V_i\}$ takes 3.2 exponentiations to generate and 3.6 exponentiations to verify using the proposed efficient NIZK proof Algorithm 3 (for generation) and Algorithm 4 (for verification) described in the section 3.5.4. Therefore, a ballot creation requires 5.2 exponentiations for both audited and confirmed ballot. The computation complexity of some well-knwon e-voting systems are summerized in Table 3.6. All of these systems consider voter registration and authentication outside of their scope; however, We have introduced a voter

| system | A | B | C | D |
|---|---|---|---|---|
| VoteBox | $6.4\|\mathbb{B}\|e$ | $(6.8\|\mathbb{A}\| + 4.8\|\mathbb{C}\|)e$ | $\|\mathbb{C}\|m + 3e$ | $\|\mathbb{C}\|m + 2.4e$ |
| STAR-Vote | $6.4\|\mathbb{B}\|e$ | $(6.8\|\mathbb{A}\| + 4.8\|\mathbb{C}\|)e$ | $\|\mathbb{C}\|m + 3e$ | $\|\mathbb{C}\|m + 2.4e$ |
| DRE-i | $10.8\|\mathbb{B}\|e$ | $(9.6\|\mathbb{A}\| + 4.8\|\mathbb{C}\|)e$ | | $\|\mathbb{B}\|m + 1e$ |
| DRE-ip | $6.4\|\mathbb{B}\|e$ | $(6.8\|\mathbb{A}\| + 4.8\|\mathbb{C}\|)e$ | | $2\|\mathbb{C}\|m + 2e$ |
| Proposed system | $5.2\|\mathbb{B}\|e$ | $(5.6\|\mathbb{A}\| + 3.6\|\mathbb{C}\|)e$ | $8e$ | $(2\|\mathbb{C}\| + 1)m + 4.8e$ |

Table 3.6: Computation complexity of some DRE-based verifiable e-voting systems assuming two-candidate election. Columns are represented as - A: Ballot calculation, B: Ballot well-formedness and consistency verification, C: Tally calculation, D: Tally verification. $\mathbb{B}, \mathbb{A}, \mathbb{C}$ represent all, audited and confirmed ballots respectively. $e$: exponentiation and $m$: multiplication.

authentication mechanism in the system. This introduces some additional computations in our system. The computation of $g^{token}$ requires one exponentiation. The zero-knowledge proof $P_K\{token\}$ takes 1 exponentiation to generate and 1.2 exponentiations to verify using the NIZK proof Algorithm 7 (for generation) and Algorithm 8 (for verification) described in section 3.5.4.5. Therefore, a ballot creation requires about 7.2 exponentiations for both audited and confirmed ballot. The ballot well-formedness and consistency verification takes about 6.8 exponentiations and 4.8 exponentiations for audited ballot and confirmed ballot respectively. The tally calculation and tally verification require about 8 exponentiations and $(2|\mathbb{C}| + 1)m + 4.8e$ computations respectively, where 'm' and 'e' denote the multiplication and exponentiation respectively.

However, in case of $n(n \geq 2)$ candidates, the zero-knowledge proof $P_{WF}\{V_i\}$ takes $(1.2(n-1)+2)$ $(= (1.2n + .8))$ exponentiations to generate and $1.2(n+1)$ exponentiations to verify using the proposed efficient NIZK proof Algorithm 3 (for generation) and Algorithm 4 (for verification) given in the section 3.5.4. Therefore, in this case, ballot calculation requires $(1.2n + 4.8)$ exponentiations for both an audited and confirmed ballot including the computations introduced due to voter authentication. Table 3.7 summarizes the computational complexity of DRE-ip (without voter authentication) and our proposed system (with voter authentication) in case of $n$ candidates.

| system (multiple candidates) | A | B | C | D |
|---|---|---|---|---|
| DRE-ip | $(2.4n + 1.6)\|\mathbb{B}\|e$ | $((2.4n + 2)\|\mathbb{A}\| + 2.4n\|\mathbb{C}\|)e$ | $8e$ | $2\|\mathbb{C}\|m + 2e$ |
| Proposed system | $(1.2n + 4.8)\|\mathbb{B}\|e$ | $((1.2n + 4.4)\|\mathbb{A}\| + 1.2(n + 2)\|\mathbb{C}\|)e$ | | $(2\|\mathbb{C}\| + 1)m + 4.8e$ |

Table 3.7: Computation complexity of DRE-ip (without voter authentication) and our proposed e-voting systems (with voter authentication) while supporting for 1 out of $n$ ($n \geq 3$). Columns are represented as - A: Ballot calculation, B: Well-formedness and consistency verification, C: Tally calculation, D: Tally verification. $\mathbb{B}, \mathbb{A}, \mathbb{C}$ represent all, audited and confirmed ballots respectively. $e$: exponentiation and $m$: multiplication.

### 3.5.6 Performance analysis

#### 3.5.6.1 Experiment on Ethereum (only for method 1).

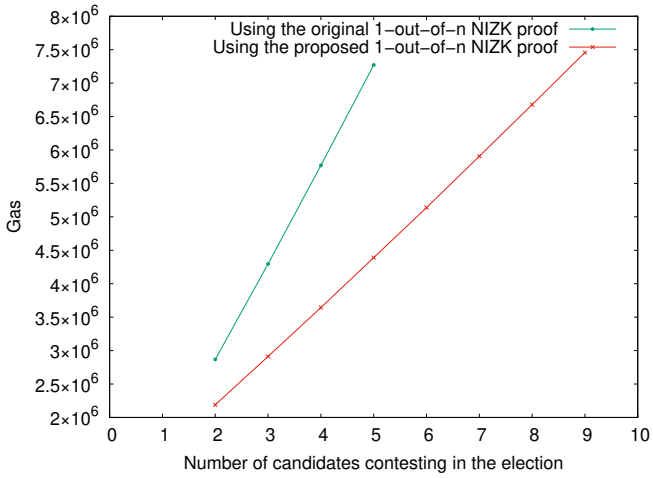


Figure 3.4: Gas cost for casting a ballot based on the number of candidates contesting in the election while using the original 1-out-of-n NIZK proof and our proposed 1-out-of-n NIZK proof.

We deployed our implementation on Ethereum's private network that mimics the production network. The private network was built based on geth 1.9.0 and Ethereum Wallet 0.8.10. We have developed smart contract in Solidity language. We have tested our implementation for different number of candidates contesting in the election. We have performed experiments with 2, 3, 4, 5, 6, 7, 8 and 9 candidates using our proposed efficient 1-out-of-n NIZK ballot well-formedness proof (Algorithm 3 and Algorithm 4 described in section 3.5.4) and plotted the results to show how the costs for casting a ballot vary with different number

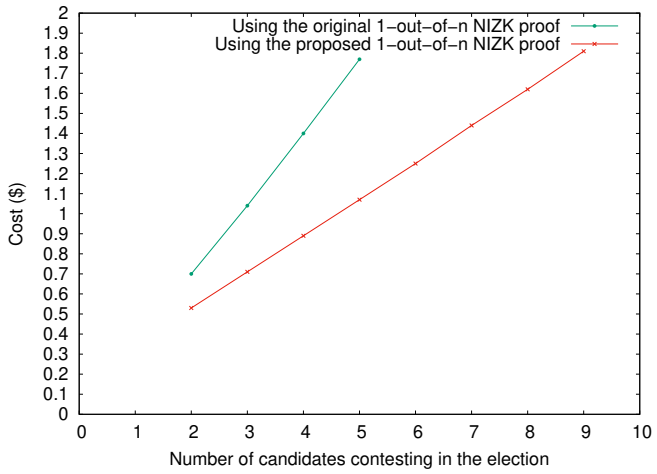


Figure 3.5: Costs for casting a ballot based on the number of candidates contesting in the election while using the original 1-out-of-n NIZK proof and our proposed 1-out-of-n NIZK proof. The costs are approximated in USD ($) using the conversion rate of 1 Ether=$243 and the gas price of 0.000000001 ether that are real world costs in July, 2020.

of candidates. We have also tested with 2, 3, 4, 5 candidates using the original 1-out-of-n NIZK ballot well-formedness proof (Algorithm 1 and Algorithm 2 described in section 3.5.4) and plotted the results to compare the performance. Figure 3.4 depicts the average gas consumption cost for casting a ballot based on the number of candidates competing in the election and while using the original 1-out-of-n NIZK ballot well-formedness proof and our proposed efficient 1-out-of-n NIZK ballot well-formedness proof. Casting a ballot involve verifying the 1-out-of-n NIZK ballot well-formedness proof and storing the ballot into the blockchain. From this figure, we see that the proposed 1-out-of-n NIZK proof is about twice more efficient than the original 1-out-of-n NIZK proof used in [123]. The maximum gas capacity that an Ethereum block can consume is about 8 million as in July, 2020. From the figure, we see that each transaction for casting a ballot reaches the computation and storage limit for about 9 candidates while using our proposed efficient 1-out-of-n NIZK proof; whereas, it reaches the maximum gas limit for about 5 candidates while using the original 1-out-of-n NIZK proof. Figure 3.5 shows the costs for casting a ballot based on the number of candidates competing in the election and while using the original 1-out-of-n NIZK ballot well-formedness proof and our proposed efficient 1-out-of-n NIZK ballot well-formedness proof. We have calculated the costs in US dollar (denoted by '$' in the figure) and rounded it to two decimal places.

#### 3.5.6.2 Timing analysis (in case of using cloud server, method 1 and method 2 ).

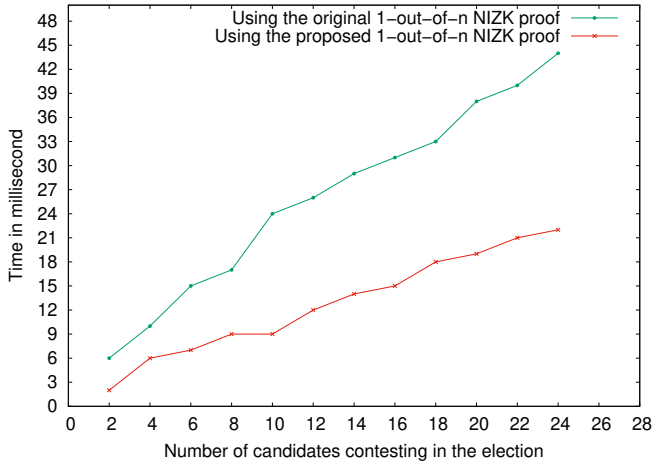


Figure 3.6: Computation time to create the 1-out-of-n NIZK proof using the proposed algorithm and the original NIZK algorithm.

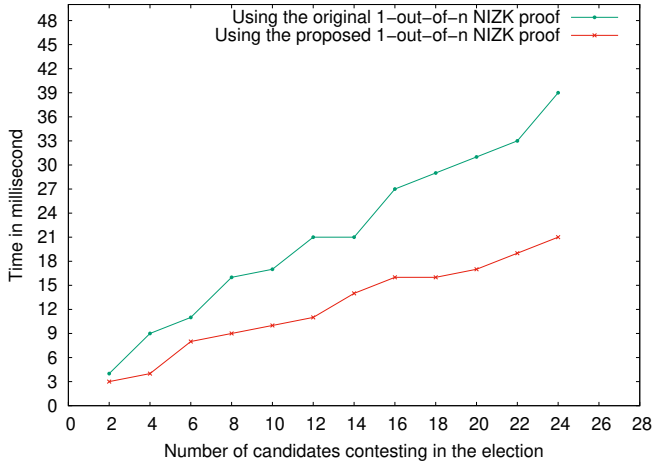


Figure 3.7: Computation time for verification of the 1-out-of-n NIZK proof using the proposed algorithm and the original NIZK algorithm.

We implemented the proposed 1-out-of-n NIZK ballot well-formedness proof which is the most time consuming part for generation and verification of a ballot. Figure 3.6 depicts the timing analysis measurements for generation of 1-out-of-n NIZK ballot well-formedness proof using our proposed NIZK proof (Algorithm 3 described in section 3.5.4) as well as using the original NIZK proof (Algorithm 1 described in section 3.5.4), where $n$ is the number of candidates contesting in the election. Figure 3.7 shows the timing measurement analysis for

verification of 1-out-of-n NIZK ballot well-formedness proof using the proposed NIZK proof (Algorithm 4 described in section 3.5.4) as well as using the original NIZK proof (Algorithm 2 described in section 3.5.4). All tests were performed on a HP Laptop running Windows 8.1 equipped with 2 cores, 1.8 GHz Intel Core i3 and 4 GB RAM. All time measurements are rounded up to the next whole millisecond. We implemented the protocol over an elliptic curve. The time to create and verify a ballot depends on the number of candidates competing in the election. However, it is independent of the number of voters.

To see how the time for generation and verification of the 1-out-of-n NIZK proof using the proposed NIZK proof vary with different number of candidates, we have carried out experiments with 2, 4, 6,..., 24 candidates. Figure 3.6 (resp. 3.7) highlights that the computation time to create (resp. verify) the 1-out-of-n NIZK ballot well-formedness proof using the proposed algorithm is almost reduced to half of the time required to create (resp. verify) that using the original NIZK proof algorithm given in [123].

## 3.6 Concluding Remarks

In this article, we have proposed a secure and verifiable voter registration and authentication mechanism. Thereafter, we have proposed an end-to-end verifiable DRE-based voting system that preserves voter's privacy and integrity of ballots without any tallying authority or secure hardware storage even if the adversary gets temporary access to the DRE machine.

The system prevents the well-known ballot stuffing attack and a weakness of the DRE-ip system. Depending on how the election is arranged, we have proposed two methods to store the ballots using blockchain and cloud server. We have presented security proofs to prove the security properties of the protocol. We have proposed an efficient 1-out-of-n NIZK proof. Both the theoretic analysis and the experimental results show that the scheme is feasible to be used in practice. In future work, we plan to design DRE-based voting solution without tallying authorities for more complex voting systems such as STV and Condorcet.

# Chapter 4

# A Smart Contract System for Decentralized Borda Count Voting

## 4.1 Introduction

This chapter is based on paper [106]. Our contributions in this chapter include the following. We propose the first self-tallying decentralized Borda count voting protocol. The proposed protocol provides maximum voter privacy: an individual vote can only be revealed by a full-collusion attack that involves all other voters. All voting data is publicly available, and the correct execution of the protocol can be verified by any public observer. The proposed protocol does not require any trusted authority to compute the tally; the tally can be computed by each voter, as well as by any observer of the election. We provide security proofs to prove the security of the proposed scheme. In particular, we show that the proposed scheme guarantees the maximum voter privacy against colluding voters. We provide an implementation of the proposed protocol over Ethereum Blockchain. It is a boardroom-scale voting system implemented as a smart contract in Ethereum. Our implementation demonstrates the feasibility of using Ethereum for secure Borda count voting with public verifiability.

## 4.2 Preliminaries

In this section, we describe the security definitions that our proposed protocol is expected to satisfy. We also state the assumptions based on which we prove these security properties.

### 4.2.1 Desirable properties

In a decentralised voting system, some voters may collude with each other to breach other voters' privacy or manipulate the voting outcome. A full collusion against a particular voter occurs when all other voters involve in the collusion. No decentralized system can preserve an honest voter's privacy in case of full collusion since the honest voter's vote can be obtained by subtracting the colluding voters' votes from the final tally. Therefore, we only consider partial collusion which involves some voters, but not all.

Under the threat model of partial collusion, the following three properties (also see [82, 57]) should be fulfilled by a decentralized voting protocol.

1. *Maximum ballot secrecy:* This is an extension of the usual ballot secrecy requirement. In a voting system with maximum ballot secrecy, an attacker who colludes with a group of voters will only learn the partial tally of the remaining voters, but nothing beyond that.

2. *Self-tallying:* During the tallying phase, the final tally can be computed by anyone including voters and third-party observers without external help. This is naturally expected in a decentralized voting protocol.

3. *Dispute-freeness:* A voting scheme is dispute-free if every observer of the election can verify the fact that every voter follows the protocol honestly. This requirement means that the result should be publicly verifiable.

In a self-tallying voting protocol, the last voter can compute the tally before casting her vote. This leads to two issues. First, the last voter can use the knowledge of the tally to decide how she will cast her vote. This issue can potentially influence the result of the election. To prevent this issue, Kiayias and Yung [82] and Groth [57] suggest that an election authority can cast the last vote. During the tallying phase, this last vote is excluded from the final tally. We can apply this method in our implementation, however, in this case, the election authority needs to be trusted not to collude with the last voter. Therefore, this method relies on the trusted election authority. Instead, McCorry et al. [96] propose an extra commitment round to address this issue. In this round, every voter stores the hash of

their encrypted vote in the blockchain as a commitment. The last voter will still be able to compute the tally before casting her vote, however, she will not be able to change her vote. We follow the same approach.

The second issue that since the last voter knows the tally before casting her vote, she may refrain from casting her vote if she is dissatisfied with the result of the election. In that case, no one will be able to compute the final tally, and the election will need to be restarted. To circumvent this issue, Kiayias and Yung [82] and Khader et al. [79] propose an additional round engaging the rest of the voters. However, in this case, we must assume the remaining voters do not drop out of the election half-way; otherwise, the election will be aborted again. To address this issue, we use Ethereum's blockchain and smart contract to enforce a financial incentive for all voters using a deposit and refund paradigm as done in [96]. In our implementation, it is mandatory for all voters to deposit some money into the smart contract to register for an election. This deposit is refunded automatically to the voter once her vote is successfully accepted by the blockchain. A voter who registers for an election but withholds her vote simply loses the deposit. This provides a countermeasure to this abortive issue. The confiscated deposit could be used as a compensation for all compliant voters or be donated to a charity.

### 4.2.2 Cryptographic assumption

We state the cryptographic assumption that we use to prove the security properties of our proposed protocol. If this assumption holds, the proposed protocol satisfies the above security properties. We first describe some notations that we use throughout our paper.

*Notation:* We follow the notation introduced by Camenisch and Stadler [23]. We use $P_K\{\lambda : \Gamma = \gamma^\lambda\}$ to denote a non-interactive proof of knowledge of a secret $\lambda$ such that $\Gamma = \gamma^\lambda$ for publicly known $\Gamma$ and $\gamma$. We shorten the notation to $P_K\{\lambda\}$ if the context is clear.

*Cryptographic setup:* Our system works on a DSA like multiplicative cyclic group setting or an ECDSA-like group setting over an elliptic curve, where the decision Diffie-Hellman problem is assumed to be intractable [42]. Let $\mathbb{G}_q$ be a subgroup of $Z_p^*$ of prime order $q$, where $q \mid p-1$. Let $g$ be a generator of that group.

The decision Diffie-Hellman assumption [42] is defined as follows:

*Assumption 1: (DDH)* If $\alpha, \beta$ are randomly and uniformly chosen from $\mathbb{Z}_q^*$, given $(g, g^\alpha, g^\beta, \rho)$ where $\rho \in \{g^{\alpha\beta}, R\}$ and $R$ is randomly and independently chosen from $\mathbb{G}_q$, it is hard to decide whether $\rho = g^{\alpha\beta}$ or $\rho = R$.

## 4.3 Our scheme

The Borda count scheme is a ranked choice voting method in which voters rank candidates in order of preference. A score is associated with each rank. Let there are $k$ candidates competing in an election. Assume a score $a_j$ is associated with the $j$-th rank and $a_{j-1} > a_j, \forall j \in \{2, ..., k\}$, i.e., a higher score implies a higher rank of the candidate. At the end of the election, the scores obtained by a candidate are added together. Finally, the candidates are ranked according to their scores obtained. The candidate with the highest score wins the election. In this scheme, a participant's vote will be of the form $(v_1, v_2, ..., v_k)$, where $(v_1, v_2, ..., v_k)$ is a permutation of $(a_1, a_2, ...., a_k)$.

In our protocol, we assume that there is an authenticated public channel available for each participant. This assumption is common in previous e-voting protocols; see, for example, Kiayias and Yung's protocol [82], Groth's protocols [57], the open vote network [60], and general multi-party secure computation protocols [52, 53]. This authenticated public channel can be realized by using physical means or a public bulletin board where recorded ballots are stored securely in an append only manner [96].

Our protocol is inspired by Open Vote Network [60] but we have adapted the protocol to support Borda count. This results in a new e-voting protocol, which is also the first ranked-choice voting system based on Borda count in a decentralized setting.

Our protocol works in two rounds. Assume that there are $n$ participants in an election. We denote the $i$-th participant as $V_i$. They all agree on public group parameters $(\mathbb{G}_q, g)$. Let there are $k$ candidates competing in the election. Let us assume that the score corresponding to the $j$-th rank is $a_j$, where $a_{j-1} > a_j; \forall j \in \{1, 2, ..., k\}$.

### 4.3.1 Voting Phase

Each participant $V_i$ generates $k$ random values $(x_{i1}, x_{i2}, ..., x_{ik})$ as their secrets, where $x_{ij} \in_R \mathbb{Z}_q; \forall j \in \{1, 2, ..., k\}$. Each participant executes the following two-round protocol. We denote the vote cast by the $i$-th participant $V_i$ as $v_i = (v_{i1}, v_{i2}, ..., v_{ik})$, where $(v_{i1}, v_{i2}, ..., v_{ik})$ is a permutation of $(a_1, a_2, ..., a_k)$. Here, $v_{ij}$ is the score given by the participant $V_i$ to the $j$-th candidate.

**First round.** Every participant $V_i$ calculates $X_{i1} = g^{x_{i1}}, X_{i2} = g^{x_{i2}}, ..., X_{ik} = g^{x_{ik}}$ and publishes $(X_{i1}, P_K\{x_{i1}\}, X_{i2}, P_K\{x_{i2}\}, \ldots, X_{ik}, P_K\{x_{ik}\})$, where $P_K\{x_{ij}\}$ is the non-interactive zero-knowledge (NIZK) proof for $x_{ij}; \forall j \in \{1, 2, ..., k\}$. The NIZK proofs are generated by using Schnorr's signature [122] (see section 4.4 for more details).

At the end of this round, every participant verifies the validity of all zero-knowledge proofs. Each participant $V_i$ then computes $g^{y_{ij}} = \prod_{l=1}^{i-1} g^{x_{lj}} / \prod_{l=i+1}^{n} g^{x_{lj}}; \forall j \in \{1, 2, \ldots, k\}$.

**Second round.** Each participant $V_i$ calculates $Z_{ij} = \{g^{y_{ij}}\}^{x_{ij}} g^{v_{ij}}; \forall j \in \{1, 2, ..., k\}$ and generates a non-interactive zero-knowledge (NIZK) proof to prove the well-formedness of the ballot. The NIZK associated with each participant's ballot proves that $(v_{i1}, v_{i2}, ..., v_{ik})$ is a permutation of $(a_1, a_2, .., a_k)$. In order to prove the statement, it is sufficient to prove the following $k$ relations.

