Nullification,
a coercion-resistance add-on for e-voting protocols

Mahdi Nejadgholi

A Thesis
in
The Concordia Institute
for
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Master of Applied Science (Information and Systems Engineering) at
Concordia University
Montréal, Québec, Canada

August 2022

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mahdi Nejadgholi**

Entitled: **Nullification, a coercion-resistance add-on for e-voting protocols**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Information and Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

________________________ Chair and Examiner
Amr Youssef

________________________ Examiner
M. Mannan

________________________ Thesis Supervisor
Jeremy Clark

Approved by ________________________
Z. Patterson, Graduate Program Director

August 1, 2022 ________________________
Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Nullification,
a coercion-resistance add-on for e-voting protocols

Mahdi Nejadgholi

Coercion is one of the remaining issues on internet voting. Many developing countries are prone to this problem due to lower income rate. In this study, a novel coercion-resistant protocol has been proposed that can be integrated with previously proposed e-voting systems. We call it nullification. We present it as a part of the VoteXX e-voting protocol that has been designed and implemented through this study. Nullification gives the voter a strategic advantage over the coercer. The voter can share her keys with a trusted proxy, called a nullifier, for later flipping that vote. Integrity and ballot secrecy are provided simultaneously through the use of zero-knowledge proofs, specifically Σ-protocols. We show how our approach is different from (and potentially composable with) re-voting or panic password techniques that have been previously proposed in the academic literature.

Through designing this protocol, we solve several issues, design new Σ-protocols and protocols for the secure evaluation of basic logic functions like exclusive-or (xor) under encryption: *True XOR* and *Online XOR* have been proposed that improves the previously proposed Mix and Match protocol for secure multi-party computation of an arbitrary function under constrained input domain.

Acknowledgments

To

Roya, *the meaning of my life*, who would always be with me. Whatever that I emanate, would be her, shining through me.

Ghasem, my kind and generous father, **Vahedeh, Amin, Sara** and **Sam** who were the joy of my life and that bright dot that kept me fighting forward.

Mj, whom the words would shatter her perfection and peacefulness. She tripped me to corners of being, showed me the secrets of life, death and the vast world in between.

The House **Farahanis**[1], family members, brothers who had fought beside me in this dark era. North remembers.

Nafise, **Bahar** and **Arash**, kind friends that I could not survive without them.

Sir **Jeremy Clark**, a family member, a safe haven, a genius and a decent mentor. Researching under his supervision was a unique, delightful and adventurous journey.

Sir **Mohammad Mannan** and Sir **Amr Youssef**, kind, trustworthy and understanding faculties who have showed me, there is still hope in Academia.

Sir **David Chaum**, a beautiful mind, a kind soul who proved me that I know almost nothing, contained my wild spirit and believed in me.

Sir **Alan Sherman**, a keen researcher and the rook of the castle, and Sir **Bart Preneel**, a master-mind and a master-piece. I was so fortunate to learn from them weekly.

Dear **Rick Carback**, who is a colorful rainbow of various talents and emotions and dear **Mario Yaksetig**, my kind, knowledgeable and “dope” friend and colleague.

[1]except Jalal; Ali Agha included.

Contents

List of Figures **ix**

List of Tables **x**

1 Introduction **1**

1.1 A Brief History of Voting 1

1.2 Motivation 2

1.3 Contributions 3

1.4 Publications 3

1.5 Structure 4

2 Background **5**

2.1 Foundations of Security 5

2.1.1 Availability 6

2.1.2 Integrity 6

2.1.3 Confidentiality 6

2.1.4 Usability 6

2.2 E-voting Requirements Glossary 6

2.2.1 Functionality 7

2.2.2 Usability and Deployability 7

2.2.3 Correctness 7

2.2.4 Secrecy 8

2.2.5 Malfeasance Resistance 8
2.2.6 Coercion and Deception Resistance 8
2.3 A Pinch of Cryptography 9
2.3.1 Intractable Problems 10
2.3.2 ElGamal Cryptoscheme 11
2.3.3 Secure Multi-Party Protocols 12
2.3.4 PET, Plain Equality Test 13
2.3.5 Knowledge Proofs 14
2.3.6 Sigma Protocols 15
2.4 Threats to Online Voting Systems 19
2.4.1 Untrusted Platforms 19
2.4.2 Online Denial-of-Service Attacks 20
2.4.3 Unauthorized Intervention 20

3 VoteXX 24

3.1 Definitions 24
3.2 Entities 26
3.3 Assumptions and Thread Model 28
3.4 Phases 28
3.5 Protocol 29
3.6 Security Analysis (Sketch) 34
3.6.1 Registration 34
3.6.2 Voting 37
3.6.3 Provisional Tally 37
3.6.4 Nullification 38
3.6.5 Final Tally 39
3.7 Implementation 39

4 Nullification 42

4.1 Design . . . 43

4.1.1 Initial State . . . 43

4.1.2 Single-Voter Construction . . . 44

4.1.3 Receipt Verification . . . 54

4.1.4 Expanding to N Voters . . . 56

4.1.5 Aggregation . . . 57

4.1.6 Complexity . . . 61

4.2 Proof Sketch . . . 62

4.2.1 Terminology . . . 62

4.2.2 Completeness . . . 66

4.2.3 Special Soundness . . . 67

4.2.4 Special Honest Verifier Zero-Knowledge . . . 70

4.2.5 Fiat-Shamir Heuristics . . . 73

4.3 Improvements . . . 73

4.3.1 Caveats . . . 73

4.3.2 Secure Nullification . . . 75

4.3.3 Efficient Nullification . . . 81

5 Wrap-up 83

5.1 Discussions . . . 83

5.1.1 Generalization . . . 83

5.1.2 Nullification or Voting . . . 85

5.1.3 Nullification, not Revoting. . . . 85

5.1.4 Nullification and Liquid Democracy . . . 85

5.1.5 Game Theoretic Analysis . . . 86

5.2 Future Works . . . 87

5.2.1 Improving the Design and Implementation . . . 87

5.2.2 Compulsory Lubrication . . . 87

5.2.3 Literature Disambiguation 88

Bibliography **90**

List of Figures

1 Phases of VoteXX e-voting Protocol 31

2 Nullification Receipt Construction 45

3 Interactive version of Nullification Protocol 65

4 Nullification protocol using Single-bit Random Oracle 71

5 Optimizing mix and match XOR. 76

6 Mixing phase of Mix and Match protocol 78

7 True XOR. (a) initial-state (b) after a sample mixing 79

8 The inner structure of Online XOR function 79

9 Proof of shuffle of Online XOR evaluation table 80

10 The process of Topic Extraction over academic e-voting papers 88

11 WordCloud of the extracted topics 89

List of Tables

1 Bulletin Board Nullification table 58

2 Bulletin Board Nullification sub-table of encryption 59

3 Nullification Lookup table, a closer look 61

4 Number of exponentiations for each Nullification phase 62

Chapter 1

Introduction

"*Fear is the path to the Dark Side. Fear leads to anger, anger leads to hate, hate leads to suffering.*"

- Yoda

1.1 A Brief History of Voting

Surowiecki in *Wisdom of Crowds* [75] collected various exmaples of the benefits of *Collective Decision Making.* We observe the concept has been utilized in various fields of study; from genetic algorithms [30] to artificial brains [67] to take advantage of those benefits. A close term to collective decision making is *democracy.* Although the word literally means the governance of people, it has been misused to either empower or enslave nations, according to Graeber [31]. He argues that we are still far from a democratic world, where all lives matter. This study is a small effort in that direction.

The most common voting practice is still paper-based voting. However, there have been flaws in this voting method that pushed countries to try electronic voting systems [41]. Haitian elections is an example of corruption in paper-based voting, where the 2015 presidential election was invalidated due to fraudulent paper ballots used to stuff the ballot boxes [11]. The fraud led to lack of trust in Democratic process in Haiti [46]. This attack that is called *Ballot Stuffing* is still being used to exploit the integrity of nation-wide paper-based elections [7, 22].

Due to these evident vulnerabilities, pioneering countries such as Estonia, Switzerland and United States have moved forward to *electronic voting*[1] or even *online voting*[2] to reduce costs, generate faster tallies, improve usability, accommodate disable voters, and, in their view, implement better security model over the whole election process.

However, there is another problem even with the electronic or online voting process. Without a private ballot booth to cast their ballots, voters can get bribed or forced to vote in a particular way. In elections with a small margin of victory[3], an attacker could change the result of the election by buying a few hundred votes or coercing this amount of voters. Developing countries are more prone to vote buying due to the lower minimum wage [74]. Mass vote buying patterns have been observed in Nigeria [10], Argentina [74], Kenya [47, 55], Philippines [62]. The coercion problem is still an active concern and an open field of study.

1.2 Motivation

To promote democracy— that is author's main motivation— online voting is an interesting approach that allows inexpensive and more flexible elections. However, online voting has three big problems: Denial of Service, coercion, and untrusted

[1]or E-voting

[2]or Internet Voting

[3]For instance, Al Gore versus George W. Bush presidential election in 2000 that a five-week war over the result of U.S. presidential election in Florida ended up with a margin less than six hundred votes. Also, Minnesota Senate election in 2008 where Al Franken won the election with less than % 0.01 of the votes. In these cases, flipping of a few hundred votes could change the result of the election.

platforms. Without mitigating these three issues, online voting will be hindered in deployment or controversial if used. Of the three problems, we study the second because it is the furthest from being solved. Also we found use cases for fully coercion-free online election systems in developing countries and narrowly-margined elections (see Section 1.1). We will argue in this thesis that currently proposed solutions either do not work very well or assume too much of voters. We intended to propose a more usable and simple approach like re-voting (tell a friend to help) but lets you evade coercion at any time (unlike revoting) and works even if the adversary steals all your keys which is a realistic threat with malware or physical possession of devices.

1.3 Contributions

The key novel contributions of this study are:

1. Proposing the Nullification Σ-protocol that enables a voter's representative (*i.e.,* the *Nullifier*) to flip a vote without revealing any information about the voter.
2. Full implementation of the nullification protocol as a phase of VoteXX e-voting protocol.
3. Improving an existing generic protocol (Mix and Match) for the specific task of computing binary XORs under encryption.

1.4 Publications

Through this program, three peer-reviewed have been published which are listed below. This thesis is an expansion of the third paper with some references to the second paper.

1. A study of oracle approximation over deep learning test suites that was focused on measuring the test codes' mathematical errors in famous deep learning frameworks such as Tensorflow, Pytorch and Keras [52]. This appeared at IEEE/ACM ASE 2019.

2. A study of the ballot secrecy problem in liquid democracy systems where we pinpointed some theoretical limits to performing a secret-ballot liquid election [53]. This appeared at VOTING co-located with Financial Cryptography 2021.

3. A proposed coercion-resistant voting system called VoteXX [17]; accepted and to appear at E-VOTE-ID 2022. This thesis emphasizes my contributions to the project, as listed in Section 1.3.

1.5 Structure

Thesis structure is as follows. In *Chapter 2*, a brief introduction over the security and cryptographic background has been presented. *Chapter 3* is to explain the VoteXX voting system design as a coercion-resistant e-voting system. In *Chapter 4*, the construction of the baseline nullification module has been proposed, analyzed and improved. Finally *Chapter 5* has been devoted to future works in different directions on subjects around security improvements of online voting systems.

Chapter 2

Background

"*Cryptography is typically bypassed, not penetrated.*"

- Adi Shamir

First, let's look through the primary concepts of *Voting Security* and *Cryptography* and also some of the academic and real-world attempts to design a secure e-voting system.

2.1 Foundations of Security

Looking at the big picture, hacking an election is an attack with minimal casualties and huge influence. Insecure elections can lead to an incorrect result, in the case that it can not be detected or proven. Even in the case of detection, it would end up causing a scandal and discredit of both the election and the election authorities.

There have been different enumerations of the properties of a secure system, often demonstrate security as a triangle. In this study, we have used Bishop's security requirements [6] which are *Availability*, *Confidentiality* and *Integrity*. Let's have a short review of these crucial concepts.

2.1.1 Availability

Availability provides assurance that a system can be accessed in a timely manner when it is needed.

2.1.2 Integrity

Integrity is one of those umbrella terms in cryptography and hence there are various definitions of system *integrity* proposed by security experts over the years, all referring to the trustworthiness of the data and data origin. Also verifiability is another close term for integrity, We say a process is *verifiable* if there exists a way for users or public observers to ensure the different properties of the system, specifically *integrity*.

2.1.3 Confidentiality

Confidentiality or *secrecy* is a system property that ensures whether the secret information stays hidden for the promised period of time from unauthorized access.

2.1.4 Usability

Bishop's [6] reduction misses one critical concept. Until recently, usability has been often been neglected is the fundamental property of an interactive system. The scope of this study is to design a usable e-voting system that is 'secure-enough' in other aspects.

2.2 E-voting Requirements Glossary

Although there have been various proposed internet voting protocols, there is still debate on whether internet voting can be a good choice for implement a decision making process [59]. Different studies mentioned different sets of requirements to define a secure voting system. Here, we mention the most used terms with a small description and relationship with the three main security properties.

2.2.1 Functionality

Putting it simple, the e-voting system should work. There should be *decisive* and *indisputable* as Chaum mentions in his random sample voting paper [16].