- $a_1 \in \{v_{i1}, v_{i2}, , ..., v_{ik}\}$ for the score corresponding to the 1-st rank.
- $a_2 \in \{v_{i1}, v_{i2}, , ..., v_{ik}\}$ for the score corresponding to the 2-nd rank.
- ...
- $a_k \in \{v_{i1}, v_{i2}, , ..., v_{ik}\}$ for the score corresponding to the $k$-th rank.

The above $k$ relations hold true if and only if the following relations are true:

$\vee_{j=1}^{k}((X_{ij} = g^{x_{ij}}) \wedge (Z_{ij} = \{g^{y_{ij}}\}^{x_{ij}} g^{a})); \forall a \in \{a_1, a_2, ..., a_k\}$. We denote this NIZK proof as $\Pi_j[x_{i1}, x_{i2}, ..., x_{ik} : X_i, Z_i]; \forall j \in \{1, 2, .., k\}$, where $X_i = (X_{i1}, X_{i2}, ..., X_{ik}), Z_i = (Z_{i1}, Z_{i2}, ..., Z_{ik})$. More details about the NIZK proof can be found in section 4.4.

Each participant $V_i$ publishes $(Z_{i1}, Z_{i2}, ..., Z_{ik})$ and $k$ NIZK proofs $\Pi_j[x_{i1}, x_{i2}, ..., x_{ik} : X_i, Z_i]; \forall j \in \{1, 2, ..., k\}$.

### 4.3.2 Tallying phase

Anyone can compute $\prod_{i=1}^{n}\{g^{y_{ij}}\}^{x_{ij}}g^{v_{ij}} = g^{\Sigma_{i=1}^{n}v_{ij}}$, where $j \in \{1, 2, ..., k\}$. This equality follows from the fact that $\Sigma_{i=1}^{n}x_{ij}y_{ij} = 0; \forall j \in \{1, 2, ..., k\}$ (proposition 1, see also [62]).

Each participant, as well as observers of the election, can check the validity of all the NIZK proofs to ensure that no badly formed vote has been cast to distort the tally.

The total score obtained by the $j$-th candidate is $\Sigma_{i=1}^{n}v_{ij}$, which is normally a small number for all $j \in \{1, 2, ..., k\}$. Since the quantity $\Sigma_{i=1}^{n}v_{ij}$ is a small number, the discrete logarithm of $g^{\Sigma_{i=1}^{n}v_{ij}}$ can be computed by exhaustive search or Shanks' baby-step giant-step algorithm [89]. Finally, each participant and all observers of the protocol can rank candidates in order of their total scores.

*Proposition 1:* For $x_{ij}$ and $y_{ij}$ as defined above $\Sigma_{i=1}^{n}x_{ij}y_{ij} = 0; \forall j \in \{1, 2, ..., k\}$.

*Proof:* According to the protocol, $y_{ij} = \Sigma_{l<i}x_{lj} - \Sigma_{l>i}x_{lj}; \forall j \in \{1, 2, ..., k\}$.

Now for any fix $j \in \{1, 2, ..., k\}$, $\Sigma_{i=1}^{n}x_{ij}y_{ij} = \Sigma_{i=1}^{n}x_{ij}(\Sigma_{l<i}x_{lj} - \Sigma_{l>i}x_{lj}) = 0$. □

The use of NIZK proofs in the protocol is to ensure that all participants follow the protocol faithfully. In the first round, each participant posts $k$ NIZK proofs to prove her knowledge of the exponents $(x_{i1}, x_{i2}, ..., x_{ik})$. The Fiat-Shamir heuristic is employed in our protocol to construct NIZK proofs [48]. Consequently, our NIZK proofs are in the Random Oracle Model [8]. The algorithm 11 and algorithm 12 provided in section 4.4 describe the procedure to create and verify these NIZK proofs respectively.

In the second round, each participant posts $k$ NIZK proofs to prove that her encrypted vote is a permutation of $(a_1, a_2, ..., a_k)$ without reveling which permutation. In order to prove this, first note that the terms of our protocol form exponential ElGamal encryptions of $v_{ij}$, where $i \in \{1, 2, ..., n\}$ and $j \in \{1, 2, ..., k\}$. This can be realized by treating $g^{y_{ij}}$ as the public key and using the published terms of 1-st round. Thus, we form

$(g^{x_{ij}}, \{g^{y_{ij}}\}^{x_{ij}}g^{v_{ij}})$, where $i \in \{1, 2, ..., n\}$ and $j \in \{1, 2, ..., k\}$.

This will be an ElGamal encryption of $g^{v_{ij}}; \forall j \in \{1, 2, ..., k\}$ with public key $g^{y_{ij}}$ and randomization $x_{ij}; \forall j \in \{1, 2, ..., k\}$. Thus, the published terms for the first and second rounds form an exponential ElGamal encryption of $v_{ij}$, $\forall i \in \{1, 2, ..., n\}$ and $\forall j \in \{1, 2, ..., k\}$. We use an efficient NIZK proof technique proposed by Cramer, Damgård, and Schoenmakers

[37] to construct proofs of conjunctive knowledge, disjunctive knowledge and combination of both. This essentially proves that $(v_{i1}, v_{i2}, ..., v_{ik})$ is a permutation of $(a_1, a_2, ..., a_k)$ without revealing which permutation (i.e. which message corresponds to which ciphertext). Note that we do not need to decrypt these ciphertexts in order to obtain the tally. The algorithm 13 and algorithm 14 provided in section 4.4 describe the procedure to create and verify these 1-out-of-k NIZK proofs respectively.

## 4.4 The NIZK proof algorithms used in the proposed Borda count protocol

**Algorithm 11:** A prover with identifier $ID$ generates a NIZK proof of knowledge of a secret $x$ such that $(\Gamma' = g^x)$ for known $ID, \Gamma', g$.

**Input :** $ID, \Gamma', g, x$ such that $(\Gamma' = g^x)$
**Output:** $\eta = P_K\{x : (\Gamma' = g^x)\}$
**begin**
  **choose** random $w \in \mathbb{Z}_q$
  **calculate**
  $t_1 = g^w$.
  **calculate**
  $c = H(ID, g, \Gamma', t_1)$
  **calculate** $r = w - cx$
  **return** $\eta = (r, t_1)$
**end**

**Algorithm 12:** Verification of proof $\eta$ generated by *Algorithm 1* given $ID, \Gamma', g$.

**Input :** $ID, \Gamma', g, \eta = (r, t_1)$
**Output:** success or failure
**begin**
  **calculate**
  $c = H(ID, g, \Gamma', t_1)$
  **calculate**
  $t_1' = g^r \Gamma'^c$
  **if** $t_1 = t_1'$ **then**
    **return** success
  **else**
    **return** failure
  **end**
**end**

**Algorithm 13:** A prover with identifier $ID$ generates a proof of knowledge of a secret $x_{ij}$ such that $\vee_{l=1}^{k}((\Gamma'_l = g^{x_{il}}) \wedge (\Gamma''_l / g^{v_m} = \{g^{y_{il}}\}^{x_{il}}))$, where $v_m$ is the score associated to the $m$-th rank, $m \in \{1, 2, ..., k\}$, and the score $v_m$ is given to the $j$-th candidate, $j \in \{1, 2, ..., k\}$.

**Input :** $ID, g, k, (\Gamma'_l, \Gamma''_l, \{g^{y_{il}}\})_{l=1}^{k}, x_{ij}, j, g^{v_m}$ such that $\Gamma'_j = g^{x_{ij}}$ and $\Gamma''_j / g^{v_m} = \{g^{y_{il}}\}^{x_{ij}}$

**Output:** $\Pi_m = P_K\{x_{ij} : \vee_{l=1}^{k}((\Gamma'_l = g^{x_{il}}) \wedge (\Gamma''_l / g^{v_m} = \{g^{y_{il}}\}^{x_{il}}))\}$

**begin**

**choose** random $w, r_1, c_1, r_2, c_2, ..., r_{j-1}, c_{j-1}, r_{j+1}, c_{j+1}, ..., r_k, c_k \in \mathbb{Z}_q$

**calculate** $t_{11} = g^{r_1}\{\Gamma'_1\}^{c_1}, t_{12} = \{g^{y_{i1}}\}^{r_1}\{\Gamma''_1 / g^{v_m}\}^{c_1}, t_{21} = g^{r_2}\{\Gamma'_2\}^{c_2}, t_{22} = \{g^{y_{i2}}\}^{r_2}\{\Gamma''_2 / g^{v_m}\}^{c_2}, ...., t_{j-11} = g^{r_{j-1}}\{\Gamma'_{j-1}\}^{c_{j-1}}, t_{j-12} = \{g^{y_{ij-1}}\}^{r_{j-1}}\{\Gamma''_{j-1} / g^{v_m}\}^{c_{j-1}}, t_{j1} = g^{w}, t_{j2} = \{g^{y_{ij}}\}^{w}, t_{j+11} = g^{r_{j+1}}\{\Gamma'_{j+1}\}^{c_{j+1}}, t_{j+12} = \{g^{y_{ij+1}}\}^{r_{j+1}}\{\Gamma''_{j+1} / g^{v_m}\}^{c_{j+1}}, ..., t_{k1} = g^{r_k}\{\Gamma'_k\}^{c_k}, t_{k2} = \{g^{y_{ik}}\}^{r_k}\{\Gamma''_k / g^{v_m}\}^{c_k}$

**calculate**

$c = H(ID, (g, \Gamma'_l, \{g^{y_{il}}\}, \{\Gamma''_l / g^{v_m}\})_{l=1}^{k}, (t_{l1}, t_{l2})_{l=1}^{k})$,

**calculate**

$c_j = c - (c_1 + c_2 + .... + c_{j-1} + c_{j+1} + ... + c_k)$

**calculate** $r_j = w - c_j x_{ij}$

**return** $\Pi_m = (c_1, c_2, ...., c_{j-1}, c_j, c_{j+1}, ..., c_k, r_1, r_2, ..., r_{j-1}, r_j, r_{j+1}, ..., r_k, (t_{l1}, t_{l2})_{l=1}^{k})$

**end**

In this section, we present the NIZK proof algorithms that are required in the first and second round of the protocol. Algorithm 11 (resp. Algorithm 12) represents the prover algorithm (resp. verifier algorithm) for generation (resp. verification) of the NIZK proof required in the first round of the protocol. Let us assume that there are $k$ candidates contesting in the election. Algorithm 13 (resp. Algorithm 14) represents the prover algorithm (resp. verifier algorithm) for generation (resp. verification) of 1-out-of-k zero-knowledge proof required in the second round of the protocol. Algorithm 13 (resp. Algorithm 14) is written for the $i$-th voter $V_i$ to prove (resp. verify) a proposition of the form $\vee_{l=1}^{k}((\Gamma'_l = g^{x_{il}}) \wedge (\Gamma''_l / g^{v_m} = \{g^{y_{il}}\}^{x_{il}}))$, where $v_m$ is the score associated to the $m$-th rank, $m \in \{1, 2, ..., k\}$, the score $v_m$ is given to the $j$-th candidate, $j \in \{1, 2, ..., k\}$, $\Gamma'_l$ represents $X_{il}$ corresponding to the $l$-th candidate in the first round of the protocol, and $\Gamma''_l$ represents $Z_{il}$ corresponding to the $l$-th candidate in the second round of the protocol as discussed in section 4.3.1. The algorithm for the case when the score $v_m$ is given to any candidate other than the $j$-th candidate can be obtained by straightforward modifications. The symbol 'ID' denotes the

**Algorithm 14:** Verification of proof $\Pi_m$ generated by *Algorithm 3* given $ID, g, k, (\Gamma'_l, \Gamma''_l, \{g^{y_{il}}\})_{l=1}^{k}, g^{v_m}$, where $v_m$ is the score associated to the $m$-th rank. However, the verifier does not know to which candidate (i.e. $j$) the score $v_m$ is given.

**Input :** $ID, g, k, (\Gamma'_l, \Gamma''_l, \{g^{y_{il}}\})_{l=1}^{k}, g^{v_m}, \Pi_m = (c_1, c_2, ..., c_k, r_1, r_2, ..., r_k, (t_{l1}, t_{l2})_{l=1}^{k})$
**Output:** success or failure
**begin**
  **calculate**
  $c' = H(ID, (g, \Gamma'_l, \{g^{y_{il}}\}, \{\Gamma''_l / g^{v_m}\})_{l=1}^{k}, (t_{l1}, t_{l2})_{l=1}^{k})$
  **if** $(c' \neq (c_1 + c_2 + ... + c_k))$ **then**
    **return** failure
  **end**

  **calculate**
  $t'_{11} = g^{r_1}\{\Gamma'_1\}^{c_1}, t'_{12} = \{g^{y_{i1}}\}^{r_1}\{\Gamma''_1 / g^{v_m}\}^{c_1}, t'_{21} = g^{r_2}\{\Gamma'_2\}^{c_2}, t'_{22} =$
  $\{g^{y_{i2}}\}^{r_2}\{\Gamma''_2 / g^{v_m}\}^{c_2}, ...,$
  $t'_{k1} = g^{r_k}\{\Gamma'_k\}^{c_k}, t'_{k2} = \{g^{y_{ik}}\}^{r_k}\{\Gamma''_k / g^{v_m}\}^{c_k}$
  **if** $((t_{11} = t'_{11})\&\&(t_{12} = t'_{12})\&\&(t_{21} = t'_{21})\&\&$
  $(t_{22} = t'_{22})\&\&...\&\&(t_{k1} = t'_{k1})\&\&(t_{k2} = t'_{k2}))$ **then**
    **return** success
  **else**
    **return** failure
  **end**

**end**

publicly known identifier of the voter. Following [62, 60], we include the voter's ID in the hash for the Fiat-Shamir transformation to bind the identity with the ZKP and to prevent replay attacks. For example, in our implementation over Ethereum, we use the sender's unique identity (msg.sender) as 'ID' in the argument of the hash function. The purpose of using the sender's unique identity (msg.sender) is already discussed in section 4.7.4. The symbols $\Gamma'$ and $x$ in the Algorithm 11 and Algorithm 12 represent $X_{ij}$ and $x_{ij}$ respectively for each of the $i$-th voter $V_i$ in the first round of the protocol as described in section 4.3.1, where $j \in \{1, 2, ..., k\}$.

## 4.5 Security analysis

In this section, we prove that our scheme is secure against all probabilistic polynomial adversaries who try to deduce the vote given by a voter. We also show that the public

bulletin board does not reveal any additional information regarding the voter's privacy other than the tally. In case of partial collusion, we prove that if an attacker colludes with some $m$ number of voters, then she will learn the partial tally of the $n - m$ voters, but nothing beyond that. The partial tally of the honest voter's vote can be obtained by subtracting the colluding voter's vote from the final tally. Therefore, colluding voters can always compute the partial tally of votes of the remaining voters. The security of our scheme relies on the intractability of the Decisional Diffie-Hellman (DDH) problem. We prove that if the DDH problem is intractable in the group $\mathbb{G}_q$, then this scheme is secure.

In the following sections, we show that our protocol satisfies the three security requirements mentioned in section 4.2.1.

### 4.5.1 Maximum ballot secrecy

In this section, we show that our protocol is secure under the threat model of partial collusion. In the protocol, each participant $V_i$ sends $k$ ephemeral public keys $(g^{x_{i1}}, g^{x_{i2}}, ..., g^{x_{ik}})$ along with $k$ NIZK proofs of exponents in the first round and sends an encrypted ballot $(g^{y_{i1}x_{i1}}g^{v_{i1}}, g^{y_{i1}x_{i2}}g^{v_{i2}}, ..., g^{y_{i1}x_{ik}}g^{v_{ik}})$ with $k$ 1-out-of-k NIZK proofs in the second round. The $k$ 1-out-of-k NIZK proofs in the second round ensure that $(v_{i1}, v_{i2}, ..., v_{ik})$ is a permutation of $(a_1, a_2, ..., a_k)$. In this protocol, the value of $y_{ij}$ depends on the values of secret keys $x_{ij}$ of all voters except $V_i$, where $j \in \{1, 2, ..., k\}$. We now discuss the security properties of $y_{ij}$. Let us consider any participant $V_i$, where $i \in \{1, 2, ..., n\}$. The following properties hold true for any participant $V_i$.

**Lemma 4.5.1.** *Considering the threat model of partial collusion against a participant $V_i$, the $y_{ij}$ is a secret random value in $\mathbb{Z}_q$ to attackers for all $j \in \{1, 2, ..., k\}$.*

*Proof.* Let us first fix any $j \in \{1, 2, ..., k\}$. Since we are considering the partial collusion against $V_i$, there exists at least one other participant $V_m$ $(m \neq i)$ who is not involved in the collusion. Hence, the secret random value $x_{mj}$ generated by the participant $V_m$ is unknown to the colluders. According to the protocol, $x_{mj}$ is uniformly distributed over $\mathbb{Z}_q$ and known only to the participant $V_m$. Note that for a fixed $j$, $y_{ij}$ is computed from all $x_{ij}$ known to colluders (in the worst case) and a random number $x_{mj}$ unknown to the colluders. Therefore,

$y_{ij}$ is uniformly distributed over $\mathbb{Z}_q$ and unknown to the colluders. Since the above discussion holds true for any $j \in \{1, 2, ..., k\}$, $y_{ij}$ is a secret random value in $\mathbb{Z}_q$ to attackers for all $j \in \{1, 2, ..., k\}$ even in the worst case. □

**Lemma 4.5.2.** *If the assumption 1 (DDH) holds in the group $\mathbb{G}_q$, then given $g, g^{\alpha_1}, g^{\alpha_2}, ..., g^{\alpha_m}$ and $g^{\beta_1}, g^{\beta_2}, ..., g^{\beta_m} \in \mathbb{G}_q$, and a challenge $\rho \in \{\rho_1, \rho_2\}$, where $\rho_1 = (g^{\alpha_1\beta_1}, g^{\alpha_2\beta_2}, ..., g^{\alpha_m\beta_m})$ and $\rho_2 = (R_1, R_2, ..., R_m), R_i \in \mathbb{G}_q, \forall i \in \{1, 2, ..., m\}$, it is hard to decide whether $\rho = \rho_1$ or $\rho = \rho_2$.*

*Proof.* We prove the lemma by showing that if there exists a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ that can decide whether $\rho = \rho_1$ or $\rho = \rho_2$, then we can use the same to construct another adversary $\mathcal{B}$ against *assumption 1.* We construct the adversary $\mathcal{B}$ as follows. The inputs to the adversary $\mathcal{B}$ are $g, g^{\alpha}, g^{\beta}$. The adversary $\mathcal{B}$ receives the challenge $\rho = \{g^{\alpha\beta}, R\}$, where $R \in_R \mathbb{G}_q$. It chooses $\gamma_1, \gamma_2, ..., \gamma_m$ and $\eta_1, \eta_2, ..., \eta_m$ uniformly at random from $\mathbb{Z}_q$. It computes $\delta_i = \rho^{\gamma_i\eta_i}, \forall i \in \{1, 2, ..., m\}$. $\mathcal{B}$ calculates $g^{\beta_i} = g^{\beta\gamma_i}, \forall i \in \{1, 2, ..., m\}$. $\mathcal{B}$ also calculates $g^{\alpha_i} = g^{\alpha\eta_i}, \forall i \in \{1, 2, ..., m\}$. Note that after this step, $\alpha_i = \alpha\eta_i$ and $\beta_i = \beta\gamma_i, \forall i \in \{1, 2, ..., m\}$. Now the adversary $\mathcal{B}$ sends $(g^{\alpha_1}, g^{\alpha_2}, ..., g^{\alpha_m}), (g^{\beta_1}, g^{\beta_2}, ..., g^{\beta_m})$ and $(\delta_1, \delta_2, ..., \delta_m)$ to $\mathcal{A}$. Now if $\rho = g^{\alpha\beta}$, then $(\delta_1, \delta_2, ..., \delta_m) = (\rho^{\gamma_1\eta_1}, \rho^{\gamma_2\eta_2}, ..., \rho^{\gamma_m\eta_m}) = (\{g^{\alpha\beta}\}^{\gamma_1\eta_1}, \{g^{\alpha\beta}\}^{\gamma_2\eta_2}, ..., \{g^{\alpha\beta}\}^{\gamma_m\eta_m}) = (\{g^{\alpha\eta_1}\}^{\beta\gamma_1}, \{g^{\alpha\eta_2}\}^{\beta\gamma_2}, ..., \{g^{\alpha\eta_m}\}^{\beta\gamma_m}) = (g^{\alpha_1\beta_1}, g^{\alpha_2\beta_2}, ..., g^{\alpha_m\beta_m})$; otherwise if $\rho = R$, then $(\delta_1, \delta_2, ..., \delta_m) = (R_1, R_2, ..., R_m)$. According to our assumption, upon receiving $(g^{\alpha_1}, g^{\alpha_2}, ..., g^{\alpha_m}), (g^{\beta_1}, g^{\beta_2}, ..., g^{\beta_m})$ and the challenge $(\delta_1, \delta_2, ..., \delta_m)$ from $\mathcal{B}$, $\mathcal{A}$ can distinguish between $(g^{\alpha_1\beta_1}, g^{\alpha_2\beta_2}, ..., g^{\alpha_m\beta_m})$ and $(R_1, R_2, ..., R_m)$. If the adversary $\mathcal{A}$ can distinguish between these two values, $\mathcal{B}$ can also distinguish between $g^{\alpha\beta}$ and $R$. Hence, $\mathcal{B}$ can identify the correct value of $\rho$. Thus, we have constructed an adversary $\mathcal{B}$ against the *assumption 1* (DDH assumption). Hence, the Lemma 4.5.2 is proved. It can be easily verified that $Adv(\mathcal{A}) \leq Adv(\mathcal{B})$. □

**Lemma 4.5.3.** *Given $g \in \mathbb{G}_q, \mathcal{G} = \{g^{x_i}, g^{y_i} : y_i = (\Sigma_{j=1}^{i-1} x_j - \Sigma_{j=i+1}^{\mu} x_j), i \in \{1, 2, ..., \mu\}\}$ and $y_i$ are unknown for all $i \in \{1, 2, ..., \mu\}$. If $\Sigma_{i=1}^{\mu} v_i = \Sigma_{i=1}^{\mu} v_i'$ and $v_i, v_i' \in \mathbb{Z}_q$, then the bulletin boards A and B as given in Table 4.1 are indistinguishable.*

*Proof.* Without loss of generality, we assume that $\Sigma_{i=1}^{\mu} v_i = \Sigma_{i=1}^{\mu} v_i' = v$. Since $y_i =$

| | |
|---|---|
| $g^{y_1x_1}g^{v_1}$ | $g^{y_1x_1}g^{v_1'}$ |
| $g^{y_2x_2}g^{v_2}$ | $g^{y_2x_2}g^{v_2'}$ |
| ... | ... |
| ... | ... |
| ... | ... |
| $g^{y_\mu x_\mu}g^{v_\mu}$ | $g^{y_\mu x_\mu}g^{v_\mu'}$ |
| $A$ | $B$ |