2.2.2 Usability and Deployability

Chaum [16] also payed attention to different aspects of usability and deployability in the modern world: *High voter turnout*, *well-informed voters*, *effectiveness of results* and *low-cost systems* all are well-defined sub-categories. *Accessibility*—particularly support for voters with disabilities—is another related property that NIST mentioned in its 5 essential voting system requirements.

2.2.3 Correctness

The voting result must be **accurate**. It must be **tamper-proof**, *transparent* and *auditable*. The correctness requirement has been partitioned into two requirements, namely *capture correctness* and *counting correctness* by Rubin et al. in [68]. Popoveniuc et al. [65] went one step further and defined correctness for each phase of the election: named as *well-formed presented ballots*, *well-formed cast*, *recorded as cast*, and *tallied as recorded*. Also Popoveniuc mentioned *consistency* and *"recorded as cast" check* as other requirements that imply correctness. Being able to verify each *voter's eligibility* can also be categorized as a correctness property.

2.2.4 Secrecy

No one should be able to tell how a voter voted. Note that not all voting systems use *ballot secrecy*. Some e-voting systems are being designed to work for *open-ballot* systems. Others might hide how voters voted individually, but allow anyone to see a *running-tally* of the results as the ballots are coming in. For this thesis, we focus on strong ballot secrecy and no running tally.

2.2.5 Malfeasance Resistance

The system must be resistant to insider attacks. For this thesis, the main strategy is to decentralize the power over the election among mutually untrusted parties (*e.g.,* the candidate parties).

2.2.6 Coercion and Deception Resistance

We define *coercion* as a sequence of actions by an arbitrary player that forces the voter to vote in a particular way that the *coercer* dictates. Also, we define *deception* as the weaker version of *coercion* that the *coercer* incentives the voter to vote against her decision. The incentive might be positive (*e.g.,* in the form of financial benefit), negative (*e.g.,* harm), or neutral (*e.g.,* propaganda). Both *coercion* and *deception* can lead to the discredit of the election. We categorize *coercion* into four different categories:

- **Complete Coercion**. This is the situation that the coercer has a complete control for the total period of the election, giving the voter no chance to freely act. Hirt et al. [37] prove that such a coercer is indistinguishable from the voter and coercion resistance is not possible under this strong adversarial model.

- **Coercion With a Window.** In this case, voter would have a time window where they could take an action without coercion (*e.g.,* nullify their vote). The timing of this window is crucial and impacts the design of a coercion resistance mechanism.

- **No Coercion but Deception.** In this case we assume that the voter is not willing to sell a vote but based on some social or financial problems and a provided incentive from the coercer, they will cooperate with the coercer. In the literature it has been called vote-buying.

- **No Coercion, Weak Deception and Voter's Compliance.** Under this circumstance, the vote would be sold by the voter as she wants to discredit the election as a part of a protest or movement.

An ideal e-voting system would be resiliant toward *voter fraud*, *social manipulation through advertisement* and protect against *voter corruption*, *coercion*, *vote buying* and *vote selling*.

2.3 A Pinch of Cryptography

Cryptography uses the science of designing sound and complete procedures over machines—procedures called *algorithms*. *Algorithm* is a broad term but we will use it here as set of instructions that can be run on a tape. The steps must be precise, easy to follow, and must reach a termination state.

Complexity theory is the science of comparing algorithms and how *efficient* they are. Through the history of the complexity theory, in the 19th century, the notation of *Big-O*, $O()$ has been proposed by Bachmann [51] and today, after more than a century we are still using it in computer science as a measurement of the essential **time** or/and **space** requirements grow when the input size grows. We compute the time requirement based on behaviour of the algorithm and space based on the growth of the amount of needed memory (in terms of bits, bytes or kilobytes). *Efficient* algorithm is an algorithm that solves the problem in probabilistic polynomial time (PPT-bounded) .

2.3.1 Intractable Problems

Intractable problems or computationally hard problems are the problems that there is no efficient algorithm to solve them. Intractable problems have a non-negligible role in cryptography such as designing one-way functions. One of the most famous Intractable problems, used as the mathematical basis of this thesis, is the discrete logarithm problem (DLP).

The problem is to find x given $y = g^{x}{}_{mod\, p}$. As it has been well-described by Boneh[8], Diffie-Hellman key exchange protocol was the beginning of a journey into morphing Discrete Logarithm problem into various cryptographic applications. Let $g \in G_q$ be a generator and $a, b, z \in_r Z_q$ be random exponents. The DL-Problem is to compute a given $< g, g^a >$. The DDH-Problem is to distinguish $< g, g^a, g^b, g^{ab} >$ from $< g, g^a, g^b, g^z >$.

$$< g, g^a, g^b, g^{ab} > \tag{1}$$

We assume the computational intractability of the DDH-Problem within G_q , which implies the intractability of the DL-Problem. It is a fascinating fact that if you start counting from zero up to infinity, some numbers would show properties out of thin *succession.* For instance, given a positive integer n and an integer a co-prime to a, the multiplicative order of a modulo n is the smallest positive integer k such that $a^k \equiv 1 mod n$ and this is the basis of the Elgamal crypto-system.

2.3.2 ElGamal Cryptoscheme

In 1985, Elgamal [26] has presented a crypto-system based on the Diffie-Helmann key-exchange [25] based on the intractibility of DDH-problem. The protocol has been shown below.

ElGamal Crypto-scheme on DL problem

Initial state

- *Alice* and *Bob* both know and agreed upon a set of parameters $\{p, q, g\}$, where p is a large prime and $p = \alpha \cdot q + 1$ and $g \in G_q$ generates *a finite cyclic group* of prime order q.
- *Bob* has chosen a random secret x and computes h_b where $h_b = g^x \bmod p$. *Bob* has also shared h_b with *Alice* previously using a *integrity-preserving* channel.
- *Alice* intends to send an encrypted message m to *Bob* where m is in the form of $\{0.1\}^t$ bit string of length t.
- There is a channel to share information with satisfactory *availability* properties. The bandwidth and memory are sufficient to transfer $\{0.1\}^{2t}$ bit string of length $2t$.

Protocol steps

1. *Alice* chooses a random value $r \in G_q$ and computes a pair of *ciphertexts* (c_1, c_2), where $c_1 = g^r$ and $c_2 = m \cdot {h_b}^r$. *Alice* sends (c_1, c_2) to *Bob*.
2. *Bob* decrypts the message m using $m = c_1^{q-x} \cdot c_2 \bmod p$

Exponential ElGamal

In the Exponential version, the message m would be blinded using the predefined generator g. So that, the encryption function $\mathsf{Enc}()$, would be described as

$$\mathsf{Enc}(m, r) \equiv \mathsf{Enc}(m) = \langle \mathsf{C}_1,\ \mathsf{C}_2 \rangle = \langle g^r, g^m \cdot h^r \rangle \tag{2}$$

and decryption function $\mathsf{Dec}()$ would be in the form of

$$\mathsf{Dec}(\mathsf{C}_1, \mathsf{C}_2) = (\mathsf{C}_1{}^{\mathsf{sk}})^{-1} \cdot \mathsf{C}_2 = g^m \tag{3}$$

where $h = g^{\mathsf{sk}}$. The Exponential Elgamal is additively homomorphic, meaning that

$$\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 + m_2) \tag{4}$$

We would use that property to perform calculations under encryption. From now on, all protocols would be based on the same initial state on Exponential ElGamal Crypto-Scheme. We will not re-iterate basic assumptions of *common knowledge* and *satisfactory availability* between the involved parties.

2.3.3 Secure Multi-Party Protocols

In [34], Hazay and Lindell explained secure multi-party computation techniques and constructions. Multi-party protocols is a set of protocols that has have been designed to perform a secure computation among mutually untrusted parties. In many applications, we do not want to trust on a central authority, such as electronic auctions, lotteries and foremost the elections. In these cases, there could be a *corrupted* authority that would try to damage the system's fundamental security requirements. Therefore, We split the power among the parties with the assumption that there is no established trust among them. A secure multi-party protocols would be applied to ensure [1]

1. *Privacy*; no party should access any information more than its own generated output.
2. *Correctness*; each party would receive a correct output of other parties.
3. *Independence of Inputs*; every corrupted party must choose their inputs independently from honest parties' inputs.
4. *Guaranteed Output Delivery*; corrupted parties should not be able to interrupt the honest parties from generating and delivering their outputs. In broader security domain, this property is commonly known as *Denial-of-Service* resistance property.
5. *Fairness*; in multi-party computation protocols, there should be a fairness defined on the distribution of received outputs.

2.3.4 PET, Plain Equality Test

Juels et al. in [39] described the PET protocol as a multi-party protocol that enables the parties compare two different encryptions to determine whether they contain the same plaintext or not. Given two encryption, $\mathsf{Enc}(m_1)$ and $\mathsf{Enc}(m_2)$, n trustees, T_1 to T_n would follow the below protocol:

1. All the trustee calculates $\mathsf{Enc}(m_1 - m_2)$ by

$$\mathsf{Enc}(m_1 - m_2) = \mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2)^{-1} \tag{5}$$

[1]There have been several lists of properties for secure multi-party computation. In this article, the Hazay and Lindell list have been used.

Note that if m_1 and m_2 are equal, then $\mathsf{Enc}(m_1 - m_2)$ would be equal to $\mathsf{Enc}(0)$.

2. Each trustee T_i chooses a random value r_i and calculates

$$(\mathsf{Enc}(m_1 - m_2))^{r_i} = \prod_{j=1}^{r_i} \mathsf{Enc}(m_1 - m_2) \tag{6}$$

   Note that $\mathsf{Enc}(0)$ does not change by homomorphic addition to itself. So by doing this, if m_1 equals to m_2, the result would stay as $\mathsf{Enc}(0)$. T_i sends $(\mathsf{Enc}(m_1 - m_2))^{r_i}$ to the Election Authority.

3. The Election Authority multiplies the received values and then decrypt it. More formally,

$$g^m = \mathsf{Dec}(\prod_{i=1}^{n} (\mathsf{Enc}(m_1 - m_2))^{r_i}) \tag{7}$$

   where $i \in \{1, n\}$ and n is the total number of the trustees. If m equals to 0, it means that m_1 and m_2 are equal. Otherwise, a random number would be spitted out and does not reveal any information about m_1 or m_2.

2.3.5 Knowledge Proofs

Knowledge

We say a machine *knows* a witness ω if it can compute ω in polynomial time.

Generally speaking, there are two basic proofs around the concept of knowing. *proof-of-knowledge* protocols are interactive cryptographic protocols that through it, a party can prove knowing a piece of information. On the other hand, *Zero-knowledge* protocols are cryptographic primitives that are there to prove a statement without revealing any secret information. To dive deeper, first we have to formally define a new concept *e.g., Knowledge* and *Relation*.

Relation

Let $\mathcal{R}$ be a binary relation in the form $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$, where the only restriction is that if $(x, \omega) \in \mathcal{R}$, then the length of ω is at most $p(|x|)$, for some polynomial $p()$. For some $(x, \omega) \in \mathcal{R}$, we may think of x as an instance of some computational problem, and ω as the solution to that instance. We call ω a *witness* for x.

2.3.6 Sigma Protocols

Sigma protocols are interactive zero-knowledge proof protocol. Before the Sigma protocols, life was so hard for cryptographer to propose a zero-knowledge protocol and prove it formally. Σ-protocols made it really simple to design complicated functions.

Sigma protocols (Σ-protocols) are interactive games that prover $\mathcal{P}$ will win if she can prove the statement $\mathcal{S}$ to an honest verifier $\mathcal{V}$ without revealing any information the statement [2]. We will talk about the mathematical proof of *Zero-knowledge* and *Proof-of-knowledge* in Chapter. Here, we briefly introduce some basic Σ-Protocols.

Σ-Protocol

A protocol is a $\Sigma-$*protocol* for relation $\mathcal{R}$ if it is a *three-round public-coin* protocol and the following requirements hold:

1. Completeness: If $\mathcal{P}$ and $\mathcal{V}$ follow the protocol on input x and private input ω to $\mathcal{P}$ where $(x, \omega) \in \mathcal{R}$, then $\mathcal{V}$ always accepts.
2. Special soundness: There exists a polynomial-time algorithm A that given any x and any pair of accepting transcripts $(a, e, z), (a, e', z')$ for x where $e \neq e'$, outputs ω such that $(x, \omega) \in \mathcal{R}$.

[2]except the validity of the statement.

3. Special honest Verifier zero knowledge: There exists a probabilistic polynomial-time simulator $\mathcal{M}$, which on input x and e outputs a transcript of the form (a, e, z) with the same probability distribution as transcripts between the honest $\mathcal{P}$ and $\mathcal{V}$ on common input x. Formally, for every x and ω such that $(x, \omega) \in \mathcal{R}$ and every $e \in \{0,1\}^t$ it holds that

$$\{\mathcal{M}(x, e)\} \equiv \{\langle \mathcal{P}(x, \omega), \mathcal{V}(x, e)\rangle\}$$

where $\mathcal{M}(x, \omega)$ denotes the output of simulator $\mathcal{M}$ upon input x and e, and $P(x, w), V(x, e)$ denotes the output transcript of an execution between $\mathcal{P}$ and $\mathcal{V}$, where $\mathcal{P}$ has input (x, ω), $\mathcal{V}$ has input x, and $\mathcal{V}$'s random tape (determining its query) equals e.