Table 4.1: Indistinguishability of Bulletin Board $A$ and $B$

$\Sigma_{j=1}^{i-1}x_j - \Sigma_{j=i+1}^{\mu}x_j$ for all $i \in \{1,2,...,\mu\}\}$, we have $\Sigma_{j=1}^{\mu}x_jy_j = 0$. Thus, we may write $g^{x_\mu y_\mu} = (1/g^{\Sigma_{i=1}^{(\mu-1)}x_iy_i})$. Also, since $\Sigma_{i=1}^{\mu}v_i = v$, we may write $g^{v_\mu} = (g^v/g^{\Sigma_{i=1}^{\mu-1}v_i})$. Note that $y_i$'s are unknown for all $i \in \{1,2,3,..,\mu\}$. From Lemma 4.5.2, we can write $A = (g^{y_1x_1}g^{v_1}, g^{y_2x_2}g^{v_2}, \ldots, g^{y_{\mu-1}x_{\mu-1}}g^{v_{\mu-1}}, g^{y_\mu x_\mu}g^{v_\mu}) = (g^{y_1x_1}g^{v_1}, g^{y_2x_2}g^{v_2}, \ldots, g^{y_{\mu-1}x_{\mu-1}}g^{v_{\mu-1}}, g^v/\prod_{i=1}^{\mu-1}g^{y_ix_i}g^{v_i}) \approx_c (R_1 * g^{v_1}, R_2 * g^{v_2}, ..., R_{\mu-1} * g^{v_{\mu-1}}, g^v/\prod_{i=1}^{\mu-1}R_i * g^{v_i}) \approx_c (R_1, R_2, ..., R_{\mu-1}, g^v/\prod_{i=1}^{\mu-1}R_i) \approx_c (R_1 * g^{v_1'}, R_2 * g^{v_2'}, \ldots, R_{\mu-1} * g^{v_{\mu-1}'}, g^v/\prod_{i=1}^{\mu-1}R_i * g^{v_i'}) \approx_c (g^{y_1x_1}g^{v_1'}, g^{y_2x_2}g^{v_2'}, \ldots, g^{y_{\mu-1}x_{\mu-1}}g^{v_{\mu-1}'}, g^v/\prod_{i=1}^{\mu-1}g^{y_ix_i}g^{v_i'}) = (g^{y_1x_1}g^{v_1'}, g^{y_2x_2}g^{v_2'}, \ldots, g^{y_{\mu-1}x_{\mu-1}}g^{v_{\mu-1}'}, g^{y_\mu x_\mu}g^{v_\mu'}) = B$, since $g^{v_\mu'} = (g^v/g^{\Sigma_{i=1}^{\mu-1}v_i'})$. □

**Lemma 4.5.4.** *Let us assume that $R = \{a_1, a_2, ..., a_k\}$, and $\nu = \Sigma_{j=1}^{k}a_j$. Given $g \in \mathbb{G}_q$, $X_i = (X_{i1}, X_{i2}, ..., X_{ik})$ and $Y_i = (Y_{i1}, Y_{i2}, ..., Y_{ik})$, where $X_{ij} = g^{x_{ij}}, Y_{ij} = g^{y_{ij}}, y_{ij} = (\Sigma_{l=1}^{i-1}x_{lj} - \Sigma_{l=i+1}^{\mu}x_{lj})$ and $y_{ij}$ is unknown to the attacker $\forall j \in \{1,2,...,k\}, \forall i \in \{1,2,...,\mu\}, \mu \in \mathbb{N}$, the two bulletin boards $A$ and $B$ as in Table 4.2 and Table 4.3 respectively are indistinguishable, where*

1. $v_{ij}, v_{ij}' \in R, \forall j \in \{1,2,..,k\}, \forall i \in \{1,2,...,\mu\}$
2. $\cup_{j=1}^{k}v_{ij} = \cup_{j=1}^{k}v_{ij}' = R, \forall i \in \{1,2,...,\mu\}$
3. $\Sigma_{j=1}^{k}v_{ij} = \Sigma_{j=1}^{k}v_{ij}' = \nu, \forall i \in \{1,2,...,\mu\}$
4. $\Sigma_{i=1}^{\mu}v_{ij} = \Sigma_{i=1}^{\mu}v_{ij}', \forall j \in \{1,2,...,k\}$

*Proof.* The proof follows from Lemma 4.5.3. □

**Lemma 4.5.5.** *Let us assume that $R = \{a_1, a_2, ..., a_k\}$, and $\nu = \Sigma_{j=1}^{k}a_j$. Given $g \in \mathbb{G}_q$, $X_i = (X_{i1}, X_{i2}, ..., X_{ik})$ and $Y_i = (Y_{i1}, Y_{i2}, ..., Y_{ik})$, where $X_{ij} = g^{x_{ij}}, Y_{ij} = g^{y_{ij}}, y_{ij} =$*

| | | | |
|---|---|---|---|
| $g^{y_{11}x_{11}}g^{v_{11}}$ | $g^{y_{12}x_{12}}g^{v_{12}}$ | ... | $g^{y_{1k}x_{1k}}g^{v_{1k}}$ |
| $g^{y_{21}x_{21}}g^{v_{21}}$ | $g^{y_{22}x_{22}}g^{v_{22}}$ | ... | $g^{y_{2k}x_{2k}}g^{v_{2k}}$ |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| $g^{y_{\mu 1}x_{\mu 1}}g^{v_{\mu 1}}$ | $g^{y_{\mu 2}x_{\mu 2}}g^{v_{\mu 2}}$ | ... | $g^{y_{\mu k}x_{\mu k}}g^{v_{\mu k}}$ |

Table 4.2: Bulletin Board $A$

| | | | |
|---|---|---|---|
| $g^{y_{11}x_{11}}g^{v'_{11}}$ | $g^{y_{12}x_{12}}g^{v'_{12}}$ | ... | $g^{y_{1k}x_{1k}}g^{v'_{1k}}$ |
| $g^{y_{21}x_{21}}g^{v'_{21}}$ | $g^{y_{22}x_{22}}g^{v'_{22}}$ | ... | $g^{y_{2k}x_{2k}}g^{v'_{2k}}$ |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| $g^{y_{\mu 1}x_{\mu 1}}g^{v'_{\mu 1}}$ | $g^{y_{\mu 2}x_{\mu 2}}g^{v'_{\mu 2}}$ | ... | $g^{y_{\mu k}x_{\mu k}}g^{v'_{\mu k}}$ |

Table 4.3: Bulletin Board $B$

$(\Sigma_{l=1}^{i-1}x_{lj} - \Sigma_{l=i+1}^{\mu}x_{lj})$ *and* $y_{ij}$ *is unknown* $\forall j \in \{1,2,...,k\}, \forall i \in \{1,2,...,\mu\}, \mu \in \mathbb{N}$*, the two bulletin boards* $A$ *and* $B$ *as in Table 4.4 and 4.5 respectively are indistinguishable, where*

1. $v_{ij}, v'_{ij} \in R, \forall j \in \{1,2,..,k\}, \forall i \in \{1,2,...,n-\mu\}$
2. $\cup_{j=1}^{k} v_{ij} = \cup_{j=1}^{k} v'_{ij} = R, \forall i \in \{1,2,...,n-\mu\}$
3. $\Sigma_{j=1}^{k} v_{ij} = \Sigma_{j=1}^{k} v'_{ij} = \nu, \forall i \in \{1,2,...,n-\mu\}$
4. $\Sigma_{i=1}^{n-\mu} v_{ij} = \Sigma_{i=1}^{n-\mu} v'_{ij} = v_j, \forall j \in \{1,2,...,k\}$

*Proof.* Let us choose votes $v_{ij} \in R$, for all $i \in \{n-\mu+1, n-\mu+2, ..., n\}$ and for all $j \in \{1,2,...,k\}$ such that $\cup_{j=1}^{k}\{v_{ij}\} = R$, for all $i \in \{n-\mu+1, n-\mu+2, ..., n\}$. Now we set $v'_{ij} = v_{ij}$, for all $i \in \{n-\mu+1, n-\mu+2, ..., n\}$ and for all $j \in \{1,2,...,k\}$. After this step, $\Sigma_{i=1}^{n} v_{ij} = \Sigma_{i=1}^{n} v'_{ij} = v_j + \Sigma_{i=n-\mu+1}^{n} v_{ij}$, for all $j \in \{1,2,...,k\}$. Now, if an attacker can distinguish between the two bulletin boards, she will be able to disprove Lemma 4.5.4. □

**Lemma 4.5.6.** *Considering a partial collusion where an attacker colludes with* $\mu < (n-1)$ *voters, the attacker will only learn the partial tally of* $n-\mu$ *honest voters, not the individual votes of honest voters.*

*Proof.* Without loss of generality, we assume that $\{V_{n-\mu+1}, V_{n-\mu+2}, ..., V_n\}$ is the set of colluding voters and $\{V_1, V_2, ..., V_{n-\mu}\}$ is the set of honest voters. In the proposed Borda count

scheme, each voter $V_i$ chooses $(x_{i1}, x_{i1}, ..., x_{ik})$ uniformly at random from $(\mathbb{Z}_q)^k$ and publishes $(g^{x_{i1}}, g^{x_{i2}}, ..., g^{x_{ik}})$ in the first round along with $k$ zero-knowledge proofs $P_K\{x_{ij}\}, \forall j \in \{1, 2, ..., k\}$. In the second round, each voter $V_i$ publishes $(Z_{i1}, Z_{i2}, ..., Z_{ik})$, where $Z_{ij} = \{g^{y_{ij}}\}^{x_{ij}} g^{v_{ij}}, y_{ij} = \Sigma_{l<i} x_{lj} - \Sigma_{l>i} x_{lj}, \forall j \in \{1, 2, ..., k\}$. The attacker will know votes of the colluding voters and their randomness, i.e. attacker will know $(v_{i1}, v_{i2}, ..., v_{ik})$ and $(x_{i1}, x_{i1}, ..., x_{ik})$ for each colluding voter $V_i$, where $i \in \{n - \mu + 1, n - \mu + 2, ..., n\}$. Therefore, the attacker can compute $(Z_{i1}, Z_{i2}, ..., Z_{ik})$ for each colluding voter $V_i$, where $i \in \{n - \mu + 1, n - \mu + 2, ..., n\}$. According to the protocol, $y_{ij} = (\Sigma_{l=1}^{i-1} x_{lj} - \Sigma_{l=i+1}^{n} x_{lj})$, $\forall j \in \{1, 2, ..., k\}, \forall i \in \{1, 2, ..., n\}$. From Lemma 4.5.1, in case of partial collusion, we can conclude that $y_{ij}$'s are unknown $\forall j \in \{1, 2, ..., k\}, \forall i \in \{1, 2, ..., n\}$. Hence. the attacker's view of the bulletin board will be same as given in Table 4.4.

Now according to the Lemma 4.5.5, the attacker will not be able to distinguish the two bulletin boards given in Table 4.4 and Table 4.5 with the same partial tally for honest voters. Therefore, the attacker will only learn the partial tally of honest voters, not the individual votes of honest voters. □

| | | | |
|---|---|---|---|
| $g^{u_{11}} g^{v_{11}}$ | $g^{u_{12}} g^{v_{12}}$ | ... | $g^{u_{1k}} g^{v_{1k}}$ |
| $g^{u_{21}} g^{v_{21}}$ | $g^{u_{22}} g^{v_{22}}$ | ... | $g^{u_{2k}} g^{v_{2k}}$ |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| $g^{u_{n-\mu 1}} g^{v_{n-\mu 1}}$ | $g^{u_{n-\mu 2}} g^{v_{n-\mu 2}}$ | ... | $g^{u_{n-\mu k}} g^{v_{n-\mu k}}$ |
| $g^{u_{n-\mu+11}}$ | $g^{u_{n-\mu+12}}$ | ... | $g^{u_{n-\mu+1k}}$ |
| $g^{u_{n-\mu+21}}$ | $g^{u_{n-\mu+22}}$ | ... | $g^{u_{n-\mu+2k}}$ |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| $g^{u_{n1}}$ | $g^{u_{n2}}$ | ... | $g^{u_{nk}}$ |

Table 4.4: Bulletin Board A, where $u_{ij} = y_{ij} x_{ij}, \forall j \in \{1, 2, ..., k\}$ and $\forall i \in \{1, 2, ..., n\}$.

### 4.5.2 Self-tallying

In our protocol, during the tallying phase, the final tally can be computed by anyone, including voters and observers of the protocol, without any external help. This can be easily

| | | | |
|---|---|---|---|
| $g^{u_{11}}g^{v'_{11}}$ | $g^{u_{12}}g^{v'_{12}}$ | ... | $g^{u_{1k}}g^{v'_{1k}}$ |
| $g^{u_{21}}g^{v'_{21}}$ | $g^{u_{22}}g^{v'_{22}}$ | ... | $g^{u_{2k}}g^{v'_{2k}}$ |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| $g^{u_{n-\mu 1}}g^{v'_{n-\mu 1}}$ | $g^{u_{n-\mu 2}}g^{v'_{n-\mu 2}}$ | ... | $g^{u_{n-\mu k}}g^{v'_{n-\mu k}}$ |
| $g^{u_{n-\mu+11}}$ | $g^{u_{n-\mu+12}}$ | ... | $g^{u_{n-\mu+1k}}$ |
| $g^{u_{n-\mu+21}}$ | $g^{u_{n-\mu+22}}$ | ... | $g^{u_{n-\mu+2k}}$ |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| $g^{u_{n1}}$ | $g^{u_{n2}}$ | ... | $g^{u_{nk}}$ |

Table 4.5: Bulletin Board B, where $u_{ij} = y_{ij}x_{ij}, \forall j \in \{1, 2, ..., k\}$ and $\forall i \in \{1, 2, ..., n\}$.

checked by observing the protocol and the *Proposition 1.* In the first round of the protocol, voters choose their private key, and in the second round their public keys are combined in such a way that the random factors vanishes after the second round. Thus, during the tallying phase, the final tally can be computed correctly by any observer of the protocol. Therefore, our protocol satisfies the self-tallying property. The use of zero-knowledge proofs in the protocol is to ensure that all voters follow the protocol faithfully.

### 4.5.3 Dispute-freeness

Our protocol also satisfies the dispute-freeness property. The use of $k$ non-interactive zero-knowledge proofs in the first round of the protocol ensures that the voter knows the secret keys $(x_{i1}, x_{i2}, ..., x_{ik})$ corresponding to the public keys $(g^{x_{i1}}, g^{x_{i2}}, ..., g^{x_{ik}})$. The use of $k$ non-interactive Cramer, Damgård, and Schoenmakers [37] zero-knowledge proofs ensures that each ballot will encode exactly one permutation of $(a_1, a_2, ..., a_k)$. Furthermore, since we use a public authenticated channel, any attempt to cast more than one vote can be detected by other participants of the protocol. Thus, our protocol enforces the one-man-one-vote requirement, which, combined with the public verifiablity of all operations in the protocol, ensures dispute-freeness.

### 4.5.4 Limitation

In this section, we discuss the limitations of our protocol. One limitation is that all voters must follow the protocol till the tally process. If some voters do not send data in the second round of the protocol, no one will be able to compute the tally, and hence the tallying will fail. However, this attack is conspicuous since everyone will be able to identify the attackers. In that case, voters can exclude the attackers from the election and restart the protocol to recover from the attack. Note that the voter's privacy is still preserved. Nevertheless, there would be delay in the election process.

In our implementation of the protocol using Ethereum Blockchain and smart contract, we use a deposit and refund paradigm [85] to enforce a financial incentive to all voters as a countermeasure to this issue. All voters are required to deposit some money into the smart contract to register for the election. This deposited money is refunded to the voter after she successfully casts her ballot. However, a registered voter will lose her deposit if she does not cast her vote successfully in the second round of the protocol.

Our protocol is not coercion-resistant since a voter can be coerced to vote for a particular candidate and to reveal her secret parameters to prove how she voted. This is another limitation of our system. Normally, this kind of coercion resistance is provided in a supervised environment like in a polling station [74]. However, in decentralized environment where the voting process is unsupervised, providing coercion resistance seems difficult.

Here, we mention that the above limitations also exist in the Kiayias and Yung [82], Groth [57] and open vote network [60] protocols. Although a centralised voting protocol that is executed in a supervised environment with trusted election authorities may prevent this kind of issues, however, the trustworthiness of the election authorities is often called into question. In our proposed protocol, the tally can be computed by voters themselves (and any observer of the protocol) without involving any tallying authority.

Therefore, for a small-scale election where the above limitations are not of great concern, the proposed decentralised Borda count protocol can prove to be useful and efficient.

## 4.6 Performance analysis of the protocol

In this section, we analyze the computation and communication cost of our proposed protocol. There are only two rounds in our protocol. As discussed in [52], in any secure multy-party computation protocol, minimum two rounds are required to compute a function securely. Since exponentioations are the most expensive operations in our protocol, we analyze our protocol in terms of the number of exponentiations.

We assume that the number of candidates competing in the election is $k$. In the first round, each voter needs to do $k$ exponentiations to generate public keys. In addition, there are $k$ NIZK proofs corresponding to their $k$ secret keys. Each of these NIZK proofs requires one exponentiation for creation and 1.2 exponentiations for verification. (Here we do not count the cost of validating a public key, which requires one exponentiation in the finite field setting but is essentially free in the elliptic curve setting.) We assume that the simultaneous multiple exponentiation (SME) technique [97] is used to optimize computations. Using the SME method, a term of the form $g^x h^y$ requires about 1.2 exponentiations to calculate. Hence, a voter needs to do $2k$ exponentiations in the first round to create her public keys along with NIZK proofs. In order to verify these NIZK proofs, one needs to do $1.2k$ exponentiations. In the second round, a voter needs to do $k$ exponentiations to generate the ballot. In addition, the voter needs to create $k$ 1-out-of-k NIZK proofs. Each of these 1-out-of-k NIZK proofs requires $(2.4(k-1)+2)$ exponentiations for generation and $2.4k$ exponentiations for verification of the same. Hence, all $k$ 1-out-of-k NIZK proofs require $(2.4k^2 - 0.4k)$ exponentiations for generation and $2.4k^2$ for verification. Hence, in the second round, a voter needs to do $(k + ((2.4k^2 - 0.4k)))$ exponentiations to generate her ballot. All together, a voter needs to do $(2k + (k + ((2.4k^2 - 0.4k))))$ exponentiations i.e. $(2.4k^2 + 2.6k)$ exponentiations to cast her vote in the election. The total number of exponentiations required to verify a ballot is equal to $(1.2k + 2.4k^2)$. In the first round of the protocol, the size of public keys of a voter is $k$ elements of group the $\mathbb{G}_q$. In addition, in the first round, the size of $k$ NIZK proofs is $k$ elements of $\mathbb{Z}_q$ plus $k$ elements of $\mathbb{G}_q$. Hence, the size of the public keys along with NIZK proofs is $2k$ elements of the group $\mathbb{G}_q$ plus $k$ elements of the group $\mathbb{Z}_q$. In the second round of the protocol, the size of each ballot is $k$

elements of the group $\mathbb{G}_q$. In addition, the size of each of $k$ 1-out-of-k NIZK proofs is $2k$ elements of $\mathbb{Z}_q$ plus $2k$ elements of $\mathbb{G}_q$. Hence, in the second round, the size of each ballot along with $k$ 1-out-of-k NIZK proofs is $(k+2k^2)$ elements of the group $\mathbb{G}_q$ plus $2k^2$ elements of the group $\mathbb{Z}_q$. Therefore, all together, the total space required by a voter is $(3k + 2k^2)$ elements of the group $\mathbb{G}_q$ plus $(2k^2 + k)$ elements of the group $\mathbb{Z}_q$.

Table 4.6 and Table 4.7 highlight the computation and communication cost (space) respectively for the proposed Borda count protocol when $k$ candidates compete in the election.

Note that the OV-Net protocol requires $(2.4k - 0.4)$ exponentiations (using the SME technique) for generation and $2.4k$ exponentiations for verification of the NIZK proof in the second round of the protocol. Therefore, the number of exponentiations required in the OV-Net protocol is $O(k)$ that is linear with the number of candidates contesting in the election. However, our proposed Borda count protocol requires $(2.4k^2 - 0.4k)$ exponentiations for generation and $2.4k^2$ exponentiations for verification of the NIZK proofs in the second round of the protocol. Therefore, the number of exponentiations required in our proposed protocol is $O(k^2)$ that is quadratic with the number of candidates competing in the election. Our protocol requires significantly more computation than OV-net which is based on plurality voting, because a ranked-choice voting system is inherently more complex than a non-ranking based voting system. However, a ranked-choice voting system tends to give a fairer outcome as the system takes in more information from voters about their preferences than a non-ranking based system (e.g., plurality).

| First round | | | Second round | | |
|---|---|---|---|---|---|
| Public keys | NIZKP | Total | Ballot | NIZKP | Total |
| $k$ | $k$ | $2k$ | $k$ | $2.4k^2 - 0.4k$ | $2.4k^2 + 0.6k$ |

Table 4.6: The computation cost for the proposed scheme in number of exponentiations when $k$ candidates compete in the election.

We now compare the performance of our proposed Borda count ranked-choice voting system with some of the well-known non-ranking based voting systems. Several cryptographic voting protocols are proposed in the literature, however, most of them offer security and integrity assurance by introducing a set of trustworthy tallying authorities [82, 57]. For the

| First round | | | Second round | | |
|---|---|---|---|---|---|
| Public keys | NIZKP | Total | Ballot | NIZKP | Total |
| $k \cdot a$ | $k \cdot a + k \cdot b$ | $2k \cdot a + k \cdot b$ | $k \cdot a$ | $2k^2 \cdot a + 2k^2 \cdot b$ | $(k + 2k^2) \cdot a + 2k^2 \cdot b$ |

Table 4.7: The communication cost (space) for the proposed scheme when $k$ candidates compete in the election. $a$ and $b$ represent the size of each element of the group $\mathbb{G}_q$ and $\mathbb{Z}_q$ respectively.

purpose of comparison, we consider only the self-tallying voting protocols without involving any tallying authorities. Therefore, we compare our protocol mainly with the Kiayias-Yung [82], Groth [57] and OV-Net [60] protocols. Table 4.8 highlights a comparison between our proposed Borda count protocol and these three previously proposed plurality voting solutions.

In [82], Kiayias-Yung proposed a 3-round veto protocol. In the first round, each voter $i$ chooses a random value $x_i$ from $\mathbb{Z}_q$ and publishes her public key $P_i = g^{x_i}$. In this round, the voter needs to compute one exponentiation and the corresponding NIZK proof for the knowledge of the exponent. In the second round, each voter chooses $n$-vector private keys $(y_1, y_2, ..., y_n)$ uniformly at random from $\mathbb{Z}_q^n$ and publishes the corresponding public keys $(g^{y_1}, g^{y_2}, ..., g^{y_n})$, where $n$ is the number of voters. To perform this step, a voter needs to compute $n$ exponentiations. Each voter also computes $(P_1^{y_1}, P_2^{y_2}, ..., P_n^{y_n})$, which requires $n$ more exponentiations. In round three, each voter performs one more exponentiation. After this step, the tally can be computed universally. Therefore, each voter needs to perform $(2n + 2)$ exponentiations in total. For each exponentiation, the voter needs to publish the corresponding NIZK proof. Each voter publishes $O(nk)$ data, where $k$ is the number of candidates contesting in the election, and $n$ is the total number of voters. The final tally is computed from $O(n^2k)$ data. Table 4.8 summarizes the performance of this protocol.