Schnorr Proof-of-Knowledge

Schnorr protocol[71] is one of the efficient ways to prove knowing a secret without revealing it.

Non-interactive Schnorr Protocol on DL problem

Initial state

- $\mathcal{P}$ and $\mathcal{V}$ both know and agreed upon a set of parameters $\{p, q, g\}$, where p is a large prime and $p = \alpha q + 1$ and $g \in G_q$ generates *a finite cyclic group* of prime order q. Both parties agreed upon the construction of a statement $\mathcal{S}$ where $\mathcal{S} = \{g^{\omega}, \{p, q, g\}\}$ for witness ω.

- $\mathcal{P}$ intends to prove knowledge of $\omega \in G_q$ to $\mathcal{V}$ without revealing its value. She constructs a statement $\mathcal{S}$ in the above mentioned structure.

- There is a channel to share information with satisfactory *availability* properties. The bandwidth and memory are sufficient to transfer $\{0.1\}^t$ bit string of length t.

Protocol steps

1. $\mathcal{P}$ chooses are random value $r \in G_q$ and calculates b where $b = g^r \bmod p$. She also calculates a strong Fiat-Shamir heuristics enabled challenge $c = H(\mathcal{S}\|b) \bmod p$ where H is a secure *one-way function* with no known *efficient* backdoor with t-bit length of output. Then, She calculates $d = (r + \omega c) \bmod q$ and sends $\{b, c, d\}$ to $\mathcal{V}$.

2. $\mathcal{V}$ checks the validity of these two statements:

   2.1. $c == H(\mathcal{S}\|b)$

   2.2. $g^d == b * (g^\omega)^c \bmod p$

Chaum-Pedersen Proof-of-Membership

Chaum-Pedersen commitment[60][3] is a type of commitment that can fit in additively homomorphic encryption scheme.

Non-interactive Chaum-Pedersen Protocol on DL problem

Initial state

- $\mathcal{P}$ and $\mathcal{V}$ both know and agreed upon a set of parameters $\{p, q, g, h\}$ describing an *Exponential ElGamal* Crypto-Scheme. Both parties agreed upon the construction of a statement $\mathcal{S}$ where $\mathcal{S} = \{g^\omega, \{p, q, g\}\}$ for witness ω.
- $\mathcal{P}$ intends to prove knowledge of $\omega \in G_q$ to $\mathcal{V}$ without revealing its value. She constructs a statement $\mathcal{S}$ in the above mentioned structure.
- There is a channel to share information with satisfactory *availability* properties. The bandwidth and memory are sufficient to transfer $\{0.1\}^t$ bit string of length t.

[3]In the literature it has been often call Pedersen commitment, however it is worth mentioning that David Chaum had a non-negligible role in the construction of the idea.

Protocol steps

1. $\mathcal{P}$ chooses are random value $r \in G_q$ and calculates b where $b = g^r \bmod p$. She also calculates a strong Fiat-Shamir heuristics enabled challenge $c = H(\mathcal{S} \| b) \bmod p$ where H is a secure *one-way function* with no known *efficient* backdoor with t-bit length of output. Then, She calculates $d = (r + \omega c) \bmod q$ and sends $\{b, c, d\}$ to $\mathcal{V}$.
2. $\mathcal{V}$ checks the validity of these two statements:
   2.1. $c == H(\mathcal{S} \| b)$
   2.2. $g^d == b * (g^\omega)^c \bmod p$

Fiat-Shamir Heuristics

Fiat-Shamir Heuristics is a trick to convert an interactive protocol to an non-interactive protocol that has been proposed by Amos Fiat and Adi Shamir in 1987 [29]. In the case of Σ-protocols, $\mathcal{P}$ would be able to prove $\mathcal{S}$ to $\mathcal{V}$ in only one single interaction.

Strong Fiat-Shamir heuristics is the case that the $\mathcal{P}$ commits to the whole parameters that involved in constructing the statement.

Conjunction and Disjunction

based on the Cramer et al.'s studies [23], more complicated Σ-protocols can be constructed by performing logical *AND* and *OR* over smaller Σ-protocols such as *Schnorr* and *Chaum-Pedersen*.

2.4 Threats to Online Voting Systems

Although some countries have already forged ahead despite security and privacy vulnerabilities, some leading computer scientists such as MIT's Ronald Rivest admonished [4] that, given current limitations of technology, Internet voting is akin to "drunk driving."

Widely used mail-in ballots are not susceptible to malware attacks at the voter's end, but they too are vulnerable to undue influence. For instance, a dishonest or coerced voter might take a video of them filling out, sealing in an envelope, and mailing the ballot. Similarly, a voter could also covertly take a video of them voting in a precinct—curiously, in some precincts such videos are even permitted. It should not be possible for a voter to prove how they voted.

Nonetheless, there are still some extra challenges the make the design of a secure internet voting system difficult:

2.4.1 Untrusted Platforms

Malware on the voter's device (e.g., phone) might undetectably modify votes and spy on voters. Internet voting relies on each voter having an uncompromised device to vote from. A general framework for this literature is to assume voters have at least two independent communication channels with the election authority and at most one is compromised. For example, the task of ballot casting and verification might be split between the paper-based postal system and a computer [15, 36, 40, 69, 78], between a computer and a mobile device [35], or assume a single computer cannot read what is communicated (e.g, via CAPTCHAs [58, 64]).

[4]At a special workshop on Uniformed and Overseas Citizens Absentee Voting Act (UOCAVA), EVT/WOTE, 2010.

2.4.2 Online Denial-of-Service Attacks

A third issue with online voting is ensuring the vote submission system remains online and responsive, thwarting both a natural increase in traffic and deliberate subterfuge through denial of service attacks. Determined adversaries might try to launch an online attack, including causing outages. For instanse It is also important to mitigate *Flooding attacks* on the bulletin board.

2.4.3 Unauthorized Intervention

The lack of a secure physical voting precinct facilitates *Coercion*, *Undue Influence*, including *Vote Selling* and *Vote Buying.* There are some occasions that the user may be under duress or pressure to interact with the system based on the coercer's intent and benefit.

Of these challenges, the most elusive has been mitigating *Unauthorized Intervention.* Coercion resistance is an extremely important and strong property that is difficult to achieve, especially for remote voting, where there is no physically-secure voting place and the machines used to interact with the voting system are untrusted. In the next section, we will

Coercion Resistance

Coercion Resistance property is to ensure that the user of the system interacts freely and express her own intent. Informally, a voting system is *coercion resistant* if and only if no voter can prove to any coercer that the voter cast a counted ballot according to the coercer's instructions. For example, voter Alice cannot prove to a coercer that Alice cast a ballot that was counted for Trump.

Coercion resistance is at least as strong as *receipt freeness.* Receipt freeness [5, 57] usually means that the voter cannot prove how she voted, simply by following the voting protocol. For example, an adversary might contact a voter only after the voter finishes voting. Coercion resistance [42, 50, 48] is stronger in that the coercer may in advance instruct the voter to carry out (or not carry out) certain additional actions that are outside of the voting protocol.

Haines and Smyth [32] surveyed four definitions of coercion resistance and found that "coercion resistance has not been adequately formalized." Three of the definitions are too weak, and the general definition by Kösters [48] is complex and too strong. To date, researchers have attempted to address vote selling and coercion using three main approaches.

1. **Re-voting**[5] schemes. VoteAgain [49, 77] makes Revoting schemes more efficient. This approach is simply letting voters to vote again, nullifying any previously cast ballots [77, 73, 49]. This assumes the adversary does not coerce the voter at the end of the election period, or cannot otherwise lock the voter out from re-voting (*e.g.,* retain their ID smartcard or change their voting credential).

2. **Panic Passwords** also has been call *Fake Credential* by [42] strategies are focusing on preparing a voting interface for the voter to have a secret way to signal EA about being under duress. This approach allows voters to create fake credentials they can use when voting under coercion [42, 21, 20, 4, 3, 28]. Fake ballots are verifiably removed from the tally without revealing which ballots were fake, and much academic effort has been devoted to linearizing the computational cost of this filtering process.

3. **Decoy Ballots** Random Sample Voting mitigates coercion in part by keeping secret who are the voters. [16].

[5]In some studies of the literature, it has been dubbed *Multiple Voting* schemes.

Previous approaches to coercion depend on very strong assumptions. For instance, some systems assume that the voter cannot be coerced during registration [43]. Others allow the voter to vote multiple times but assume that the voter can vote freely the last time. Our nullification approach does not have these limitations. Some systems attempt to achieve coercion resistance by allowing the voter to modify her vote up to the end of the voting interval, with a rule that only the last vote counts (e.g., see [77]). Others assume the presence of an *untappable channel* [5, 70]. Depending on the rule that only the last vote counts is problematic because the coercer could physically interfere during the final moments of the voting interval. In contrary, in our system, we assume the voter has atleast a single opportunity at an arbitrary time during the election period, before or after coercion, to alert a nullifier of their intention and authorize them to void their ballot.

An important aspect of strong incoercibility is the ability for a voter to use an untappable channel. It is generally believed that without an untappable channel, the coercer and voter are indistinguishable and therefore incoercibility is impossible to achieve [38]. Re-voting assumes the channel is present after a coercive action, while the use of fake credentials assumes registration is conducted over an untappable channel.

To prove coercion resistance, it is helpful to consider the strongest possible attack, which involves the voter providing to the coercer all of the voter's information, including cryptographic keys. Coercion resistance is challenging because the voter must be convinced that her vote was correctly cast, collected, and counted, yet she must be unable to transfer this proof to the coercer.

The system cannot distinguish between different entities that know the voter's secrets. Therefore, once the coercer learns the voter's secrets, the only mitigation remaining is for the voter to *Nullify* their vote.

Our strategy for resisting coercion involves three ideas.

- First, we introduce the concept of a *Nullifier*, a covert entity trusted by the voter to act on the voter's behalf, and to whom the voter provides certain secrets.
- Second, since we assume the coercer has all of the voter's secrets (except for the complete list of the voter's *Nullifiers*), we consider the post-casting actions of the voter, Nullifier, and coercer indistinguishable.
- Third, after casting her ballot, we permit the voter to modify or nullify her vote, up to the end of the voting interval.

Chapter 3

VoteXX

"*One of the penalties for refusing to participate in politics is that you end up being governed by your inferiors.*" - Plato

As we have seen in the previous section, one of the most suppressed voting properties is coercion-resistant. To tackle that issue, we introduce a new internet voting system, designed by a team of researchers including myself [17].

3.1 Definitions

Since no protocol can be satisfactory to hold in any and every election, we focused on the elections with the following properties.

1. For simplicity and without losing generality, we assume that it is a binary election, where there are only two valid choices: "YES" and "NO". Through the paper, We have used Y and N subscripts to denote variables and calculations that are related to "YES" vote and "NO" respectively.
2. The number of voters are less than 10,000. This again does not affect the generality, as in real world, elections would be broken into geological sub-elections.

3. The election consists of five main phases [Setup, Registration, Voting, Nullification, Tally] and that authorities have time to sign the previous phase to ensure the correctness of the protocol.

4. There are two available channels. One is through CMix network with strong privacy preserving mechanisms that provide network-level anonymity. We refer to this as the anonymous communication channel (ACS). The other one is secure connection over TLS. Depending on the entity and the level of privacy if the action, one of these two would be chosen.

5. There is a secure public key encryption scheme available as a cryptographic primitive to preserve confidentiality. We used three different symbols to denote Encryption, $[\![x]\!]$, $\mathsf{Enc}(x)$ and $\mathsf{Enc}(x, \alpha)$, where x is the plaintext and α is the random value using for encryption.

The signed version of election definition[1], SignedElectionConfiguration would be posted on a public website by the election authority. The election definition should contain

1. ProtocolVersion. First the version of the protocol would be defined in the configuration file

2. ElectionIdentity. The election process starts by broadcasting ElectionIdentity, a relatively short (20 characters) human readable string, only including English capital letters and numbers.

3. ElectionDescription. The Election Description must include the election question and the possible vote choices. In the case of this study a binary voting system has been assumed, the vote choice or VoteCode would be in the same format of the Election identity.

[1]signed by the Election Authorities

4. ElectionTimetable. The UNIX timestamp of start and end of every phase.

5. Cryptoscheme definition. The definition of the applied cryptoscheme would be mentioned here. In *Exponential Elgamal* case, the values p, q, g, h must be included, where p, q, g defines an Elgamal encryption scheme and h is the EA's public key.

6. Registrars' public keys.

7. Addresses. Any address that must be used during the election would be mentioned here. Also Election authorities can publish new addresses in case of Denial of Service attacks. Addresses includes, boot-strapping ledger and bulletin board addresses, also the Mixnet endpoint for the Election authorities.

To start generating our own set of election entities based on our requirements, we have to have a baseline glossary of what is going on in a voting system. For that purpose, we double-checked the list with NIST E-voting glossary [54] to ensure that no entity has been missed and no known problem's sub-space is untouched.

3.2 Entities

There are different roles and entities in VoteXX protocol to let the game to go on. Here, we enumerate them with a short description.