Groth [57] investigated the Kiayias-Yung's protocol in order to reduce its system complexity. Groth's protocol has $(n+1)$ rounds, where $n$ is the total number of voters. In round 1, each voter $i$ publishes her public key. After this step, each voter sends her encrypted vote one after another depending on the result sent by the previous voter. As a result, instead of

finishing the protocol in three rounds as in [82], the protocol requires $n+1$ rounds, where $n$ is the total number of voters. In total, the first voter needs to compute three exponentiations and the corresponding NIZK proofs. All other voters need to compute four exponentiations along with four corresponding NIZK proofs in total. Each voter publishes $O(k)$ data, where $k$ is the number of candidates contesting in the election. The final tally is computed from $O(nk)$ data.

The OV-Net protocol [60] executes in two rounds, which is more efficient than the Kiayias-Yung [82] and Groth's [57] protocols. In the first round, every voter publishes one public key along with a NIZK proof of the corresponding secret key. A voter needs to compute one exponentiation and one NIZK proof for the knowledge of the exponent. In the second round, each voter sends her encrypted vote along with one 1-out-of-k NIZK proof. In this round, the voter needs to compute one exponentiation and one 1-out-of-k NIZK proof. Each voter publishes $O(k)$ data, $k$ being the number of candidates contesting in the election. The final tally is computed from $O(nk)$ data, where $n$ is the total number of voters.

All of the protocols discussed above are designed to implement a plurality voting electoral system. We propose a Borda count voting protocol that is a ranked-choice voting system. Our proposed protocol runs in two rounds. In the first round, each voter $V_i$ calculates $X_{i1} = g^{x_{i1}}, X_{i2} = g^{x_{i2}}, ..., X_{ik} = g^{x_{ik}}$ and publishes $(X_{i1}, P_K\{x_{i1}\}, X_{i2}, P_K\{x_{i2}\}, \ldots, X_{ik}, P_K\{x_{ik}\})$, where $P_K\{x_{ij}\}$ is the NIZK proof for $x_{ij}; \forall j \in \{1, 2, ..., k\}$. A voter needs to compute $k$ exponentiations and $k$ NIZK proofs for the exponents. In the second round, each voter $V_i$ publishes $(Z_{i1}, Z_{i2}, ..., Z_{ik})$ and $k$ 1-out-of-k NIZK proofs $\Pi_j[x_{i1}, x_{i2}, ..., x_{ik} : X_i, Z_i]; \forall j \in \{1, 2, ..., k\}$, where $Z_{ij} = \{g^{y_{ij}}\}^{x_{ij}} g^{v_{ij}}; \forall j \in \{1, 2, ..., k\}$. Our protocol adopts the same cancelling technique as in [60] to achieve self-tallying without involving any tallying authority, but our system is designed to support ranked-choice voting instead of plurality voting. The construction and verification of zero-knowledge proofs are different from [60]. In our system, each voter publishes $O(k^2)$ data, where $k$ is the number of candidates contesting in the election. The final tally is computed from $O(nk^2)$ data, where $n$ is the total number of voters.

| Protocols | Election type | Rounds | Exp | NIZKP for exponent | NIZKP for equality | 1-out-of-k NIZKP | Total traffic | Total computation |
|---|---|---|---|---|---|---|---|---|
| Kiayias-Yung [82] | Plurality (Non-ranking based ) | 3 | $2n+2$ | $n+1$ | $n$ | 1 | $O(n^2k)$ | $O(n^2k)$ |
| Groth [57] | Plurality (Non-ranking based ) | $n+1$ | 4 | 2 | 1 | 1 | $O(nk)$ | $O(nk)$ |
| OV-Net [60] | Plurality (Non-ranking based ) | 2 | 2 | 1 | 0 | 1 | $O(nk)$ | $O(nk)$ |
| Proposed protocol | Borda count (rank-choice based ) | 2 | $2k$ | $k$ | 0 | $k$ | $O(nk^2)$ | $O(nk^2)$ |

Table 4.8: Comparison with related protocols proposed in the literature. The number of participants in the election is $n$, and the number candidates contesting in the election is $k$.

## 4.7 A smart contract implementation

The proposed decentralized Borda Count protocol is suitable for implementation over a Blockchain. In this paper, we propose a smart contract implementation of our protocol on Ethereum in order to enforce the execution of the voting protocol. We are using Ethereum since it can store and execute programs that are written as smart contracts. Ethereum's consensus mechanism enforces the correct execution of these smart contracts. Its peer-to-peer network serves as an authenticated public channel. We use the blockchain as a public bulletin board as well as to enforce the execution of the election process in a timely manner.

### 4.7.1 Ethereum

An Ethereum transaction is a digitally signed instruction constructed by a user of the Ethereum blockchain. There are two types of transactions: those that create smart contracts and those that are responsible for message calls. An Ethereum transaction consists of several fields, which specify the sender's and the receiver's addresses, and the transaction data [96].

There are two types of accounts available in Ethereum blockchain. They are called Contract Account and Externally Owned Account.

A user creates a smart contract using her externally owned account. The Ethereum

currency 'Ether' can be stored in both of these types of accounts. A user needs to purchase 'gas' using 'Ether' currency in order to execute a smart contract. The gas price is set according to the conversion rate of ether to gas. For each assembly 'opcode' (instruction), there is a fixed gas cost to execute the instruction depending on its execution time. The cost of gas to execute a transaction is regarded as the transaction fee. This transaction fee is an incentive for the miners to add the transaction into their block.

The integrity of the Ethereum blockchain depends on its 'proof-of-work' mechanism. The 'proof-of-work' is a computationally difficult puzzle that must be solved by a miner in order to get a block added to the blockchain. Ethereum's blockchain is a simplified version of the GHOST protocol introduced by Sompolinsky and Zohar [128]. In Ethereum's blockchain, blocks are created in every 12 seconds interval. As a result, there is a possibility that two or more blocks would be created at the same time by different miners, and hence some blocks would be discarded. These discarded blocks are added to the blockchain as 'uncle blocks', and the corresponding miners still get some rewards in 'Ether' currency. Furthermore, if the same smart contract is called by multiple transactions, the final state of the smart contract is determined by the order of execution of the transactions.

We now discuss our implementation of the voting protocol on Ethereum in the following sections.

### 4.7.2 Structure of Implementation

We follow the similar design architecture as proposed by McCorry et al. in [96]. However, in [96], McCorry et al. implemented the OV-Net protocol only for a two-candidate election ('yes'/'no' voting). We implement the proposed Borda count protocol for multiple candidates who are contesting in the election. Note that OV-Net protocol is designed for plurality voting which is a non-ranking based voting system. However, our proposed protocol is designed for Borda count voting which is a 'ranking-based' voting system. The two election processes are different. In addition, the structure of ballots and the NIZK proofs are also different.

While implementing our protocol over Ethereum, some of the tasks exceeded the maximum gas limit (approximately 8 million gas as in June, 2019) of a block. In [96], a single transaction is sufficient to perform a task, however, due to the complexity of the Borda

count system, we had to send multiple transactions in parallel to complete some tasks. We split the tasks into several smaller tasks (within the gas limit) in order to execute them in parallel. We have developed two smart contracts in Solidity language:

(1) **VoteContract:** The voting protocol is implemented in this contract. This contract also controls the voting process and checks the validity of all zero-knowledge proofs.

(2) **CryptoContract:** This contract consists of the code for creating zero-knowledge proofs. All voters call this smart contract to create zero-knowledge proofs so that the same code can be used by all voters. This contract can be executed locally without interacting with Ethereum blockchain.

The election administrator is the owner of these two smart contracts. We have also developed the following three HTML5/Javascript pages to provide browser interfaces for the users:

(1) **administrator.html:** This page is constructed for the administrator to manage the entire election process. The election administrator sets a list of eligible voters and a list of timers using this page. The timers are used to set the deadlines for various stages of the election process so the voting process progresses in a timely manner. This page also contains the code to notify the Ethereum blockchain to begin the registration process, to close the registration process and begin to cast the vote, to close the voting and compute the tally based on these timers.

(2) **voter.html:** This page is implemented to facilitate the voter to register and cast her vote.

(3) **live.html:** This page is dedicated for any observer of the election process to watch the progress of the election including the beginning and ending of each stage, voter registration process and vote casting process. This page shows the addresses of current registered voters, addresses of the voters who have committed their vote and addresses of the voters who have already cast their vote. However, it is not possible to compute the running tally.

In addition, we have implemented a Java program to locally generate the random private keys $(x_{i1}, x_{i2}, ..., x_{ik})$ and the corresponding public keys $(g^{x_{i1}}, g^{x_{i2}}, ..., g^{x_{ik}})$ for a voter. The private keys are kept secret by the voter. These keys are required by the voter's web browser voter.html to register and cast her vote.

We use the Web3 framework provided by the Ethereum foundation for communication between the user's web browser and the Ethereum client. In our implementation, the user does not need to directly interact with the Ethereum client, and it is sufficient if the Ethereum client runs in the background. The election administrator and all voters need to have their own Ethereum account. The election administration sets the list of eligible voters using their Ethereum account address. Users (including voters and the election administrator) need to unlock their account using their password to send transactions to the blockchain from their web browser. The password is used to decrypt their private key that is necessary to generate the digital signature of the transaction.

### 4.7.3 Election Stages

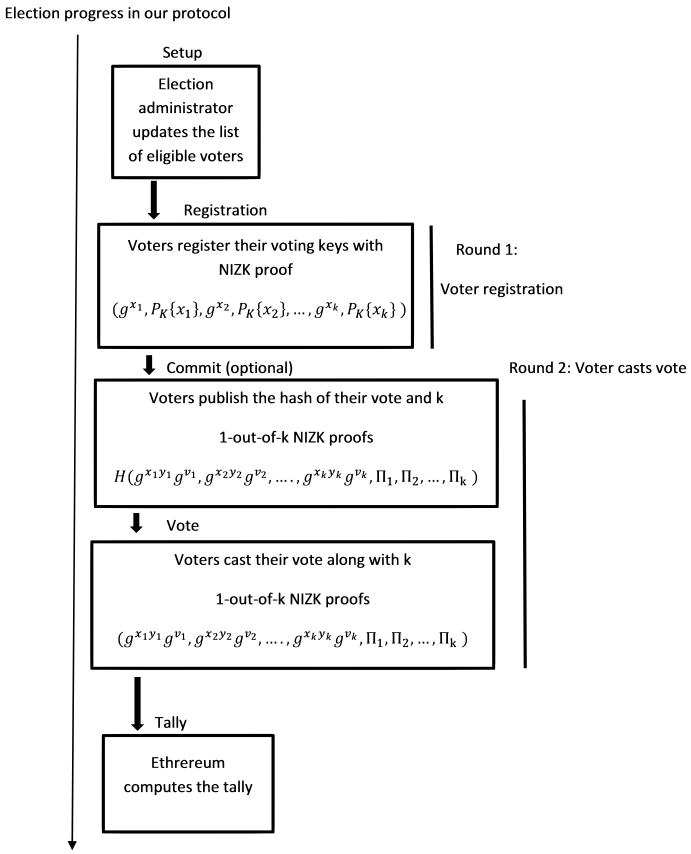


Figure 4.1: Election stages in our protocol implementation.

The election runs in five stages in our implementation. Figure 4.1 depicts these stages. The election administrator arranges the election process by providing a list of timers and

eligible voters to the smart contract. The contract only permits eligible voters to register and cast their vote before the deadline as specified by the list of timers. In addition, as mentioned before, an eligible voter may need to deposit ether in the contract to register for the election. This deposit is automatically refunded to the voter after her vote is successfully accepted by Ethereum blockchain. We now describe these election stages in detail.

**Setup:** At this stage, the election administrator sets a list of eligible voters using their Ethereum account address, a list of timers and the registration deposit $d$ on the Ethereum blockchain. The election administrator also sets whether the optional commit stage should be enabled or not. The timers are used to set the closing time (deadline) for each stage of the election to enforce that the election process progresses in a predefined timely manner. The list of timers includes the following timers.

$-T_{endreg}$*:* Voters must register for the election before this time limit. To register for the election, a voter needs to send her voting keys along with the NIZK proofs $(g^{x_{i1}}, P_K\{x_{i1}\}, g^{x_{i2}}, P_K\{x_{i2}\}, ..., g^{x_{ik}}, P_K\{x_{ik}\})$, where $P_K\{x_{ij}\}$ is the NIZK proof of $x_{ij}; \forall j \in \{1, 2, ..., k\}$ and $k$ is the number of candidates competing in the election, to the blockchain before this time.

$-T_{startvote}$*:* This represents the time before which the blockchain must be notified by the election administrator to start the election (i.e. the second round of our protocol). If the election administrator fails to notify the blockchain by this time, the voting process will be aborted, and all registered voters will get their deposit back.

$-T_{endcommit}$*:* If the Commit (optional) stage is enabled by the election administrator in this Setup stage, voters must send the hash of their vote to the blockchain as a commitment before this time. All voters send $H((Z_{i1}, Z_{i2}, ..., Z_{ik}), \Pi_j[x_{i1}, x_{i2}, ..., x_{ik} : X_i, Z_i]; \forall j \in \{1, 2, ..., k\})$, where $\Pi_j[x_{i1}, x_{i2}, ..., x_{ik} : X_i, Z_i]; \forall j \in \{1, 2, ..., k\}$ are $k$ non-interactive zero-knowledge proofs corresponding to the vote in the second round of our protocol and $Z_{ij} = \{g^{y_{ij}}\}^{x_{ij}} g^{v_{ij}}; \forall j \in \{1, 2, ..., k\}$, $k$ is the number of candidates competing in the election. This time is effective only if the Commit stage (optional) is enabled. If some registered voters fail to commit their vote by this time, the election process will be aborted. In that case, all other registered voters who have successfully committed their vote by this time can get their deposit back.

$-T_{endvote}$*:* This represents the time before which all voters must cast their vote in

the election. All voters send their vote $((Z_{i1}, Z_{i2}, ..., Z_{ik}), \Pi_j[x_{i1}, x_{i2}, ..., x_{ik} : X_i, Z_i]; \forall j \in \{1, 2, ..., k\})$ as defined in the second round of the protocol. If some registered voters fail to cast their vote by this time, the election process will be aborted. The deposit of those voters will be confiscated. However, all other registered voters who have successfully cast their vote by this time will automatically get their deposit back.

$-T_{min}$*:* This represents the minimum length of time in which the commitment stage and voting stage must remain active. In a decentralized voting protocol, voters should be given sufficient time to commit and cast their votes. Therefore, the commitment and voting stage should remain active for a sufficient amount of time.

At the end of this stage, the election administrator notifies the Ethereum blockchain to change the state (of the contract) from the Setup to the Registration stage.

**Registration:** At this stage, all eligible voters first review the parameters of the election set by the election administrator such as timers (closing time for each stage of the election) and the registration deposit $d$. Then eligible voters may choose to register for the election. Let there be $k$ candidates competing in the election. To register for the election, a voter needs to compute her voting keys along with the non-interactive zero-knowledge proofs. The voter sends the voting keys along with NIZK (non-interactive zero-knowledge) proofs to the blockchain along with a deposit of $d$ ether. Ethereum verifies the NIZK proofs by executing the smart contract code and adds the transaction to the blockchain upon successful verification of the NIZK proofs. Ethereum rejects all registration request transactions that are received after $T_{endreg}$. At the end of this stage, the election administrator notifies the blockchain to change the state from Registration to either the optional Commit stage or the Vote stage depending on whether the optional Commit stage is enabled or not. During this transition, Ethereum computes all voter's reconstructed keys $(g^{y_{i1}}, g^{y_{i2}}, ..., g^{y_{ik}})$, where $g^{y_{ij}} = \prod_{l=1}^{i-1} g^{x_{lj}} / \prod_{l=i+1}^{n} g^{x_{lj}}; \forall j \in \{1, 2, ..., k\}$.

**Commit (optional):** If the optional commit stage is enabled, all registered voters publish the hash of their vote and NIZK proofs $H((Z_{i1}, Z_{i2}, ..., Z_{ik}), \Pi_j[x_{i1}, x_{i2}, ..., x_{ik} : X_i, Z_i]; \forall j \in \{1, 2, ..., k\})$ to the blockchain. Ethereum rejects all commitment transactions that are received after $T_{endcommit}$. After receiving commitments from all registered voters, the contract automatically changes its state to the Vote stage.

**Vote:** All voters cast their vote at this stage. All voters publish their ballot - $(Z_{i1}, Z_{i2}, ..., Z_{ik}), \Pi_j[x_{i1}, x_{i2}, ..., x_{ik} : X_i, Z_i]; \forall j \in \{1, 2, ..., k\}$ to the blockchain. Ethereum accepts the transaction if the verification of all the NIZK proofs succeeds. The smart contract refunds the deposit $d$ to the voter after successfully accepting her ballot (encrypted vote). Ethereum rejects all ballots that are received after $T_{endvote}$. The administrator notifies Ethereum to change the state of the contract to the Tally stage after all voters cast their ballots.

**Tally:** At this stage, Ethereum computes the tally for each candidate once it receives the corresponding notification from the election administrator. To compute the tally for the $j$-th candidate, Ethereum evaluates $\prod_{i=1}^{n}\{g^{y_{ij}}\}^{x_{ij}} g^{v_{ij}} = g^{\Sigma_{i=1}^{n} v_{ij}}$, where $j \in \{1, 2, ..., k\}$. Ethereum then computes the total score $\Sigma_{i=1}^{n} v_{ij}$ obtained by the $j$-th candidate using brute force search, for each $j \in \{1, 2, ..., k\}$. Note that this brute force search is feasible since the tally is a small number.

### 4.7.4 Design choices

We now focus on the design choices for implementing our protocol on Ethereum.

**Score associated with each rank and the tally computation.** In a Borda count protocol, a score is associated with each rank. Let there be $k$ candidates contesting in the election. In our proposed scheme, a voter's vote will be of the form $(v_1, v_2, ..., v_k)$, where $(v_1, v_2, ..., v_k)$ is a permutation of $(a_1, a_2, ..., a_k)$. The score $a_j$ is associated with the $j$-th rank and $a_{j-1} > a_j, \forall j \in \{1, 2, .., k\}$ i.e a higher score implies higher rank of the candidate. In our implementation, we set the following scores associated to each rank: $a_1 = k, a_2 = k - 1, ..., a_k = 1$. Therefore, at the tallying phase, the total score obtained by a candidate (final tally) can be at most $nk$, where $n$ is the number of participants in the election. Furthermore, the minimum total score that can be obtained by a candidate is $n$. In our proposed protocol, at the tallying stage, anyone can compute $\prod_{i=1}^{n}\{g^{y_{ij}}\}^{x_{ij}} g^{v_{ij}} = g^{\Sigma_{i=1}^{n} v_{ij}}$, where $j \in \{1, 2, ..., k\}$. The total score obtained by the $j$-th candidate is $\Sigma_{i=1}^{n} v_{ij}$. Therefore, $n \leq \Sigma_{i=1}^{n} v_{ij} \leq nk$, which is normally a small number. Thus, the exhaustive search (brute-force) requires at most $(n(k-1)+1)$ operations to compute the discrete logarithm of $g^{\Sigma_{i=1}^{n} v_{ij}}$ for all $j \in \{1, 2, .., k\}$, where $n$ is the number of participants in the election.

**The election administrator.** In our implementation, the election administrator sets

the parameters of the election, list of eligible voters. The election administrator notifies the smart contract to begin the Registration stage. Note that a smart contract cannot start executing code without user interaction. Therefore, there must be a user who can take the responsibility of notifying the smart contract to begin the election process and compute the tally. We assume that the election administrator notifies the blockchain. A registered voter can also notify the blockchain.

**Preventing re-entrancy attack.** This attack is possible if a smart contract sends 'ether' to a user before deducing their balance. An attacker can exploit this vulnerability by calling the smart contract recursively in such a way that sending of ether is repeated, however, the deduction of balance is done only once. Using re-entrancy vulnerability in DAO, an attacker stole 3.6 million ether. Luu et al. [93] analyze that there are 186 distinct smart contracts stored on the blockhain vulnerable to this attack. To thwart this attack, Reitwiessner [115] suggests to first deduce the balance before sending the ether. We follow the same in our implementation.

**Preventing replay attack.** In the first round of our protocol, all voters send publicly their public keys with NIZK proofs $(g^{x_{i1}}, P_K\{x_{i1}\}, g^{x_{i2}}, P_K\{x_{i2}\}, ..., g^{x_{ik}}, P_K\{x_{ik}\})$. Another eligible voter might replay the same keys and NIZK proofs but without actually knowing the corresponding private keys. This will invalidate the purpose of the NIZK proof which is supposed to enforce the user to prove the knowledge of the private key. To prevent this replay attack, we follow the technique suggested in [60], by including the sender's unique identity ($msg.sender$) as an input argument to the hash function in the NIZK proofs. Therefore, no other voter can use the same NIZK proof as every sender's identity over the blockchain is different.

**Downloading full blockchain.** Currently, it is necessary to download the full Ethereum blockchain to verify the correct execution of the voting protocol. Note that a voter can participate in the election without downloading the full Ethereum blockchain, however in that case, the voter needs to trust Ethereum blockchain for correctly executing the protocol. The voter only needs to broadcast her transactions in the Ethereum blockchain network. They can use live.html page for confirmation of their registration and to view that their vote has been included in the blockchain. This enables voters with limited resources to cast their

vote.

**Refund of the deposit.** In our implementation, all registered voters must send their ballots before the deadline so as to ensure that their ballots are included in the blockchain. The deposit $d$ acts as a financial incentive that is refunded to the voters who adhere to the protocol and submit their ballots in time. However, those voters who fail to do so in time will face forfeiture of their deposit. The confiscated deposit can be used to compensate compliant votes or to support a charitable cause. The deposit is automatically refunded to a voter if either the voting process is successfully completed or if the voting process is aborted due to some other voter's failure in adhering to the protocol.

### 4.7.5 Experiment on Ethereum

We deployed our implementation on Ethereum's private network that mimics the production network. We have tested our implementation for different numbers of voters and for different candidates competing in the election. We present the results of our experiments with eighty voters and a varying number of candidates. In Table 4.9, we have outlined each transaction's computational and financial cost for eighty voters and five candidates contesting the election.

The total number of transactions in an election depends on the number of voters and the number of candidates competing in the election. The number of transactions that we sent to simulate the election process is shown in Table 4.9. The election administrator is denoted by the prefix 'AD:' and each voter is denoted by the prefix 'VT:' (see Table 4.9). Each transaction is a broadcast transaction. We have calculated the cost in US Dollar (denoted by '$' in the table) and rounded it to two decimal places.