- **Bootstrapping public ledger**.would be used for initiating the election. Since it would be crucial for starting the election, availability of this step is more important than the privacy. So, the design choice is to use a public memory to broadcast the initial configuration of the election. Also *non-repudiation* is a feature that we cover. So append-only ledger will public read access must be used. Fortunately, we are living in an era that blockchains have been invented.

- **Public Bulletin board**. Denoted as BB, *Bulletin Board* would be used to publish all public information and all the information that is needed for the further auditing. It will be using strong integrity-preserving mechanisms.

- **Anonymous Communication Channel (ACS)**. We would be using Chaum's CMix [18] as a private channel. Mixnets make the communications almost untraceable. The CMix version is one of the fastest implementation due to heavy precomputation.

- **Election Authorities**. Denoted by $\mathsf{EA} = \{\mathsf{EA}_1, \mathsf{EA}_2, ..., \mathsf{EA}_{n_{\mathsf{EA}}}\}$, authorities are a set of authenticated mutually untrusted key-holders that are collectively responsible for

  - Initiating the election.
  - holding the election and ensuring the correctness of execution of the written protocol by properly signing players' packets.
  - Performing Threshold Elgamal key-exchange protocol.
  - Performing multi-party Nullification protocol
  - Sign the result of the election and approve the process to accumulate the public trust.

- **Voters** Denoted by $\mathcal{V} = \{V_1, V_2, ..., V_{n_V}\}$ is the set of eligible The person that is freely choosing to vote and what to vote.

- **Nullifiers** Denoted by $\mathcal{N} = \{N_1, N_2, ..., N_{n_N}\}$, *Nullifiers* are responsible to flip votes to mitigate voting under duress situations. The full functionality would be explained in Chapter 4.

- **Observers** [2] Denoted by $\mathcal{O} = \{O_1, O_2, ..., O_{n_O}\}$, auditors are responsible to

[2]The term is coming from European Voting systems such as Estonian i-vote

assure that authorities do not collude together and break the *Integrity* or *correctness* of the system.

3.3 Assumptions and Thread Model

We classify the assumptions into three categories: assumptions of the underlying cryptographic building blocks; trust assumptions and assumptions of the coercer's ability.

For the first one, we assume that the Zero-knowledge proofs are complete, sound and zero-knowledge; the ElGamal encryption scheme is IND-CPA secure; the signature scheme is secure against existential forgery; and during the Registration phase, we assume that Discrete Logarithm problem is hard.

For the second one, we assume that at least one hedgehog is honest; at least one party of the mixnet is honest (Anonymous Communication Channel); and no more than $t-1$ parties of the EA are corrupted.

For the third one, we assume that a voter can establish an undetected channel with his Nullifiers that will be explained in Section 4. This is realistic because we consider it impossible to coerce a party all the time during his lifetime. In other words, a voter can generate his public-private key pair and share it with his Nullifiers even before the election. When a voter is coerced during the voting phase, he will use the undetected channel to let his Nullifier to nullify the vote, where the undetected channel can be failing to contact the Nullifier until some deadline. This is a realistic assumption and captures the real world scenario.

3.4 Phases

VoteXX election phases have been described as followed.

- **Setup** phase will ensure that it is impossible to stop the election from initiating. All that is needed for a correct election is prepared. Otherwise, it will generate a report containing the warnings about future complications.
- **Registration**. During registration, the voter presents the public key but never reveals the private key to the system. To flip a vote, the voter engages in a zero-knowledge proof with the system to prove that they know the private key, without revealing the private key.
- **Voting**. [A moment of silence, showing respect to whoever sacrificed their time and health to make the voting systems running safe and sound.]
- **Nullification** would be used to publish all public information and all the information that is needed for the further verification.
- **Tally** would be counting the total votes and nullifications that had been splitted into two phases: *Provisional Tally* and *Final Tally* to avoid Running Tally problem.

3.5 Protocol

Protocol Boxes 1–3 explain the main phases of the VoteXX protocols: setup and registration (presented together), voting, and nullification and tallying (presented together). In Figure 1, a summarized version of the whole protocol has been illustrated.

There have been previously proposed Coercion Resistant Registration mechanisms in the E-voting literature. For instance, [43] uses randomly generated passphrase splits (by a set of trustees) to create the first voter identification key. Another approach by Clark and Hengartner [20] is to set a panic password. Here, we are proposing a new technique that focuses on multiple identity verification without revealing

Registration Set-up. Registration uses a trapdoor commitment scheme. The commitment aspect allows the voter to present her passphrase in a hidden form to the EA and answer queries about specific characters within it. The trapdoor is revealed after registration closes and allows each voter to convert the format of their commitments into the format of a public key.

1. The generator g_0 is a parameter of the election.
2. The EA computes a generator g_1 as follows: each EA trustee $\mathsf{EA}_1, \mathsf{EA}_2, \dots$ privately chooses one random value a_1, reveals $g_0^{a_1}$, and proves knowledge of a_1 with a Schnorr $\Sigma-\mathsf{Protocol}$. Then $g_1 = g_0^{(a_1+a'_2+a''_3+\dots)}$.
3. This process is repeated, with new random a_i values, to complete a set of N generators: $\mathsf{base} \leftarrow \langle g_0, g_1, g_2, \dots, g_{N-1}\rangle$. The same base is used for all voters in a registration period.
4. Call the set of all a values (split across the trustees): $\mathsf{trapdoor}$.

Registration.

1. Each voter generates two N-character passphrases (for YES and NO). Steps 2-4 describe the process for the first passphrase and are repeated for the second.
2. The voting client parses the passphrase as a sequence of Base64 characters $\langle c_0, c_1, c_2, \dots, c_N\rangle$ and computes its deterministic commitment using base: $\mathsf{passCommit} \leftarrow \langle g_0^{c_0} \cdot g_1^{c_1} \cdot g_2^{c_2} \cdot \dots \cdot g_{N-1}^{c_{N-1}}\rangle$.
3. The voting client sends $\mathsf{passCommit}$ to the EA.
4. The EA officer issues a challenge like: "Reveal Character 4." The voter responds "F." The EA client computes $\mathsf{disclosedChar} \leftarrow (\mathsf{passCommit}/g_4^{\mathtt{F}})$. The voting client proves knowledge of a representation of $\mathsf{disclosedChar}$ using a $\Sigma-\mathsf{Protocol}$. This step is repeated to build confidence that the voter knows the passphrase, but bounded in repetitions to protect the passphrase.
5. The EA client posts $\langle \mathsf{VoterID}, [\![\mathsf{passCommit_Y}]\!], [\![\mathsf{passCommit_N}]\!]\rangle$ to the BB, where $[\![\mathsf{passCommit}]\!]$ is an encryption of $\mathsf{passCommit}$ under the EA's threshold encryption scheme. Finally the EA client proves to the voter client the correctness of the encryptions using the Chaum-Pedersen $\Sigma-\mathsf{Protocol}$.