Currently, the maximum gas limit of a single Ethereum block is approximately 8 million. The amount of gas required to store our code as a smart contract exceeds the gas limit of a single Ethereum block. Therefore, we had to create two smart contracts, namely, VoteContract and CryptoContract . The main protocol logic is implemented in the contract 'VoteContract', whereas, the code for the creation of the NIZK proofs is implemented in the contract 'CryptoContract'. The verification of the NIZK proofs is implemented in the contract 'VoteContract'.

As shown in Table 4.9, the voter registration costs approximately 48% of the block

| Entity: Transaction | A | B | C | D |
|---|---|---|---|---|
| AD: VoteContract | 1 | 5,498,780 | 5,498,780 | 1.70 |
| AD: CryptoContract | 1 | 3,698,878 | 3,698,878 | 1.14 |
| AD: Eligible | 1 | 4,349,336 | 4,349,336 | 1.34 |
| AD: Begin setup | 1 | 257,209 | 257,209 | 0.08 |
| VT: Register | 1 | 3,850,910 | 3,850,910 | 1.19 |
| AD: Begin election | 5 | 7,149,237 | 35,746,186 | 11.05 |
| VT: Commit | 1 | 183,457 | 183,457 | 0.06 |
| VT: Vote | 5 | 7,213,571 | 36,067,855 | 11.14 |
| AD: Tally | 5 | 5,629,272 | 28,146,362 | 8.70 |
| Administrator Total | 14 | | 77,696,751 | 24.01 |
| Voter total | 7 | | 40,102,222 | 12.39 |

Table 4.9: A breakdown of the cost for eighty participants using our protocol with five candidates competing in the election. The costs are approximated in USD ('$') using the conversion rate of 1 Ether=$309 and the gas price of 0.000000001 ether that are real world costs in June, 2019. We have approximated the cost for the election administrator 'AD' and the voter 'VT'. Columns are represented as - A: Number of transactions, B: Cost per transaction in Gas, C: Cumulative cost in Gas, D: Cumulative cost in '$'.

capacity when five candidates are competing in an election. Note that, as mentioned before, the maximum block capacity in Ethereum blockchain is about 8 million gas as in June, 2019. Thus, the current block supports at most two voter registrations per block. The vote casting costs more than the capacity of a block. Therefore, we made five transactions to cast a vote. The reason is that the number of exponentiations increases with the number of candidates completing in the election, and hence the cost of verifying NIZK proofs also increases. As shown in Table 4.9, we have made five transactions to cast a vote when the number of candidates competing in the election is five. Each transaction of vote casting costs approximately 90% of the block capacity. This suggests that five blocks are needed to cast a vote. Note that, in Ethereum, currently blocks are generated at a rate of one block in every 12 seconds.

Overall, the cost for the election administrator to run the election with 80 voters and 5 candidates is approximately $24.01. The average cost for each voter to run an election with 80 participants and 5 candidates is approximately $12.39.

The code in the contract 'CryptoContract' is executed locally on the voter's device, not on the Ethereum network. This contract provides the same NIZK proofs for all voters

throughout the execution of the protocol.

We have performed experiments with 3, 10, 20, 45, 60, 80 voters and with 2, 3, 4, 5 candidates and plotted the results to show how the costs for the election administrator and voter vary with different numbers of candidates. Figure 4.2 depicts the average cost for the election administrator based on the number of voters and the number of candidates competing in the election. From this figure, we see that the cost for the election administrator increases linearly with the number of voters. Figure 4.3 depicts the cost for each voter based on the number of voters participating in the election and the number of candidates competing in the election. From this figure, we see that the cost for each voter remains nearly constant with the number of voters participating in the election.

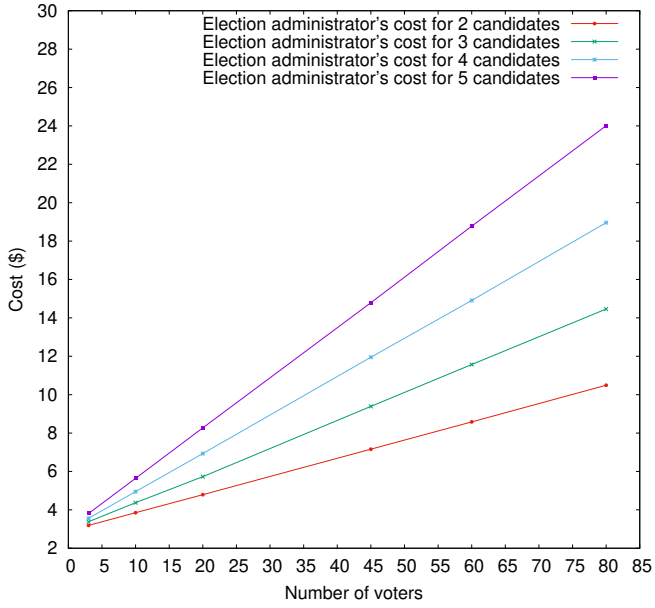


Figure 4.2: The election administrator's cost based on the number of voters participating in the election and the number of candidates competing in the election.

Figure 4.4 shows a breakdown of the election administrator's gas cost based on the number of voters when 5 candidates are competing in the election. The figure shows that the gas consumption for different tasks increases linearly with the number of voters except for beginning the registration. The maximum gas limit that an Ethereum block can consume was set at 8 million as in June, 2019. In our implementation, the number of transactions to compute reconstructed keys is equal to the number of candidates. Still, each transaction for

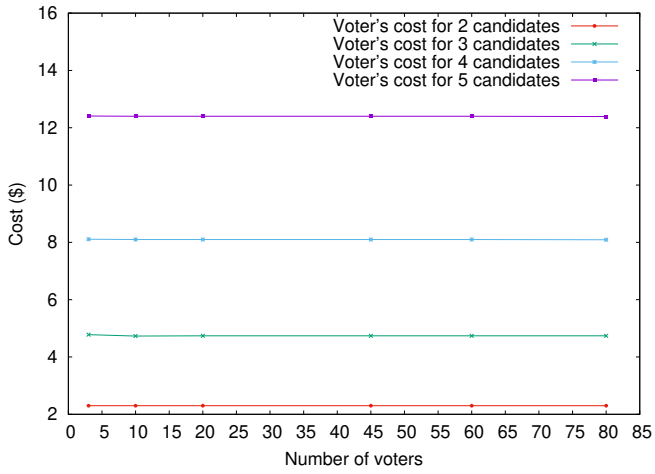


Figure 4.3: Each voter's cost based on the number of voters participating in the election and number of candidates competing in the election.

computing reconstructed keys reaches computation and storage limit for around 87 voters due to the block's gas limit. Due to this block gas limit, we had to make as many transactions as the number of candidates to compute the tally.

**Timing measurements analysis.** We have performed all the tests on a Chieftec desktop machine running Windows 10 Enterprise version 1809 equipped with 4 cores, 3.10 GHz Intel Core i5-2400 and 32 GB RAM. The timing analysis measurements of different tasks of our protocol are highlighted in Table 4.10. We have rounded all time measurements up to the nearest millisecond. The time measurement of each task is performed using the .call() function on the local daemon.

The voting contract 'VoteContract' contains the main logic of the protocol including the code for enforcing the election process in a timely manner. The task 'Register voting key' in Table 4.10 involves the verification of the Schnorr's zero-knowledge proofs created in the first round of the protocol. The time required to complete this action depends on the number of candidates competing in the election. Table 4.10 shows the time required to complete this action for different numbers of candidates. Beginning the election involves computing the reconstructed keys for every voter corresponding to each candidate. The time required for this action depends on both the number of voters and the number of candidates competing in the election. Casting a vote involves the verification of the 1-out-of-k zero-knowledge proofs created in the second round of the protocol. It depends on the number of candidates

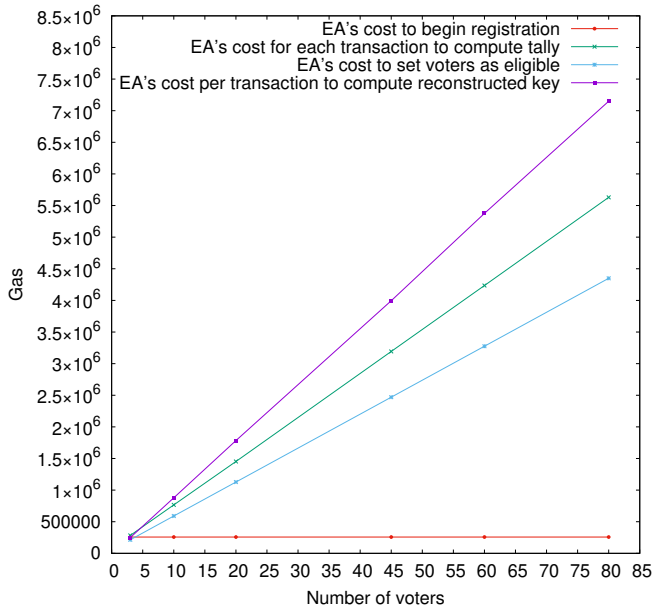


Figure 4.4: The gas costs for different tasks of the election administrator (EA) based on the number of voters participating in the election when the number of candidates competing in the election is 5.

competing in the election. The task 'Tally' in Table 4.10 includes calculating the sum of all the cast votes and then getting the tally for each candidate by brute-force (exhaustive searching). The time required for these tasks is shown in Table 4.10 for different numbers of candidates contesting in the election.

The cryptography contract 'CryptoContract' is used to create all zero-knowledge proofs using the .call() function. There is no need to send the transactions to this smart contract. The time required to create the schnorr's ZKP in the first round of the protocol depends on the number of candidates competing in the elections, as shown in Table 4.10.

## 4.8 Discussion on applications

Ranking based voting systems are popular among voting experts as they tend to gather more information from the voters with regard to their preferences as opposed to simple plurality voting where a voter is able to make only a single choice among a set of candidates. It is well known that a plurality voting system can produce election results that do not reflect the true sentiments of the voters [102]. An example of the fact is the 1988 election for

| Action | A | B | C | D |
|---|---|---|---|---|
| Create ZKP($X$) | 34 | 48 | 75 | 91 |
| Register voting keys | 57 | 84 | 104 | 138 |
| Begin election | 204 | 305 | 401 | 493 |
| Create $k$ 1-out-of-k ZKP | 213 | 515 | 908 | 1465 |
| Cast vote | 207 | 483 | 824 | 1261 |
| Tally | 162 | 385 | 457 | 694 |

Table 4.10: A timing measurement for different functions that run on Ethereum daemon. Here $X$ is a $k$-tuple $(x_1, x_2, ..., x_k)$, where $k$ is the number of candidates competing in the election. The $k$ in the Create-1-out-of-k ZKP represents the number of candidates competing in the election. The number of participants is eighty. Columns are represented as the average time in milliseconds for A: 2 candidates, B: 3 candidates, C: 4 candidates, D: 5 candidates.

electing the prime minister in Canada [102]. The key issue in this election was free trade with the USA. Approximately 60% of the population was opposed to free trade in this three-candidate election. However, the pro free trade candidate won the election since the anti free trade votes got split between two anti free trade candidates. Such pitfalls of the plurality voting system are caused due to the electoral system taking only limited information from voters: only a favorite candidate is indicated. These pitfalls can be avoided by taking more information from voters, e.g., using a ranking-based voting system [102].

Borda count is a typical ranking based voting system that elects a winner taking into account a voter's degree of proclivity toward one or more candidates. Thus, Borda count is sometimes described as a consensus-based voting system [92]. One of the good features of Borda Count is that it is "monotone", as increasing the score for a candidate only helps them win [41]. Hence, this voting system more faithfully reflects the sentiment of the electors than common plurality voting systems. However, these schemes are not heavily deployed in practice due to the complexity involved with them. The complexity around ranking based voting systems have so far led people into shying away from using them in real life. Our paper proposes an easily implementable Borda count e-voting system, and it brings ranking based voting systems closer to practice. Our scheme is publicly verifiable and it guarantees the privacy of every voter to the maximum such that only a full-collusion can break it. Moreover, the scheme does not rely on any trusted third parties. This scheme could

motivate researchers to explore new applications in the field of ranking-based e-voting, this previously uncharted territory. Below we provide a list of areas where the Borda Count voting method has been applied.

The Borda count voting has been used to aggregate preferences in many contexts [17]. In addition to democratic elections ([51, 114]), the Borda count voting scheme has been used in elections by several academic institutions and professional bodies [51, 61]. For instance, Borda count is used by X.Org Foundation to elect its board of directors [20]. The Borda count is also used for granting awards in several sports competitions (such as Most Valuable Player Award, Heisman Trophy etc. [104]) and singing competitions (such as Eurovision Song Contest [104]), soccer competitions (such as the RoboCup autonomous robot soccer competition [41]). Borda count is also used by the OpenGL Architecture Review Board as one of the feature-selection methods. It is used as a rank aggregation method for the Web (where voters are the search engines, and candidates are the pages). A summary of these applications of Borda count voting can be found in [148, 44]. In [83], Kijazi and Kant used Borda count method to establish group preferences for alternatives for forest use on Mount Kilimanjaro. Laukkanen et al. [88] and Hiltunen et al. [66] applied several voting methods including Borda count to assess group preferences for forest management plans in Finland. In [21], Burgman et al. discuss potential utility of various voting systems including Borda count for environmental decision making. Borda count is used for Waste Management in several countries [135]. It is used in TOPSIS (technique for order performance by similarity to ideal solution) ranking [124, 126] and an extension of TOPSIS for group decision making [125]. In another context, the Borda count method is widely used for rank aggregation in the information retrieval area [43]. Chatzichristofis et al. [28] propose an image retrieval technique using Borda count.

The public verifiability and freeness from any tallying authorities make our proposed Borda count system suitable for deployment over an Ethereum-like blockchain as we have demonstrated in Section 4.7. An e-voting system based on blockchain can be effective for corporate governance and shareholder activism [87, 140]. In February 2016, Nasdaq, in cooperation with the Estonian Government, announced a blockchain based e-voting that allows shareholders to vote remotely in Annual General Meetings (AGMs) [19, 87]. Besides Nas-

daq, the Abu Dhabi Stock Exchange used blockchain based e-voting to organise shareholder voting in annual general meetings [65, 87]. Compared with these blockchain-based e-voting systems, ours does not require any tallying authorities, so the tallying and the verification of the tallying integrity can be done publicly by the consensus algorithm that underpins the blockchain.

## 4.9 Concluding remarks

In this paper, we have proposed a two-round self-tallying Borda count e-voting scheme. This scheme does not require any trusted party to compute the tally. Instead, the scheme ensures that anyone can compute the tally from the public information made available on the bulletin board. Our scheme ensures the maximum voter privacy, and upon the successful completion of the protocol, the voters are strictly limited to learn only the tally of the election and their own inputs. We have presented security proofs to prove the security of the protocol. Further, the scheme offers public verifiability. Every voter generates NIZK proofs to prove that they have been faithfully following the protocol specification without revealing their secret input. We have implemented the scheme on the Ethereum blockchain. Both the theoretic and the experimental analysis results show that this scheme is feasible to be used in practice. In future work, we plan to investigate extending this work to support more complex ranked choice voting systems such as STV and Condorcet in a decentralized setting.

## 4.10 Open source code

The source code for the proof-of-concept implementation of the proposed Borda count voting system over the Ethereum blockchain can be found at `https://github.com/smartcontract68/Borda_count_smart_contract`.

# Chapter 5

# Deniable Secret Handshake Protocol - Revisited

## 5.1 Introduction

This chapter is based on paper [107]. The concept of deniability is defined following the simulation based paradigm. An adversary tries to convince a third party (judge) by producing a *proof* of a conversation held between two authorized users of the same organization. An secret handshake (SH) protocol is said to be deniable if a simulator can generate the same view of the protocol transcript which is indistinguishable from the real view. Mainly, two types of deniability exist in the literature - *full* deniability and *strong* deniability. In a fully deniable SH protocol, an adversary's view should be indistinguishable from the view of a simulator when two honest parties faithfully perform a SH protocol. We note that the simulator is able to produce the view without the honest users' certificates and CA's master secret key. However, the notion of strong deniability takes care of the scenario when one of the users is malicious and "acts" as the aforementioned adversary to trap an honest user with whom he is communicating. The definition of strong deniability requires that the simulator must be fed with the same inputs as the malicious party/ adversary which includes secret certificates and randomness of the adversary. Now if the simulator can produce an indistinguishable view from the real view then the protocol is said to be strongly deniable. For more details, we refer to the paper of [132].

In this chapter, we consider the situation when a user behaves semi-honestly and participates in an SH protocol with an honest user. A semi-honest user is not completely honest

but follows the protocol as an honest user. However, he may store the randomness that he uses in the protocol and may try to gather more information from the run of the protocol than an honest user would. We first show that in presence of a semi-honest user, the protocol of Tian et al. [132] is not fully deniable. More specifically, if the responder of a SH protocol is semi-honest and maintains receipt(s) then a simulator is unable to generate a transcript which is indistinguishable from the transcript generated during the real execution of the protocol- resulting in the loss of full deniability for the initiator. We then propose a possible countermeasure to fix the issue. We furthermore present two attacks- viz. "man-in-the-middle" (MITM) attack and "cutting-last-message-attack" to the protocol. We conclude the paper with solutions to resist these two types of attacks.

## 5.2 Preliminaries

**Online and Offline Judge:** When we discuss deniability, we must do so with respect to a type judge (or distinguisher). Two primary types of judges have been discussed in the secure messaging literature such as offline judges, and online judges.

An offline judge examines the transcripts of a protocol execution that occurred in the past and decides whether or not the parties mentioned in the transcript were actually involved in the conversation. An online judge interacts with a protocol participant, referred to as the informant, while the protocol conversation is occurring.

### 5.2.1 Security model

In this section, we introduce the security model due to [132]. The linkable affiliation-hiding (LAH) and untraceability properties of their model follow directly from [69, 94] respectively.

**States:** Let us define a user set $\mathcal{U}$ of $n$ users. The $i$-th session established by a user U is denoted by $\Pi_U^i$. An oracle $\Pi_U^i$ may be considered as used or unused. An oracle $\Pi_U^i$ is considered as unused if it has never been initialized. When an oracle $\Pi_U^i$ is initialized, it becomes part of a group. After initialization, the oracle is marked as used. The internal states $state_U^i$ are stored by the oracle. The oracle $\Pi_U^i$ *accepts* and *terminates* the protocol as soon as the session key $K_U^i$ is computed. The oracle stops sending or receiving messages

after terminating the protocol. If the protocol execution fails, the oracle terminates without having accepted.

**Partnering:** We denote the pseudonym of all users recognized by the $i$-th session $\Pi_U^i$ as $pid_U^i$ and the corresponding session identifier as $sid_U^i$. Two instance oracles $\Pi_U^i$ and $\Pi_{U'}^j$ are partners if and only if $pid_U^i$=$pid_{U'}^j$ and $sid_U^i$=$sid_{U'}^j$.

### 5.2.2 System model

A deniable secret handshake scheme is a collection of the following algorithms.

**Setup:** On input of a security parameter $\kappa$, the algorithm outputs public parameters *params*.

**KeyGen:** This algorithm is executed by the central authority (CA). The algorithm, on input *params*, outputs the group public key $mpk$, corresponding secret key $msk$ and an empty pseudonym revocation list $\mathcal{L}$.

**Add:** This algorithm is an interactive algorithm between the CA and a user. The algorithm, on input $msk$ and a user $U \in \mathcal{U}$, generates a public pseudonym $pk$ and the corresponding secret certificate *cert* for the user $U$. The CA updates the group pseudonym list by adding public pseudonym $pk$. The user will be registered after executing this algorithm.

**Revoke:** This algorithm is executed by CA. On input $pk$ from a user, the algorithm updates the group pseudonym revocation list $\mathcal{L}$ by adding the public pseudonym $pk$ of the user.

**Handshake:** This is an interactive algorithm executed by some set of participants $\Delta = \{U_1, U_2, ..., U_n\} \subseteq \mathcal{U}$. Each user $U_i$ runs the session $\Pi_i^\tau$ of the protocol on some inputs: $(pk_i^\tau, cert_i^\tau)$, $mpk_i^\tau$ and $\mathcal{L}$, where $(pk_i^\tau, cert_i^\tau)$ is the public pseudonym/secret certificate pair of user $U_i$, $mpk_i^\tau$ is the group public key in $U_i$'s view and $\mathcal{L}$ is the group pseudonym revocation list. The algorithm outputs a session key $SK$ if and only if her counter part users are registered and non-revoked users, otherwise the the algorithm rejects.

### 5.2.3 Session key security

Now we describe the adversarial behavior. An adversary is allowed to fully control the communication network. The adversary can inject, modify, or delete messages at will. He

can also launch man-in-the-middle attack. He can corrupt some users and obtain their secret keys, internal states and session keys. A session $\Pi_U^t$ is secure if an adversary can not obtain the established session keys unless it is compromised trivially through party corruption. We consider the same session key security model and security definition as described in Tian et al. [132]. The session key security model is defined via a game between a probabilistic polynomial time adversary $A$ and a challenger (i.e. simulator) $S$.

-Setup: The simulator $S$ generates the group public and secret key pair $(mpk_j, msk_j)$ for $m$ groups (CA), $\forall j \in \{1, 2, ..., m\}$. $S$ creates $n$ users and their corresponding public and secret key pairs $(pk_i, cert_i^j)$ in group $\mathbb{G}_j$, $\forall i \in \{1, 2, ..., n\}$, $\forall j \in \{1, 2, ..., m\}$, by executing the corresponding Add algorithm. Then $S$ sends all the public keys $(mpk_j, pk_i)$ to the adversary $A$. $S$ then generates a random coin $b$ that will be used later in the game. The set all registered and nonusers in the group $\mathbb{G}_j$ is denoted by $\mathcal{U}$.

-Training: $A$ can make following queries to the simulator $S$ in arbitrary order.

- Establish: $A$ can register a user $U^*$ with public pseudonym $pk^* \in \mathbb{G}_\beth$. If the user $U^*$ is registered by $A$ , then it will be dishonest and the user $U^*$ with public pseudonym $pk^*$ will added to the system.

- Send: If $A$ makes a send query of the form $(U_i, \mathbb{G}_j, t, m')$ to simulate a message $m'$ in the $t$-th session in group $\mathbb{G}_j$, then the simulator $S$ will simulate the reaction of the oracle $\Pi_{U_i}^t$ and returns the message that the oracle $\Pi_{U_i}^t$ would generate. If $A$ makes a send query of the form $(U_i, \mathbb{G}_\beth, t, \text{`}start')$, then $S$ creates a new instance oracle $\Pi_{U_i}^t$. Then $S$ returns the first protocol message to $A$.

- Long-term secret key reveal: If $A$ makes a long-term secret key reveal query to user $U_i$, then $S$ returns $(cert_i^j)$ to $A$.

- Ephemeral secret key reveal: If $A$ makes an ephemeral secret key reveal query to $\Pi_{U_i}^t$ (possibly in an unaccepted session), then $S$ returns all the ephemeral secret keys contained in the oracle $\Pi_{U_i}^t$ to $A$.

- Group secret key reveal: If $A$ makes an group secret key reveal query with respect to the group $\mathbb{G}_j$, then $S$ returns $msk_j$ to $A$.

- Session key reveal: If $A$ makes a session key reveal query to $\Pi_{U_i}^t$ for an accepted session, then $S$ returns the session key to $A$ if the session is accepted; otherwise, it returns a special symbol '$\perp$' to $A$.

- Test: This query can only be issued to a user $U$ in group $\mathbb{G}_j$ for an accepted and fresh session $t$. The simulator $S$ performs the following action:

  1. If $b = 1$, then the simulator $S$ returns the real session key to $A$.

  2. Otherwise, $S$ returns a random key from the session key space to $A$.

  The test session must remain fresh throughout the entire game. Note that $A$ can issue other queries after the test query.

At the end, $A$ outputs $b'$ as her guess for $b$. If $b = b'$, then the simulator outputs 1; else the simulator $S$ outputs 0.

**Freshness:** An accepted instance $\Pi_{U_i}^t$ will be fresh if the adversary $A$ does not make any of the following queries during the entire game.

- $A$ makes an establish query for a new user $U^* \in pid_U^t$.

- $A$ makes both the long-term secret key reveal query to user $U^* \in pid_U^t$ and the ephemeral secret key reveal query for some partner instance $\Pi_{U^*}^{t^*}$ of the instance $\Pi_{U_i}^t$.

- Before the acceptance of the instance $\Pi_{U_i}^t$, $A$ makes a long-term secret key reveal query to a user $U^*$ , where $U^* \in pid_U^i$, and there does not exist any instance $\Pi_{U^*}^{t^*}$ partnered with $\Pi_{U_i}^t$.

- $A$ makes a session key reveal query to an instance $\Pi_{U_i}^t$ or an instance $\Pi_{U^*}^{t^*}$ that is partnered with $\Pi_{U_i}^t$.

It can be realized that the group secret key reveal query to CA is equivalent to making long-term secret key queries to all users in group $\mathbb{G}_j$. The advantage of the adversary $A$ in this game is defined as

$$Adv_A(\kappa) = |Pr[S \rightarrow 1] - 1/2| \tag{5.2.1}$$

**Definition 5.2.1.** *A deniable secret handshake protocol is said to be session key secure if, for any probabilistic polynomial time adversary A, the advantage of the adversary* $Adv_A(\kappa)$

*is a negligible function of $\kappa$ (where $\kappa$ is the security parameter).*

*Generic Concurrent Knowledge Extraction Assumption (GCKEA):* We follow the same assumption used in Tian et al. This is a generalized version of Concurrent KEA [145] and Knowledge of Pairing Pre-Image Assumption (KPA) [121].

**Definition 5.2.2.** *(GCKEA) Let us define a domain $\{\mathcal{D}_\kappa\}_{\kappa\in\mathbb{N}}$, where $\mathbb{N}$ is the set of all natural numbers. Let $D$ be a set randomly chosen from $\mathcal{D}_\kappa$ and $p(\kappa)$, $p(\kappa)$ are two polynomials in the security parameter $\kappa$. Let $O_C$ be a predicate algorithm with respect to the random challenge set $C = \{C_1, C_2, ..., C_{p(\kappa)}\}$. On a query of the form $(X, Y, Z)$, where $(X, Y)$ is randomly chosen from the domain $D$, it outputs success (or 1) if $Y$ is chosen (randomly) from $C$ and $Z = PKDF(X, Y)$, where $PKDF$ is a public key derivation function. We define an algorithm $A^{O_C}$ with predicate oracle $O_C$, which, on input $C$, outputs $\{(X_1, Y_1, Z_1), (X_2, Y_2, Z_2), ..., (X_{q(\kappa)}, Y_{q(\kappa)}, Z_{q(\kappa)})\}$. An algorithm $A^{O_C}$ is called a GCKEA extractor if, with overwhelming probability, $A^{O_C}(C)$ outputs $\{(X_1, Y_1, Z_1), (X_2, Y_2, Z_2), ..., (X_{q(\kappa)}, Y_{q(\kappa)}, Z_{q(\kappa)})\}$ such that $Y_i \in C$ and $Z_i = PKDF(X_i, Y_i), \forall i \in \{1, 2, ..., q(\kappa)\}$.*

*We say that GCKEA holds if for every probabilistic polynomial time algorithm $A$, there exists another probabilistic algorithm $A^*$ that on the same inputs as $A$, random coins, random oracles, outputs $x_i$ such that $X_i = f(x_i), \forall i \in \{1, 2, ..., q(\kappa)\}$, where $f$ is a computationally efficient function that, on input $x_i$, outputs $X_i$.*

## 5.3 Revisiting DSH Protocol of Tian et al. [132]

### 5.3.1 The Protocol

Tian et.al. have introduced the notion of deniable secret handshake framework DSH [132]. For the sake of completeness, we describe the protocol almost verbatim. Their proposed generic framework consists of the following building blocks.

1. A forward-secure secret handshake protocol SH=(Setup, KeyGen, Add, Revoke, Handshake),

2. A blind digital signature scheme BS=(KeyGen, Signer and User, Verify),

3. A public key based key derivation function PKDF,

4. A proof of knowledge PoK and

5. A collision-resistant hash function H.

The DSH protocol given in [132] is as follows.

**Setup:** Given a security parameter $\lambda$ as input, the Setup algorithm outputs public parameters: $params \leftarrow$ SH.Setup.

**KeyGen**: The CA runs the SH.KeyGen algorithm to obtain the group public/secret key pair $(mpk, msk)$ and an empty pseudonym revocation list $\mathcal{L}$.

**Add:** The CA and user $\hat{A}$ run the BS.Signer and User(msk) interactive algorithm to obtain a pseudonym/certificate pair $(pk_a, cert_a)$ of user $\hat{A}$. User $\hat{A}$ takes $pk_a$ as public pseudonym.

**Revoke:** The group CA runs the SH.Revoke($pk_a$) algorithm to update the group pseudonym revocation list $\mathcal{L}$. Note that public pseudonym $pk_a$ is added to revocation list $\mathcal{L}$.

**Handshake:**

- User $\hat{A}$ runs the SH.Handshake.Ephemeral algorithm to obtain ephemeral secret/public key pair $(esk_a, epk_a)$ and sends $(epk_a, pk_a)$ to user $\hat{B}$;

- After receiving $(epk_a, pk_a)$ from user $\hat{A}$, user $\hat{B}$ performs the following steps:

  1. Run the SH.Handshake.Ephemeral algorithm to obtain ephemeral secret and public key pair $(esk_b, epk_b)$;

  2. Compute the proof of knowledge PoK$\{(esk_b) : H(PKDF(epk_b, epk_a))\}$;

  3. Send $(epk_b, pk_b, PoK(esk_b))$ to user $\hat{A}$.

- Upon receiving $(epk_b, pk_b, PoK(esk_b))$ from user $\hat{B}$, user $\hat{A}$ computes the proof of knowledge (i.e., non-malleable zero-knowledge)
  PoK$\{(esk_a, cert_a) : H(PKDF(epk_a, epk_b)||PKDF(pk_a, epk_b))\}$ and sends it to user $\hat{B}$. Meanwhile, $\hat{A}$ computes the final session key $SK_a = H(K_a||sid)$, where $K_a = SH.Handshake.KDF(esk_a, epk_b, cert_a, mpk, L, init)$ and the session identifier is $sid = (epk_a||epk_b)$.

- Upon receiving PoK$(esk_a, cert_a)$ from user $\hat{A}$, user $\hat{B}$ computes the proof of knowledge PoK$\{(esk_b, cert_b) : H(PKDF(epk_b, epk_a)||PKDF(pk_b, epk_a))\}$ and sends it to user

$\hat{A}$. Meanwhile, $\hat{B}$ computes the final session key $SK_b = H(K_b||sid)$, where $K_b = SH.Handshake.KDF(esk_b, epk_a, cert_b, mpk, L, resp)$. We notice that $K_a = K_b$ holds due to the correctness of SH.Handshake algorithm.

### 5.3.2 Analysis of the Protocol

According to the protocol, at the beginning of the handshake session, the user $\hat{A}$ sends her public pseudonym $pk_a$ to the user $\hat{B}$. In the next step, upon receiving $pk_a$ from the $\hat{A}$, a semi-honest user $\hat{B}$ can generate her ephemeral secret and public key pair $(esk_b, epk_b)$ and can keep a receipt of its randomness and keys. User $\hat{B}$ can show this receipt to a judge to prove their conversation. As a result of this attack, a simulator can not generate the transcript that is indistinguishable from the transcript during real protocol execution. The next subsection describes an example of such flaw. We note that such a flaw shares a similar idea as given in [55].

**Receipt Flaw:** Suppose the user $\hat{A}$ is an honest user and $\hat{B}$ is a semi-honest user. They execute the deniable secret handshake protocol and communicate with each other following the protocol specification. Meanwhile, user $\hat{B}$ keeps the receipt of its randomness whenever he receives a request from the initiator user $\hat{A}$ with public key $pk_a$. Later if only the receipt is obtained by the police then the full deniability will be lost for both the users. This is a flaw in the system in the sense that it is easy for user $\hat{B}$ to keep a receipt depending on the public key in the first message.

To accomplish this task, upon receiving $(epk_a, pk_a)$ from user $\hat{A}$, user $\hat{B}$ chooses a value uniformly at random from $\mathbb{Z}_q^*$ for generating ephemeral public key/secret key pair and keeps a receipt of the randomness to prove how she generated the public key/secret key pair. Since user $\hat{B}$ keeps a receipt, the exact ephemeral public key/secret key of user $\hat{B}$ can be generated later from that receipt. However, whenever $\hat{B}$ receives a pseudonym different from $pk_a$, she chooses a value uniformly at random from $\mathbb{Z}_q^*$ for generating her ephemeral public key/secret key pair and may not keep a receipt. In future, police approaches to judge with $\hat{B}$'s receipt to prove the conversation with user $\hat{A}$ whose pseudonym is $pk_a$.

In the above scenario: (1) a simulator $S$ can not simulate the transcript of the ephemeral

public/secret keys generated by user $\hat{B}$ since user $\hat{B}$ maintains a receipt of randomness and the exact ephemeral public/secret key pair of $\hat{B}$ can be generated from that receipt. This implies that the simulator will no be able to simulate the second message of the handshake protocol $(epk_b, pk_b, PoK(esk_b))$. (2) A simulator $S$ can not generate the proof of knowledge (i.e., non-malleable zero-knowledge) PoK$\{(esk_a, cert_a) : H(PKDF(epk_a, epk_b)||PKDF(pk_a, epk_b))\}$ in the third message without having the secret certificate, $cert_a$, of user $\hat{A}$ and the secret key of the semi-honest user $\hat{B}$. As user $\hat{B}$ keeps the receipt of randomness, the exact real transcript of this third message can be generated from that receipt. Thus, user $\hat{A}$ can not deny of having handshake session with user $\hat{B}$ i.e. breaking full deniability property of the protocol.

**Man-in-the-middle (MITM) attack:** In this attack, we consider two honest users $\hat{A}$ and $\hat{B}$ trying to execute the deniable secret handshake protocol [132]. However, upon observing the public pseudonym $pk_a$, a man-in-the-middle adversary (MITM) $\hat{M}$ tries to interfere in their communication. We now present this attack on DSH, which is depicted in Fig. 5.1. Since the PKDF is assumed to be a public key bases key derivation function, let us, for example, assume PKDF$\{(epk_a, epk_b)\} = g^{xy}$ if $epk_a = g^x$ and $epk_b = g^y$.

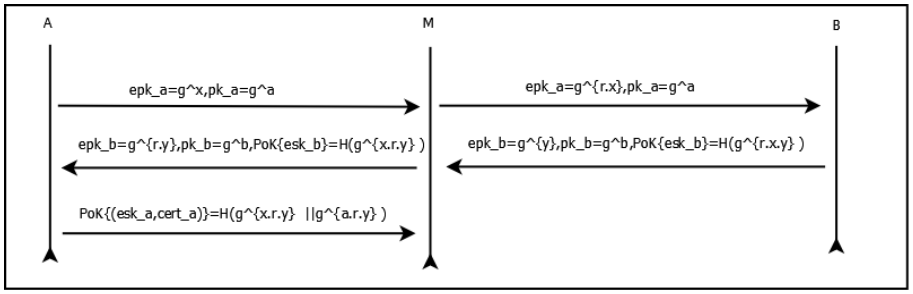


Figure 5.1: Full deniability loss of user $\hat{A}$ under man-in-the-middle (MITM) adversary attack.

Here, user $\hat{A}$ and user $\hat{B}$ are honest users. In this scenario, the MITM adversary $M$ keeps a receipt of its random variable $r$ and presents it to an offline or online judge. The MITM adversary $M$ communicates with the honest user $\hat{A}$ in the name of user $\hat{B}$ as shown on the left session of the Fig 5.1. The right session of the Fig. 5.1 depicts the communication between the MITM initiator $M$ with the honest responder $\hat{B}$ in the name of user $\hat{A}$.

Upon receiving the first round message $(epk_a, pk_a)$ (for example, $(g^x, g^a)$) from $\hat{A}$, the adversary $M$ chooses a number uniformly at random from $\mathbb{Z}_q^*$ and sends $((epk_a)^r, pk_a)$ (for example, $(g^{rx}, g^a)$) to user $\hat{B}$ in the name of $\hat{A}$. The adversary keeps a receipt of its randomness $r$. Upon receiving the second round message $(epk_b, pk_b, PoK\{esk_b : H(PKDF(epk_b, (epk_a)^r))\})$ (for example, $(g^y, g^b, H(g^{rxy}))$) from honest user $\hat{B}$, the adversary modifies the message to $((epk_b)^r, pk_b, PoK\{esk_b : H(PKDF(epk_b, epk_a))\})$ (for example, $(g^{ry}, g^b, H(g^{rxy}))$) and sends it to user $\hat{A}$ in the name of user $\hat{B}$. In the third round, the adversary receives the message $PoK\{(esk_a, pk_a) : H(PKDF(epk_a, epk_b)||PKDF(pk_a, epk_b))\}$ (for example, $H(g^{rxy}||g^{ray})$) from honest user $\hat{A}$ and aborts the conversation with both user $\hat{A}$ and user $\hat{B}$. Note that in these conversations, the adversary $M$ does not need to do any oracle query. After these conversations, the adversary $\hat{M}$ approaches to judge with her receipt to prove that the user $\hat{A}$ whose pseudonym is $pk_a$ is involved in the conversation.

In the above scenario, in case of honest user $\hat{A}$ and $\hat{B}$, a simulator S can not generate the proof of knowledge (i.e., non-malleable zero-knowledge) PoK$\{(esk_a, cert_a) : H(PKDF(epk_a, epk_b)||PKDF(pk_a, epk_b))\}$ in the third message without having the secret certificate, $cert_a$, of user $\hat{A}$ and the random number $r$ generated by the adversary $M$. Since the adversary does not make any oracle query, the simulator can not extract the exponent $r$ chosen by the adversary $M$. Therefore, The user $\hat{A}$ losses deniability in this case.

**Cutting-last-message attack:** Since the PKDF is assumed to be a public key based key derivation function, we assume PKDF$\{(epk_a, epk_b)\} = g^{xy}$ if $epk_a = g^x$ and $epk_b = g^y$.

This attack works as follows. A man-in-the-middle adversary $M$ interacts with the uncorrupted user $\hat{A}$ in the name of user $M \neq \hat{B}$ in a session (referred to as first session), while concurrently communicating with the uncorrupted user $\hat{B}$ in the name of user $\hat{A}$ in another session (referred to as second session). The adversary $M$ relays the messages between user $\hat{A}$ and user $\hat{B}$ in these two sessions, but aborts the first session after receiving last message from the user $\hat{B}$ in the second session.

Such an attack results in authentication failure as follows: the user $\hat{B}$ is perfectly fooled to believe that the user $\hat{B}$ has taken a part in conversation with user $\hat{A}$ in the second session and shared a session key with that user. However, user $\hat{A}$ thinks that it only took part in

an aborted session with user $M$ in the first session. The Fig. 5.2 depicts this attack.

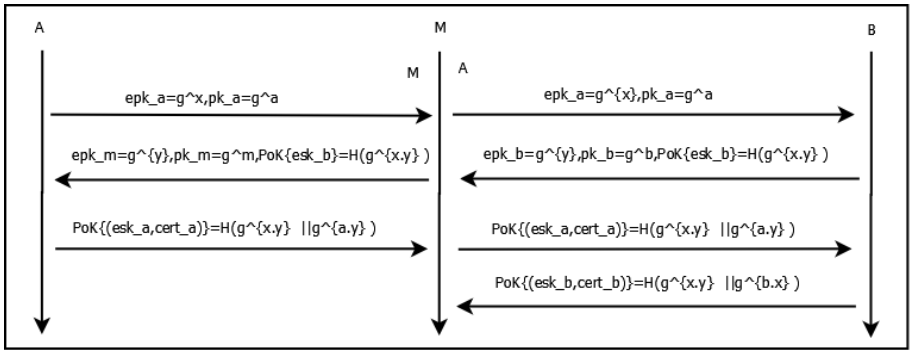


Figure 5.2: "cutting-last-message" attack.

Such an attack can be ruled out by adding $pk_b$ in the proof-of-knowledge in the second message, such as PoK$\{(epk_b) : H(pk_b||PKDF(epk_b, epk_a))\}$ or by including $pk_b$ in the PKDF function. Specifically, after receiving the second message $(epk_b, pk_b, PoK\{esk_b : H(pk_b||PKDF(epk_b, epk_a))\})$ from user $\hat{B}$, the adversary can not send correctly $(epk_m, pk_m, PoK\{esk_b : H(pk_m||PKDF(epk_b, epk_a))\})$, where $epk_m$ is the ephemeral public key of adversary $M$, $pk_m$ is the public pseudonym of adversary $M \neq \hat{B}$.

### 5.3.3 Possible countermeasures

**Solution: Using public random oracle model (pRO)**

*(Public) Random Oracle:* Random oracle [8] $H : \{0,1\}^* \rightarrow \{0,1\}^k$ is a *random function* such that it has following two properties: $(i)$ it is a function such that the same input gives the same output, and $(ii)$ it is completely random in the sense that for any input $x$, $H(x)$ is uniformly distributed over $\{0,1\}^k$. A random oracle (RO) can be described as follows.

Let us consider a set $L \subset \{0,1\}^* \times \{0,1\}^k$. Initially, $L = \phi$. For an input $x$, the value of $H(x)$ is computed as follows. First, check whether there exists $y \in \{0,1\}^k$ such that $(x, y) \in L$. If there is no such $y$, choose a random $y \leftarrow \{0,1\}^k$ and add the tuple $(x, y)$ into the set $L$. In any case, return $y$ as the computed value $H(x)$.

Our protocol in this section is proven to be deniably secure in the public random oracle (pRO) model. In this model, the random oracle is a public random function that is accessible by the adversary and the simulator by submitting input and receiving output. The simulator can see the input and output pairs of all random oracle queries. This type of oracle is

introduced in [111] for proving deniable zero knowledge. Note that deniability means the simulator's code can also be executed by the adversary himself and thus, he can simulate the transcript in adversary's view without actually interacting with honest parties. In this case, the adversary is the only entity interacting with the public random oracle (The adversary can forward the simulator's query to the public random oracle) and he can feed to the simulator with the oracle input/output. Thus, the simulation under the public random oracle model can be replayed by the adversary. However, in the traditional random oracle model [8], the simulator maintains the random oracle evaluation. Now if the simulator's code is executed by the adversary, then the random oracle evaluation is maintained by the adversary since the simulator is his subroutine. However, in the real protocol, the adversary can only submit input to the random oracle and receive the output. Therefore, in the traditional random oracle model, the simulator's code can not be played by an adversary and thus, the simulation is not guaranteed to be deniable.

**The protocol:** Let $H : \{0,1\}^* \rightarrow \{0,1\}^k$ be a hash function. We have described the modified procedure below under the pRO model described above. The protocol uses following components: (1) a forward secure secret handshake protocol, (2) a blind signature algorithm BS=(KeyGen, Signer and User, Verify); (3) a public key derivation function PKDF, a proof of knowledge PoK and a collision- resistant hash function $H$. For simplicity, in this section, we have described the protocol in the two-party setting using user $\hat{A}$ and user $\hat{B}$. It can be extended to multi-party setting using BD protocol [22].

– Setup: On input of a security parameter $\kappa$, this algorithm outputs public parameters $params \leftarrow$ SH.Setup.

– KeyGen: This algorithm is executed by the group CA. The algorithm, on input *params*, outputs the group public key $mpk$, corresponding secret key $msk$ and an empty pseudonym revocation list $\mathcal{L}$.

– Add: This algorithm is an interactive algorithm between the CA and a user. The algorithm, on input $msk$ and a user $\hat{A}$, executes the BS.Signer and User(msk) interactive algorithm ([75]) to generate a public pseudonym $pk_a$ and the corresponding secret certificate $cert_a$ for the user $\hat{A}$. User $\hat{A}$ takes $pk_a$ as public pseudonym.

– Revoke: This algorithm is executed by CA. On input $pk_a$ from a user $\hat{A}$, the algorithm

executes the SH.Revoke($pk_a$) algorithm to update the group pseudonym revocation list $\mathcal{L}$ by adding the public pseudonym $pk_a$ of the user $\hat{A}$.

– Handshake:

- User $\hat{A}$ executes the SH.Handshake.Ephemeral algorithm that outputs the ephemeral secret/public key pair $(esk_a, epk_a)$ for user $\hat{A}$. User $\hat{A}$ sends $epk_a$ to user $\hat{B}$.

- Upon receiving $epk_a$ from user $\hat{A}$, user $\hat{B}$ performs the following steps.

  1. It executes the SH.Handshake.Ephemeral algorithm to get ephemeral secret and public key pair $(esk_b, epk_b)$;

  2. It computes the proof of knowledge PoK$\{(esk_b) : H(pk_b||PKDF(epk_b, epk_a))\}$;

  3. User $\hat{B}$ sends $(epk_b, pk_b, PoK(esk_b))$ to user $\hat{A}$.

- Upon receiving $(epk_b, pk_b, PoK(esk_b))$ from user $\hat{B}$, user $\hat{A}$ evaluates the proof of knowledge (i.e., non-malleable zero-knowledge)
  PoK$\{(esk_a, cert_a) : H(PKDF(epk_a, epk_b)||PKDF(pk_a, epk_b))\}$ and sends it along with public pseudonym $pk_a$ to user $\hat{B}$. $\hat{A}$ calculates the final session key $SK_a = H(K_a||sid)$, where $K_a = SH.Handshake.KDF(esk_a, epk_b, cert_a, mpk, L, init)$ and the session identifier $sid = (epk_a||epk_b)$.

- Upon receiving PoK$(esk_a, cert_a)$ and $pk_a$ from user $\hat{A}$, user $\hat{B}$ computes the proof of knowledge PoK$\{(esk_b, cert_b) : H(PKDF(epk_b, epk_a)||PKDF(pk_b, epk_a))\}$ and sends it to user $\hat{A}$. Meanwhile, $\hat{B}$ calculates the final session key $SK_b = H(K_b||sid)$, where $K_b = SH.Handshake.KDF(esk_b, epk_a, cert_b, mpk, L, resp)$ and and the session identifier $sid = (epk_a||epk_b)$. The correctness of the protocol i.e. the equation $K_a = K_b$ holds due to the correctness of the underlying SH.Handshake algorithm.

**Theorem 5.3.1.** *The proposed generic framework achieves session key security in the public random oracle model provided the underlying secret handshake protocol SH is session key secure.*

*Proof.* To prove this theorem, let us define five games $G_i, \forall i \in \{0, 1, 2, 3, 4\}$, and analyze the advantage of an adversary in those games. We denote the advantage of the adversary

in game $G_i$ by $Adv_i^{GF}$, $i \in \{0, 1, 2, 3, 4\}$. Let the adversary $A$ generates at most $m$ sessions in each game.

- Game $G_0$: This first game is the original game for session key security.

- Game $G_1$: This game is same as $G_0$ except the following difference: $S$ will output a random bit if the initiator $i$ and the responder $j$ accept, however $pid_i \neq pid_j, sid_i \neq sid_j$. This game captures the replay attack i.e. no probabilistic polynomial time adversary can find the hash collision of the hash function $H$. Let $\kappa$ be the security parameter and $n$ be the number of users in the game. Then we have:

$$|Adv_0^{GF} - Adv_1^{GF}| \leq n.m^2/2^{\kappa} \tag{5.3.1}$$

- Game $G_2$: This game is same as $G_1$ except that $S$ chooses a session $t$ from $\{1, 2, .., m\}$ randomly as a guess for the test session. However, if $A$'s test query does not in the $t$-th session, $S$ will output a random bit. In this game, we have

$$Adv_1^{GF} = m.Adv_2^{GF} \tag{5.3.2}$$

- Game $G_3$: This game is same as $G_2$ except the following difference: the real SH session key is replaced by a random value from the session key space in the $t$-th session. $S$ does this replacement in the $t$-th session. We prove that, if the underlying SH protocol is session key secure, then the difference between the advantages of the adversary in game $G_2$ and $G_3$ is negligible.

  Let us consider an attacker $S$ against the underlying SH protocol, who wishes to distinguish between a real session key and a random value taken from session key space. $S$ is given the corresponding oracles. The game for $A$ is simulated by $S$ as described below.

  - Setup: $S$ generates the group public and secret key pair $(mpk, msk)$ for group CA. $S$ creates $n$ users and their corresponding public and secret key pairs $(pk_i, cert_i), \forall i \in$

$\{1, 2, ..., n\}$, from her oracle queries (i.e. long-term secret key query). Then $S$ sends all the public keys to $A$.

- $A$ can make queries to $S$. $S$ answers the queries as described below.

1. If $A$ makes a send query of the form $(epk_j)$ to user $i$, then $S$ simulates the answer as follows. $S$ forwards $(epk_j)$ to her challenger and get $(epk_i, pk_i)$ from her send oracle and $PoK(esk_j)$ from her public random oracle. It returns $(epk_i, pk_i, PoK(esk_j))$ as answer to $A$.

If $A$ makes a send query of the form $(pk_j, PoK(esk_j, cert_j))$ to user $i$, then $S$ simulates PoK$\{(esk_i, cert_i) : H(PKDF(epk_i, epk_j)||PKDF(pk_i, epk_j))\}$ by randomly choosing a hash value and returns it to $A$ as answer. Then $S$ obtains the a session key $K_i$ for SH protocol from her session key reveal oracle or test oracle, where $K_i = SH.Handshake.KDF(esk_i, epk_j, cert_i, mpk, L, resp)$. $S$ computes the final session key $SK_i = H(K_i||sid)$, where $sid = ((epk_j||epk_i))$. Note that $A$ cannot make both the ephemeral secret key reveal query and the long-term secret key reveal query in the test session.

If $A$ makes a send query of the form $(U', \mathbb{G}, t, `start')$, then $S$ creates a new instance oracle $\Pi^t_{U'}$. Then $S$ chooses $(epk')$ and returns it to $A$.

2. If $A$ makes a long-term secret key reveal query to $S$, then $S$ returns $(cert_i)$ to $A$.

3. If $A$ makes an ephemeral secret key reveal query to $S$, then $S$ makes an ephemeral secret key reveal query to its own oracle to get the ephemeral secret key $(esk_i)$ and sends it to $A$.

4. If $A$ makes an group secret key reveal query to $S$, then $S$ returns $msk$ to $A$.

5. If $A$ makes a session key reveal query or test query, then $S$ returns $SK_i$ that it has computed in the protocol simulation process above.

6. If $A$ makes an establish query in the form $(U^*, pk^*)$ for a user $U^*$ with public pseudonym $pk^* \in \mathbb{G}$, then the user $U^*$ with public pseudonym $pk^*$ will be added to the system.

It can be observed that $S$ will receive either the real session key or a random session

key from its own oracle. If $S$ gets the real session key, then this simulation is consistent with $G_2$; Otherwise, if $S$ gets the random key, then this simulation is consistent with $G_3$. Hence, we prove that if the difference the advantages of the adversary in game $G_2$ and game $G_3$, $S$ can beak the session key security of the underlying SH protocol. Therefore, we get

$$|Adv_2^{GF} - Adv_3^{GF}| \leq Adv_S^{SH}(\kappa) \tag{5.3.3}$$

- Game $G_4$: This game is same as $G_3$ except the following: in the test session, the public random oracle $H(K_i||sid)$ is replaced by a random function. We prove that the difference of the advantages between the game $G_3$ and $G_4$ is negligible if $H$ is in the pseudo random function (RO) family.

  Let us consider an attacker $S$ against the RO that wishes to distinguish between $H(K_i||sid)$ and a random value from the key space. $S$ is given the an oracle either $H$ or a random function. The game for $A$ is simulated by $S$ in the same way as the game $G_3$ except that $S$ sends the $sid$ to its challenger and assigns the returned value as its session key $SK_i$. $S$ sets its output whatever $A$ outputs. As per this simulation, if the oracle is $H$ then this simulation is same as the game $G_3$. On the other hand, if the selected oracle is the random function, then this simulation is same as the game $G_4$. Hence, if $A$ can distinguish between the game $G_3$ and $G_4$, then $S$ can break the RO. Therefore, we have

$$|Adv_3^{GF} - Adv_4^{GF}| \leq Adv_S^{RO}(\kappa) \tag{5.3.4}$$

  It can be realized that there is no advantage for $A$ in game $G_4$ i.e.

$$Adv_4^{GF} = 0 \tag{5.3.5}$$

From equations 5.3.1, 5.3.2, 5.3.3, 5.3.4 and 5.3.5, we have
$Adv_A^{GF}(\kappa) = n.m^2/2^{\kappa} + m.(Adv_S^{SH}(\kappa) + Adv_S^{RO}(\kappa))$, where RO is a pseudo random

function.

Hence, the proposed generic framework is session key secure in the public random oracle model if the underlying SH protocol is session key secure.

□

**Discussions on deniability:** We outline the main idea of the proof of deniability. To prove the deniability, a simulator needs to simulate the answer of *send*, *session key reveal*, *test* and *ephemeral secret key reveal*, *long-term secret key reveal* and *group secret key reveal* queries such that the adversary's view in the simulated transcript is indistinguishable from the real transcript, while the simulator should not use any uncorrupted secret keys. The difficult part is to answer the send query. Consider a *send* query of the form $(U_i, G_j, s, m)$ to simulate a message $m$. Let us assume $m = H(g^s||K)$; where $K$ is publicly known, $s$ is a secret, $g$ is an publicly known element of the considered group and '||' is the concatenate operation. Therefore, $(g^s||K)$ must have been queried to the oracle $H$ to be consistent with the input $(g^s||K)$ and output $m$, that can be verified by the simulator since he sees all the input/output for all $H$ oracle queries. Otherwise, the message $m$ is inconsistent with probability $1 - negligible(\kappa)$ if $(g^s||K)$ is not queried to the oracle, where $negligible(\kappa)$ is a negligible function in security parameter $\kappa$ (i.e. for any positive polynomial $p(n)$, there exists integer $n_0 > 0$ such that $negligible(n) > (1/p(n))), \forall n > n_0$.). Thus, if $(g^s|K)$ is queried to the oracle, the simulator can extract $s$ by the generic concurrent knowledge extraction assumption (GCKEA). This assumption is also used in [132](a similar assumption is used in [145]). Thus, the simulator can answer adversary's *send* queries. Similarly, the simulator can answer other *send* queries.

**Theorem 5.3.2.** *Let $H$ be a public random oracle. Then the secret handshake protocol described above is deniable.*

*Proof.* Here we describe an overview of the simulation procedure for deniability analysis focusing on the tricks of using GCKEA assumption. The detailed proof is similar to the proof given in [145].

**High-level description:** We first consider a left session between an honest initiator $\hat{(}A)$ and a malicious responder $\hat{B}$. The malicious responder may be an Man-In-The-Middle

adversary $A$. In this case, the simulator simulates the transcripts of the protocol conversations as follows. The simulator chooses an ephemeral secret key $esk_a$ and computes $epk_a = g^{esk^a}$ by itself. The simulator sends $epk_a$ as the first-round message of the protocol. Then the initiator $\hat{(}A)$ receives $(epk_b, pk_b, PoK(esk_b))$ from the adversary $A$ as the second-round message of the protocol, where PoK$\{(esk_b) : H(pk_b||PKDF(epk_b, epk_a))\}$. To compute PoK$\{(esk_b) : H(pk_b||PKDF(epk_b, epk_a))\}$, with overwhelming probability, the adversary $A$ has queried the random oracle $H$ with $(pk_b, epk_b, epk_a, PKDF(epk_b, epk_a))$. Now, the value $esk_b$ can be extracted by the GCKEA assumption. Then using the value $esk_b$, the third-round message PoK$\{(esk_a, cert_a) : H(PKDF(epk_a, epk_b)||PKDF(pk_a, epk_b))\}$ (along with $pk_a$) can be generated using public random oracle $H$ without knowing the long-term secret key $cert_a$ of the honest initiator $\hat{(}A)$. If the adversary $A$ finishes the session successfully, she sends PoK$\{(esk_b, cert_b) : H(PKDF(epk_b, epk_a)||PKDF(pk_b, epk_a))\}$ as the fourth-round message. In this case, with overwhelming probability, the adversary $A$ has queried the random oracle $H$ with $(epk_b, epk_a, PKDF(epk_b, epk_a), pk_b, PKDF(pk_b, epk_a))$ , from which the long-term secret key $cert_b$ can be extracted by GCKEA assumption. Now, if the session finishes successfully, the session key can be computed using the secret keys $esk_b$ and $cert_b$. Therefore, in this case, the simulator can simulate the protocol transcripts without using the honest initiator $\hat{(}A)$'s long-term secret key $cert_a$. Hence, the honest initiator $\hat{(}A)$ can deny her participation in the protocol execution and her messages.

Now, we consider a right session between a malicious initiator $\hat{(}A)$ and an honest responder $\hat{B}$. The malicious initiator may be an Man-In-The-Middle adversary $A$. In this case, the simulator simulates the transcripts of the protocol conversations as following. $A$ sends $epk_a$ as the first-round message. Upon receiving $epk_a$ from $A$, the simulator chooses the ephemeral secret key $esk_b$ and computes $epk_b = g^{esk_b}$ and PoK$\{(esk_b) : H(pk_b||PKDF(epk_b, epk_a))\}$. Then the simulator sends $(epk_b, pk_b, PoK(esk_b))$ as the second-round message. If $A$ finishes the protocol successfully, she sends PoK$\{(esk_a, cert_a) : H(PKDF(epk_a, epk_b)||PKDF(pk_a,$ $epk_b))\}$ along with $pk_a$ as the third round message. To compute PoK$\{(esk_a, cert_a) :$ $H(PKDF(epk_a, epk_b)||PKDF(pk_a, epk_b))\}$, with overwhelming probability, the adversary $A$ has queried the oracle $H$ with $(epk_a, epk_b, PKDF(epk_a, epk_b), pk_a, PKDF(pk_a, epk_b))$, from which both the long-term secret key $cert_a$ and ephemeral secret key $esk_a$ can be ex-

tracted by GCKEA assumption. Using these extracted keys $cert_a$ and $esk_a$, the simulator can generate fourth-round message $\text{PoK}\{(esk_b, cert_b) : H(PKDF(epk_b, epk_a)||PKDF(pk_b, epk_a))\}$ without knowing the honest responder $\hat{(}B)$'s long-term secret key $cert_b$. Now, if the session finishes successfully, the session key can be computed using the secret keys $esk_a$ and $cert_a$. Therefore, in this case, the simulator can simulate the protocol transcripts without using the honest responder $\hat{(}B)$'s long-term secret key $cert_b$. Hence, the honest responder $\hat{(}B)$ can deny her participation in the protocol execution and her messages.

Similarly, we can prove the deniability of both the initiator $\hat{(}A)$ and responder $\hat{(}B)$ when both the protocol participants are honest.

□

**Note 1 (Unlinkability):** This solution does not satisfy the unlinkability property since the public pseudonym is being sent by each user in every session. Thus, these sessions are linkable.

## 5.4 Concluding remarks

We have discussed some possible attacks to the fully deniable secret handshake protocol of Tian et al. [132]. We mention that there may be a subjective issue about the model of deniability. However, this paper opens up possibilities to scrutinize definitions for deniability in the context of secret handshake. Due to the fully deniable property, this deniable secret handshake protocol could also be used as a coercion resistant internet-based voting system. We think that this could be a potential future work.

# Chapter 6

# Conclusion

Democracy relies on voters having well-founded trust in the election process. Unfortunately, several case studies on real-world e-voting systems [61] provide abundant grounds for skepticism. E-voting systems need to be designed in such a way that it can produce sufficient evidence preserving the privacy of each voter to convince rationally skeptical observers that the election outcome is correct. Advances in end-to-end verifiable e-voting have the potential to restore trust in election and thereby improve the democracy. In this thesis, we have proposed some secure E2E verifiable (both precinct-based and internet-based) e-voting systems. We have traced the development of privacy, verifiability and security properties in the literature and discussed the current state-of-the-art E2E verifiable e-voting systems.

## 6.1 Future directions of research

In chapter 3, we propose a secure E2E verifiable e-voting system using blockchain and cloud server. We also propose a novel voter registration and authentication method. The proposed system prevents ballot stuffing attack. We provide the security proofs of our systems. We show one weakness of the DRE-ip system and propose a solution. Thereafter, we prove the the efficient NIZK proof algorithm proposed by Lin et al. [91] is not correct. We then improve the efficiency of NIZK proofs and provide security proofs of the proposed NIZK proof algorithm. The performance analysis of our protocol implementation shows potential for real-world deployment of our proposed system. Designing DRE-based e-voting system without tallying authorities for more complex voting system such as single transferable vote (STV) is a potential future work.

In chapter 4, we propose the first verifiable self-tallying decentralized Borda Count e-voting protocol. The security proofs of our proposed are also provided. The protocol is implemented as a smart contract in Ethereum. Our implementation of the protocol demonstrates the feasibility of using Ethereum for secure and publicly verifiable Borda count voting. Designing an efficient self-tallying E2E verifiable protocol for nation-wide Borda Count voting is part of our future work. In future, we also plan to design self-tallying protocols for more complex rank-choice based voting system such as STV and Condorcet in decentralized setting and investigate the feasibility of storing the ballots in Ethereum state channel to avoid the gas cost of running the protocol on-chain.

In chapter 5, we propose a deniable secret handshake protocol. We show some possible attacks to the fully deniable secret handshake protocol of Tian et al. [132] and propose solutions to prevent them. Due to the fully deniable property, our protocol can be useful in developing coercion resistant e-voting system over the Internet, which is a potential future work.

# List of Publications

- Somnath Panja and Bimal Kumar Roy. A secure end-to-end verifiable e-voting system using zero knowledge based blockchain. *IACR Cryptol. ePrint Arch.*, 2018:466, 2018.
- Somnath Panja and Bimal Kumar Roy. A secure end-to-end verifiable e-voting system using blockchain and cloud server. *Under Submission*, 2020.
- Somnath Panja, Samiran Bag, Feng Hao, and Bimal Kumar Roy. A smart contract system for decentralized borda count voting. *IEEE Trans. Engineering Management*, 67(4):1323-1339, 2020.
- Somnath Panja, Sabyasachi Dutta, and Kouichi Sakurai. Deniable secret handshake protocol - revisited. In Leonard Barolli, Makoto Takizawa, Fatos Xhafa, and Tomoya Enokido, editors, *Proceedings of the 33rd International Conference on Advanced Information Networking and Applications, volume 926 of Advances in Intelligent Systems and Computing*, pages 1266–1278. Springer, 2019.

# Bibliography


[1] Ben Adida. Helios: Web-based Open-audit Voting. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 335–348. USENIX Association, 2008.

[2] Ben Adida, Olivier De Marneffe, Olivier Pereira, Jean-Jacques Quisquater, et al. Electing a university president using open-audit voting: Analysis of real-world use of Helios. *EVT/WOTE*, 9(10), 2009.

[3] Ben Adida and Ronald L. Rivest. Scratch & Vote: Self-contained Paper-based Cryptographic Voting. In *Proceedings of the 5th ACM Workshop on Privacy in Electronic Society*, WPES '06, pages 29–40. ACM, 2006.

[4] Adam Back. Hashcash - a denial of service counter-measure. `http://www.hashcash.org/papers/hashcash.pdf`, 2002. Accessed on 15th Dec., 2020.

[5] Samiran Bag, Muhammad Ajmal Azad, and Feng Hao. E2E Verifiable Borda Count Voting System without Tallying Authorities. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–9. ACM, 2019.

[6] Dirk Balfanz, Glenn Durfee, Narendar Shankar, Diana K. Smetters, Jessica Staddon, and Hao-Chi Wong. Secret Handshakes from Pairing-Based Key Agreements. In *IEEE Symposium on Security and Privacy*, pages 180–196. IEEE Computer Society, 2003.

[7] Mihir Bellare and Oded Goldreich. On Defining Proofs of Knowledge. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer, 1992.

[8] Mihir Bellare and Phillip Rogaway. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73. ACM, 1993.

[9] Jonathan Ben-Nun, Niko Fahri, Morgan Llewellyn, Ben Riva, Alon Rosen, Amnon Ta-Shma, and Douglas Wikström. A New Implementation of a Dual (Paper and Cryptographic) Voting System. In Manuel J. Kripp, Melanie Volkamer, and Rüdiger Grimm, editors, *5th International Conference on Electronic Voting 2012*, volume P-205 of *LNI*, pages 315–329. GI, 2012.

[10] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM, 1988.

[11] Josh Benaloh. Verifiable Secret-Ballot Elections. `https://www.microsoft.com/en-us/research/publication/verifiable-secret-ballot-elections/`, September 1987.

[12] Josh Benaloh. Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. In Ray Martinez and David A. Wagner, editors, *USENIX/ACCURATE Electronic Voting Technology Workshop*. USENIX Association, 2007.

[13] Josh Benaloh, Michael D. Byrne, Bryce Eakin, Philip T. Kortum, Neal McBurnett, Olivier Pereira, Philip B. Stark, Dan S. Wallach, Gail Fisher, Julian Montoya, Michelle Parker, and Michael Winn. STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. In *2013 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections*. USENIX Association, 2013.

[14] Josh Benaloh and E. Lazarus. The Trash Attack: An Attack on Verifiable Voting Systems and a Simple Mitigation. Technical Report MSR-TR-2011-115, Microsoft, 2011.

[15] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (ex-

tended abstract). In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 544–553. ACM, 1994.

[16] Josh Cohen Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters (extended abstract). In Joseph Y. Halpern, editor, *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing*, pages 52–62. ACM, 1986.

[17] Duncan Black. *The Theory of Committees and Elections*. University Press, Cambridge, 1958.

[18] Jens-Matthias Bohli, Jörn Müller-Quade, and Stefan Röhrich. Bingo voting: Secure and coercion-free voting using a trusted random number generator. In Ammar Alkassar and Melanie Volkamer, editors, *E-Voting and Identity, First International Conference, VOTE-ID*, volume 4896 of *Lecture Notes in Computer Science*, pages 111–124. Springer, 2007.

[19] P. Boucher. What if blockchain technology revolutionised voting? `http://www.europarl.europa.eu/RegData/etudes/ATAG/2016/581918/EPRS_ATA(2016)581918_EN.pdf`, Sept. 2016. Accessed on 14th Oct., 2019.

[20] Robert Bredereck, Jiehua Chen, Piotr Faliszewski, André Nichterlein, and Rolf Niedermeier. Prices Matter for the Parameterized Complexity of Shift Bribery. *Inf. Comput.*, 251(C):140–164, December 2016.

[21] Mark A. Burgman, Helen M. Regan, Lynn A. Maguire, Mark Colyvan, James Justus, Tara G. Martin, and Kris Rothley. Voting Systems for Environmental Decisions. *Conservation Biology*, 28(2):322–332, 2014.

[22] Mike Burmester and Yvo Desmedt. Efficient and Secure Conference-Key Distribution. In T. Mark A. Lomas, editor, *Security Protocols, International Workshop*, volume 1189 of *Lecture Notes in Computer Science*, pages 119–129. Springer, 1996.

[23] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '97, pages 410–424. Springer-Verlag, 1997.

[24] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy. In *19th USENIX Security Symposium*, pages 291–306. USENIX Association, 2010.

[25] Claude Castelluccia, Stanislaw Jarecki, and Gene Tsudik. Secret Handshakes from CA-Oblivious Encryption. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 293–307. Springer, 2004.

[26] Ann Cavoukian and Alex Stoianov. Biometric Encryption: The New Breed of Untraceable Biometrics. *Biometrics: Theory, Methods, and Applications*, pages 655–718, 2009.

[27] Ann Cavoukian and Alex Stoianov. Encryption, biometric. In Stan Z. Li and Anil K. Jain, editors, *Encyclopedia of Biometrics*, pages 260–269. Springer US, 2009.

[28] Savvas A. Chatzichristofis, Konstantinos Zagoris, Yiannis Boutalis, and Avi Arampatzis. A Fuzzy Rank-Based Late Fusion Method for Image Retrieval. In Klaus Schoeffmann, Bernard Merialdo, Alexander G. Hauptmann, Chong-Wah Ngo, Yiannis Andreopoulos, and Christian Breiteneder, editors, *Advances in Multimedia Modeling*, pages 463–472. Springer Berlin Heidelberg, 2012.

[29] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. Sherman, and P. Vora. Scantegrity: End-to-End Voter-Verifiable Optical- Scan Voting. *IEEE Security & Privacy*, 6(3):40–46, May 2008.

[30] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

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

[32] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity II: End-to-end Verifiability for Optical Scan Election Systems Using Invisible Ink Confirmation Codes. In *Proceedings of the Conference on Electronic Voting Technology*, EVT'08, pages 14:1–14:13, Berkeley, CA, USA, 2008. USENIX Association.

[33] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II: end-to-end verifiability by voters of optical scan elections through confirmation codes. *IEEE Trans. Inf. Forensics Secur.*, 4(4):611–627, 2009.

[34] Nikos Chondros, Bingsheng Zhang, Thomas Zacharias, Panos Diamantopoulos, Stathis Maneas, Christos Patsonakis, Alex Delis, Aggelos Kiayias, and Mema Roussopoulos. D-DEMOS: A Distributed, End-to-End Verifiable, Internet Voting System. In *36th IEEE International Conference on Distributed Computing Systems*, pages 711–720. IEEE Computer Society, 2016.

[35] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *26th Annual Symposium on Foundations of Computer Science*, pages 372–382. IEEE Computer Society, 1985.

[36] Kevin J Coleman and Eric A Fischer. The Help America Vote Act: Overview and Issues. `https://www.hsdl.org/?view&did=746115`, 2011, Accessed on 16th July, 2020.

[37] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.

[38] Lorrie Faith Cranor and Ron Cytron. Sensus: A security-conscious electronic polling system for the internet. In *30th Annual Hawaii International Conference on System Sciences (HICSS-30)*, pages 561–570. IEEE Computer Society, 1997.

[39] Chris Culnane, Peter Y. A. Ryan, Steve A. Schneider, and Vanessa Teague. vVote: A Verifiable Voting System. *ACM Trans. Inf. Syst. Secur.*, 18(1):3:1–3:30, 2015.

[40] Chris Culnane and Steve A. Schneider. A Peered Bulletin Board for Robust Use in Verifiable Voting Systems. In *IEEE 27th Computer Security Foundations Symposium*, pages 169–183. IEEE Computer Society, 2014.

[41] Jessica Davies, George Katsirelos, Nina Narodytska, Toby Walsh, and Lirong Xia. Complexity of and algorithms for the manipulation of Borda, Nanson's and Baldwin's voting rules. *Artificial Intelligence*, 217:20 – 42, 2014.

[42] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[43] D. Ding, Le Chen, J. Li, and B. Zhang. AP-Scored Borda Counting for Information Retrieval. In *The Proceedings of the Multiconference on "Computational Engineering in Systems Applications"*, volume 2, pages 1473–1478, Oct 2006.

[44] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank Aggregation Methods for the Web. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 613–622. Association for Computing Machinery, 2001.

[45] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.

[46] Peter Emerson. *Designing an All-Inclusive Democracy: Consensual Voting Procedures for Use in Parliaments, Councils and Committees.* Jan 2007.

[47] Aleksander Essex, Jeremy Clark, and Carlisle Adams. Aperio: High integrity elections for developing countries. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards*

*Trustworthy Elections, New Directions in Electronic Voting*, volume 6000 of *Lecture Notes in Computer Science*, pages 388–401. Springer, 2010.

[48] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 186–194, London, UK, 1987. Springer-Verlag.

[49] Kevin Fisher, Richard Carback, and Alan T. Sherman. Punchscan: Introduction and System Definition of a High-Integrity Election System. In *Workshop on Trustworthy Election. 2006*, 2006.

[50] Followmyvote. Follow my vote. `https://followmyvote.com/`, 2012. Accessed on 14th Oct., 2019.

[51] Jon Fraenkel and Bernard Grofman. The Borda Count and its real-world alternatives: Comparing scoring rules in Nauru and Slovenia. *Australian Journal of Political Science*, 49(2):186–205, 2014.

[52] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-Round Secure Multiparty Computation. In *Proceedings of the 22Nd Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '02, pages 178–193, Berlin, Heidelberg, 2002. Springer-Verlag.

[53] O. Goldreich, S. Micali, and A. Wigderson. How to Play ANY Mental Game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[54] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[55] Oded Goldreich and Hugo Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM J. Comput.*, 25(1):169–192, 1996.

[56] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[57] Jens Groth. Efficient Maximal Privacy in Boardroom Voting and Anonymous Broadcast. In Ari Juels, editor, *Financial Cryptography, 8th International Conference, FC 2004, Key West, FL, USA, February 9-12, 2004. Revised Papers*, volume 3110 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2004.

[58] Feng Hao, Dylan Clarke, and Avelino Francisco Zorzo. Deleting Secret Data with Public Verifiability. *IEEE Trans. Dependable Secur. Comput.*, 13(6):617–629, 2016.

[59] Feng Hao, Matthew N. Kreeger, Brian Randell, Dylan Clarke, Siamak F. Shahandashti, and Peter Hyun-Jeen Lee. Every Vote Counts: Ensuring Integrity in Large-Scale Electronic Voting. *USENIX Journal of Election Technology and Systems (JETS)*, (3):1–25, August 2014.

[60] Feng Hao, Peter Y. A. Ryan, and Piotr Zieliński. Anonymous voting by two-round public discussion. *IET Information Security*, 4:62–67, 2010.

[61] Feng Hao and Peter YA Ryan. *Real-World Electronic Voting: Design, Analysis and Deployment*. CRC Press, 2016.

[62] Feng Hao and Piotr Zielinski. A 2-Round Anonymous Veto Protocol. In Bruce Christianson, Bruno Crispo, James A. Malcolm, and Michael Roe, editors, *Security Protocols, 14th International Workshop*, volume 5087 of *Lecture Notes in Computer Science*, pages 202–211. Springer, 2006.

[63] Mark A Herschberg. Secure electronic voting over the world wide web. `http://web.mit.edu/people/hershey/thesis/all.pdf`, 1997.

[64] A. Hertig. The First Bitcoin Voting Machine Is On Its Way. `http://motherboard.vice.com/read/the-first-bitcoin-voting-machine-ison-its-way`, Nov. 2015. Accessed on 14th Oct., 2019.

[65] S. Higgins. Abu Dhabi Stock Exchange Launches Blockchain Voting. `http://www.coindesk.com/abu-dhabi-exchange-blockchain-voting/`, Oct. 2016. Accessed on 14th Oct., 2019.

[66] Veikko Hiltunen, Jyrki Kangas, and Jouni Pykäläinen. Voting methods in strategic forest planning - Experiences from Metsähallitus. *Forest Policy and Economics*, 10(3):117–127, 2008.

[67] Anil K. Jain, Karthik Nandakumar, and Abhishek Nagar. Biometric template security. *EURASIP J. Adv. Signal Process.*, 2008, 2008.

[68] Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Group Secret Handshakes Or Affiliation-Hiding Authenticated Group Key Agreement. In Masayuki Abe, editor, *Topics in Cryptology - CT-RSA*, volume 4377 of *Lecture Notes in Computer Science*, pages 287–308. Springer, 2007.

[69] Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Beyond Secret Handshakes: Affiliation-Hiding Authenticated Key Exchange. In Tal Malkin, editor, *Topics in Cryptology - CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 352–369. Springer, 2008.

[70] Stanislaw Jarecki and Xiaomin Liu. Private Mutual Authentication and Conditional Oblivious Transfer. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2009.

[71] Shaoquan Jiang and Reihaneh Safavi-Naini. An Efficient Deniable Key Exchange Protocol (Extended Abstract). In Gene Tsudik, editor, *Financial Cryptography and Data Security, 12th International Conference*, volume 5143 of *Lecture Notes in Computer Science*, pages 47–52. Springer, 2008.

[72] Hugo Jonker, Sjouke Mauw, and Jun Pang. Privacy and verifiability in voting systems: Methods, developments and trends. *Comput. Sci. Rev.*, 10:1–30, 2013.

[73] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine, editors, *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, pages 61–70. ACM, 2005.

[74] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections, New Directions in Electronic Voting*, volume 6000 of *Lecture Notes in Computer Science*, pages 37–63. Springer, 2010.

[75] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of Blind Digital Signatures (Extended Abstract). In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 1997.

[76] Ari Juels and Martin Wattenberg. A fuzzy commitment scheme. In Juzar Motiwalla and Gene Tsudik, editors, *CCS '99*, pages 28–36. ACM, 1999.

[77] Fatih Karayumak, Maina M. Olembo, Michaela Kauer, and Melanie Volkamer. Usability analysis of helios - an open source verifiable remote electronic voting system. In Hovav Shacham and Vanessa Teague, editors, *Electronic Voting Technology Workshop / Workshop on Trustworthy Elections*. USENIX Association, 2011.

[78] Yutaka Kawai, Shotaro Tanno, Takahiro Kondo, Kazuki Yoneyama, Kazuo Ohta, and Noboru Kunihiro. Extension of Secret Handshake Protocols with Multiple Groups in Monotone Condition. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 93-A(6):1122–1131, 2010.

[79] Dalia Khader, Ben Smyth, Peter Y. A. Ryan, and Feng Hao. A Fair and Robust Voting System by Broadcast. In Manuel J. Kripp, Melanie Volkamer, and Rüdiger Grimm, editors, *5th International Conference on Electronic Voting*, volume 205 of *LNI*, pages 285–299. GI, 2012.

[80] A. Kiayias, M. Korman, and D. Walluck. An Internet Voting System Supporting User Privacy. In *22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 165–174, Dec 2006.

[81] Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas

Zacharias. On the Security Properties of e-Voting Bulletin Boards. In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks - 11th International Conference*, volume 11035 of *Lecture Notes in Computer Science*, pages 505–523. Springer, 2018.

[82] Aggelos Kiayias and Moti Yung. Self-tallying Elections and Perfect Ballot Secrecy. In David Naccache and Pascal Paillier, editors, *PKC*, volume 2274 of *Lecture Notes in Computer Science*, pages 141–158. Springer, 2002.

[83] Martin Herbert Kijazi and Shashi Kant. Forest stakeholders' value preferences in Mount Kilimanjaro, Tanzania. *Forest Policy and Economics*, 12(5):357–369, 2010.

[84] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an Electronic Voting System. In *IEEE Symposium on Security and Privacy*, page 27. IEEE Computer Society, 2004.

[85] Ranjit Kumaresan and Iddo Bentov. How to Use Bitcoin to Incentivize Correct Computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 30–41. ACM, 2014.

[86] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash attacks on the verifiability of e-voting systems. In *IEEE Symposium on Security and Privacy*, pages 395–409. IEEE Computer Society, 2012.

[87] Anne Lafarre and Christoph Van der Elst. Blockchain Technology for Corporate Governance and Shareholder Activism. *SSRN Electronic Journal*, Jan 2018.

[88] Sanna Laukkanen, Teijo Palander, Jyrki Kangas, and Annika Kangas. Evaluation of the multicriteria approval method for timber-harvesting group decision support. *Silva Fennica*, 39(2):249–264, 2005.

[89] A. K. Lenstra and H. W. Lenstra, Jr. Handbook of Theoretical Computer Science (Vol. A). chapter Algorithms in Number Theory, pages 673–715. MIT Press, Cambridge, MA, USA, 1990.

[90] Samantha Levine. Hanging Chads: As the Florida Recount Implodes, the Supreme Court Decides Bush v. Gore, Jan. 17 2008. http://www.usnews.com/news/articles/2008/01/17/the-legacy-of-hanging-chads, Jan. 2008.

[91] Yikang Lin and Peng Zhang. Blockchain-based Complete Self-tallying E-voting Protocol. In *APSIPA ASC*, pages 47–52. IEEE, 2019.

[92] David Lippman. Voting Theory. *Math in Society*, 2013. Accessed on 17th January, 2020.

[93] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making Smart Contracts Smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 254–269, New York, NY, USA, 2016. ACM.

[94] Mark Manulis, Bertram Poettering, and Gene Tsudik. Affiliation-Hiding Key Exchange with Untrusted Group Authorities. In Jianying Zhou and Moti Yung, editors, *Applied Cryptography and Network Security, 8th International Conference*, volume 6123 of *Lecture Notes in Computer Science*, pages 402–419, 2010.

[95] Mark Manulis, Bertram Poettering, and Gene Tsudik. Taming Big Brother Ambitions: More Privacy for Secret Handshakes. In Mikhail J. Atallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies, 10th International Symposium*, volume 6205 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 2010.

[96] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. A Smart Contract for Boardroom Voting with Maximum Voter Privacy. In Aggelos Kiayias, editor, *Financial Cryptography and Data Security - 21st International Conference*, volume 10322 of *Lecture Notes in Computer Science*, pages 357–375. Springer, 2017.

[97] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[98] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf, 2008.

[99] Karthik Nandakumar, Anil K. Jain, and Sharath Pankanti. Fingerprint-Based Fuzzy Vault: Implementation and Performance. *IEEE Trans. Inf. Forensics Secur.*, 2(4):744–757, 2007.

[100] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, *CCS*, pages 116–125. ACM, 2001.

[101] C. Andrew Neff. Practical High Certainty Intent Verification for Encrypted Votes. Available from `http://citeseer.ist.psu`, 2004. Accessed on 17th Sept., 2020.

[102] Jill Van Newenhizen. The borda method is most likely to respect the condorcet principle. *Economic Theory*, 2(1):69–83, 1992.

[103] Valtteri Niemi and Ari Renvall. How to prevent buying of votes in computer elections. In Josef Pieprzyk and Reihaneh Safavi-Naini, editors, *Advances in Cryptology - ASIACRYPT '94*, volume 917 of *Lecture Notes in Computer Science*, pages 164–170. Springer, 1994.

[104] E. Niou and P.C. Ordeshook. *Strategy and Politics: An Introduction to Game Theory.* EBL-Schweitzer. Taylor & Francis, 2015.

[105] Jan Franz Palngipang, Miguel Luis Ting, and Rowel Atienza. BBCast: Cloud-Based Interactive Public Bulletin Board. In Khalid Al-Begain and Nidal AlBeiruti, editors, *9th International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 35–40. IEEE, 2015.

[106] Somnath Panja, Samiran Bag, Feng Hao, and Bimal Kumar Roy. A smart contract system for decentralized borda count voting. *IEEE Trans. Engineering Management*, 67(4):1323–1339, 2020.

[107] Somnath Panja, Sabyasachi Dutta, and Kouichi Sakurai. Deniable secret handshake protocol - revisited. In Leonard Barolli, Makoto Takizawa, Fatos Xhafa, and Tomoya Enokido, editors, *Proceedings of the 33rd International Conference on Advanced Information Networking and Applications*, volume 926 of *Advances in Intelligent Systems and Computing*, pages 1266–1278. Springer, 2019.

[108] Somnath Panja and Bimal Kumar Roy. A secure end-to-end verifiable e-voting system using zero knowledge based blockchain. *IACR Cryptol. ePrint Arch.*, 2018:466, 2018.

[109] Somnath Panja and Bimal Kumar Roy. A secure end-to-end verifiable e-voting system using blockchain and cloud server. *Under Submission.*, 2020.

[110] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, pages 248–259, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[111] Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 316–337. Springer, 2003.

[112] Birgit Pfitzmann and Andreas Pfitzmann. How to break the direct rsa-implementation of mixes. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques*, volume 434 of *Lecture Notes in Computer Science*, pages 373–381. Springer, 1989.

[113] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Deniable authentication and key exchange. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 400–409. ACM, 2006.

[114] Benjamin Reilly. Social Choice in the South Seas: Electoral Innovation and the Borda Count in the Pacific Island Countries. *International Political Science Review*, 23(4):355–372, 2002.

[115] C. Reitwiessner. Smart contract security. `https://blog.ethereum.org/2016/06/10/smart-contract-security/`, June 2016.

[116] P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia. Prêt á voter Voter:

a Voter-Verifiable Voting System. *IEEE Transactions on Information Forensics and Security*, 4(4):662–673, Dec 2009.

[117] Peter Y. A. Ryan and Vanessa Teague. Pretty good democracy. In Bruce Christianson, James A. Malcolm, Vashek Matyas, and Michael Roe, editors, *Security Protocols XVII, 17th International Workshop*, volume 7028 of *Lecture Notes in Computer Science*, pages 111–130. Springer, 2009.

[118] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology - EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer, 1995.

[119] Daniel Sandler, Kyle Derr, and Dan S. Wallach. VoteBox: A Tamper-evident, Verifiable Electronic Voting System. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium*, pages 349–364. USENIX Association, 2008.

[120] Daniel Sandler and Dan S. Wallach. Casting Votes in the Auditorium. In Ray Martinez and David A. Wagner, editors, *USENIX/ACCURATE Electronic Voting Technology Workshop*. USENIX Association, 2007.

[121] Sven Schäge. TOPAS: 2-Pass Key Exchange with Full Perfect Forward Secrecy and Optimal Communication Complexity. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1224–1235. ACM, 2015.

[122] C. P. Schnorr. Efficient Signature Generation by Smart Cards. *J. Cryptol.*, 4(3):161–174, January 1991.

[123] Siamak F. Shahandashti and Feng Hao. DRE-ip: A Verifiable E-Voting Scheme Without Tallying Authorities. In Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows, editors, *Computer Security - ESORICS*, volume 9879 of *Lecture Notes in Computer Science*, pages 223–240. Springer, 2016.

[124] Hsu-Shih Shih, Wen-Yuan Lin, and ES Lee. Group decision making for TOPSIS. In *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference (Cat. No. 01TH8569)*, pages 2712–2717. IEEE, 2001.

[125] Hsu-Shih Shih, Huan-Jyh Shyur, and E. Stanley Lee. An extension of TOPSIS for group decision making. *Mathematical and Computer Modelling*, 45(7):801 – 813, 2007.

[126] Hsu-Shih Shih, Chih-Hung Wang, and E.S. Lee. A multiattribute GDSS for aiding problem-solving. *Mathematical and Computer Modelling*, 39(11):1397 – 1412, 2004.

[127] Warren D. Smith. Three Voting Protocols: ThreeBallot, VAV, and Twin. In Ray Martinez and David A. Wagner, editors, *USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07*. USENIX Association, 2007.

[128] Yonatan Sompolinsky and Aviv Zohar. Secure High-Rate Transaction Processing in Bitcoin. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security - 19th International Conference*, volume 8975 of *Lecture Notes in Computer Science*, pages 507–527. Springer, 2015.

[129] C. Soutar, D. Roberge, A. Stoianov, R. Gilroy, and B. V. K. V. Kumar. Biometric encrpytion. In *ICSA Guide Cryptograp., New York:McGraw-Hill*, 1999.

[130] Douglas Stinson. *Cryptography: Theory and Practice,Second Edition*. CRC/C&H, 2nd edition, 2002.

[131] Pavel Tarasov and Hitesh Tewari. Internet Voting Using Zcash. *IACR Cryptology ePrint Archive*, 2017:585, 2017.

[132] Yangguang Tian, Yingjiu Li, Yinghui Zhang, Nan Li, Guomin Yang, and Yong Yu. DSH: Deniable Secret Handshake Framework. In Chunhua Su and Hiroaki Kikuchi, editors, *Information Security Practice and Experience - 14th International Conference, ISPEC 2018, Proceedings*, volume 11125 of *Lecture Notes in Computer Science*, pages 341–353. Springer, 2018.

[133] Yangguang Tian, Shiwei Zhang, Guomin Yang, Yi Mu, and Yong Yu. Privacy-Preserving k-time Authenticated Secret Handshakes. In Josef Pieprzyk and Suriadi

Suriadi, editors, *Information Security and Privacy - 22nd Australasian Conference*, volume 10343 of *Lecture Notes in Computer Science*, pages 281–300. Springer, 2017.

[134] George J Tomko, Colin Soutar, and Gregory J Schmidt. Fingerprint controlled public key cryptographic system, 1996. US Patent 5,541,994.

[135] Bojana Tot, Goran Vujić, Zorica Srđević, Dejan Ubavin, and Mário Augusto Tavares Russo. Group assessment of key indicators of sustainable waste management in developing countries. *Waste Management & Research*, 35(9):913–922, 2017.

[136] Gene Tsudik and Shouhuai Xu. A Flexible Framework for Secret Handshakes. In George Danezis and Philippe Golle, editors, *Privacy Enhancing Technologies, 6th International Workshop*, volume 4258 of *Lecture Notes in Computer Science*, pages 295–315. Springer, 2006.

[137] P.T. Tuyls, B. Skoric, and T.A.M. Kevenaar, editors. *Security with noisy data : on private biometrics, secure key storage and anti-counterfeiting*. Springer, Germany, 2007.

[138] Nik Unger and Ian Goldberg. Deniable Key Exchanges for Secure Messaging. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1211–1223. ACM, 2015.

[139] Nik Unger and Ian Goldberg. Improved Strongly Deniable Authenticated Key Exchanges for Secure Messaging. *Proc. Priv. Enhancing Technol.*, 2018(1):21–66, 2018.

[140] Christoph Van der Elst and Anne Lafarre. Blockchain and Smart Contracting for the Shareholder Community. *European Business Organization Law Review*, 20(1):111–137, Mar 2019.

[141] B. Wire. Now You Can Vote Online with a Selfie. `http://www.businesswire.com/news/home/20161017005354/en/VoteOnline-Selfie`, Oct. 2016. Accessed on 14th Oct., 2019.

[142] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger EIP-150 REVISION (759dccd - 2017-08-07), 2017. Accessed on: 2018-01-03.

[143] Naoyuki Yamashita and Keisuke Tanaka. Secret Handshake with Multiple Groups. In Jae-Kwang Lee, Okyeon Yi, and Moti Yung, editors, *Information Security Applications, 7th International Workshop*, volume 4298 of *Lecture Notes in Computer Science*, pages 339–348. Springer, 2006.

[144] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society, 1986.

[145] Andrew Chi-Chih Yao and Yunlei Zhao. Privacy-Preserving Authenticated Key-Exchange Over Internet. *IEEE Trans. Inf. Forensics Secur.*, 9(1):125–140, 2014.

[146] Moti Yung and Yunlei Zhao. Interactive Zero-Knowledge with Restricted Random Oracles. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 21–40. Springer, 2006.

[147] Filip Zagórski, Richard Carback, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L. Vora. Remotegrity: Design and use of an end-to-end verifiable remote voting system. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security - 11th International Conference, Proceedings*, volume 7954 of *Lecture Notes in Computer Science*, pages 441–457. Springer, 2013.

[148] Y. Zhang, W. Zhang, J. Pei, X. Lin, Q. Lin, and A. Li. Consensus-Based Ranking of Multivalued Objects: A Generalized Borda Count Approach. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):83–96, Jan 2014.

[149] Zhichao Zhao and T.-H. Hubert Chan. How to vote privately using bitcoin. In Sihan Qing, Eiji Okamoto, Kwangjo Kim, and Dongmei Liu, editors, *Information and Com-*

*munications Security - 17th International Conference*, volume 9543 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 2015.