Registration Finalization.

1. After the registration period, the EA takes the list of $\langle \mathsf{VoterID}, [\![\mathsf{passCommit_Y}]\!], [\![\mathsf{passCommit_N}]\!]\rangle$ entries, removes the $\mathsf{VoterID}$ component, and verifiably shuffles, threshold-decrypts, and posts $\langle \mathsf{passCommit_Y}, \mathsf{passCommit_N}\rangle$ for each (now anonymous) voter.
2. Each trustee $T, T', T'', \dots$ reveals their values producing $\mathsf{trapdoor}$.
3. Each voter uses $\mathsf{trapdoor}$ to reformat their two $\mathsf{passCommit}$ values into key pairs $\langle \mathsf{sk}, \mathsf{pk}\rangle$ such that $\mathsf{pk} = \mathsf{passCommit} = g_0^{\mathsf{sk}}$ as follows. Consider generator g_i and let $\alpha_i = a_i + a'_i + \dots$. With this notation, $\mathsf{sk} = c_0 + \alpha_1 \cdot c_1 + \alpha_2 \cdot c_2 \dots$.
4. Given that $\langle \mathsf{passCommit_Y}, \mathsf{passCommit_N}\rangle = \langle \mathsf{pk_Y}, \mathsf{pk_N}\rangle$, the EA holds an anonymized list (Roster) of $\langle \mathsf{pk_Y}, \mathsf{pk_N}\rangle$ keys for each registered voter.

Protocol 1: Registration Protocol.

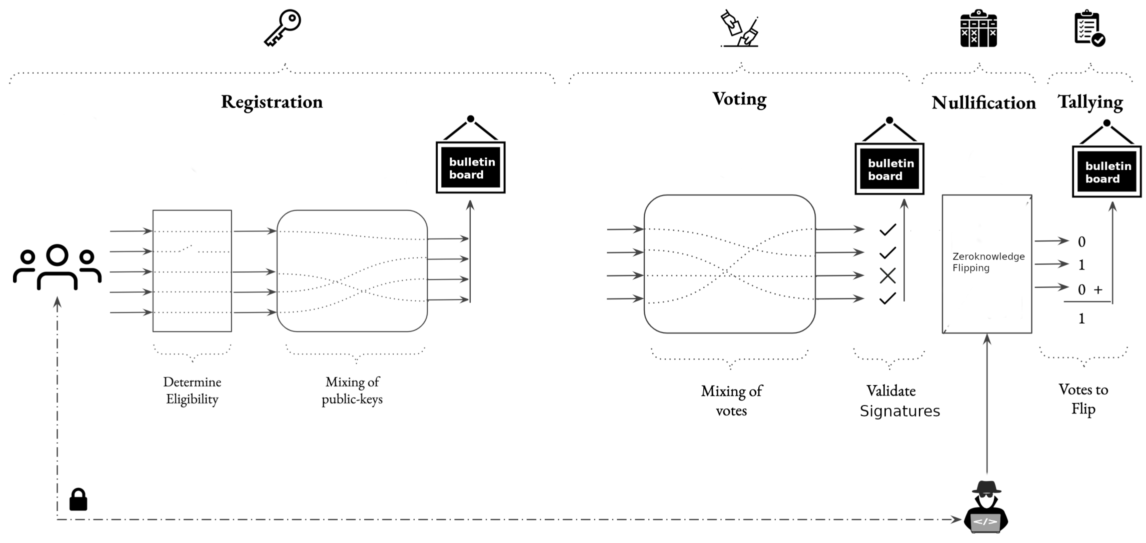

Figure 1: Phases of VoteXX e-voting Protocol

Voting. Each voter completes voting online. At completion, each voter will have submitted their ballot using a passphrase from registration.

1. The nonce nonce is a parameter of the election.
2. To mark a ballot for YES, the voter uses their YES passphrase to generate $\mathsf{sk_Y}$ and uses this key to sign $\mathsf{n_0}$: $\sigma_\mathsf{Y} \leftarrow \mathsf{Sig}(\mathsf{nonce})$. Corresponding values are used to vote NO.
3. The voter uses the EA's threshold encryption scheme to compute $\mathsf{ballot} \leftarrow \langle [\![pk_\mathsf{Y}]\!], [\![\sigma_\mathsf{Y}]\!], \pi_\mathsf{ppk}\rangle$, where each group element of σ is individually encrypted and π_ppk is a proof of plaintext knowledge using the Chaum-Pedersen $\Sigma-\mathsf{Protocol}$.
4. The voter submits ballot over the ACS to the BB. The EA marks it as invalid if it is an exact duplicate or if the proofs are invalid.

Protocol 2: Voting Protocol.

the password, Registration is an in-person ceremony between the voter, using a *voting client* device, and an officer for the EA. At completion, the voter registers two public keys $\langle \mathsf{pk_Y}, \mathsf{pk_N}\rangle$, which are not learned by the EA officer and will be used to vote YES and NO, respectively. The keys are for a digital signature. They are based on a passphrase that can be regenerated from any voting client. The EA additionally does not learn the passphrase but has high assurance through the protocol the human voter knows the passphrase.

Provisional Tally. After the voting period is over, the EA produces a verifiable provisional tally.

1. The EA takes the list of $\langle [\![\mathsf{pk}]\!], [\![\sigma]\!] \rangle$, then verifiably shuffles and threshold-decrypts them: $\langle \mathsf{pk}, \sigma \rangle$.
2. For each ballot, the ballot is marked invalid if σ does not verify under its corresponding pk.
3. For each valid signature, pk is matched to its entry on the Roster. The EA determines if it is a YES or NO key, and counts the vote only if it is the only ballot cast that corresponds to that roster entry. (Since ballots are not shuffled, other policies are feasible such as counting the most recent vote.)

Nullification. The goal of nullification is to allow voters to modify their cast ballots, particularly in the case of coercion. Unlike other protocols, voters can enlist the help of others parties, called hedgehogs. The nullification period runs after the provisional tallying. If the provisional tally contains $\mathsf{pk_N}$, it can be nullified using $\mathsf{sk_Y}$ (the "opposite" key). In other words, casting a YES and nullifying a NO vote use the *same* key, as these two actions are aligned in their intention.

1. At any convenient time, before or after voting, the voter covertly communicates with a hedgehog to develop a coercion-resistant strategy. Assume the following strategy: the voter wants to vote YES and reveals $\mathsf{sk_Y}$ to the hedgehog, along with $\langle \mathsf{pk_N}, \mathsf{pk_N} \rangle$. They request the hedgehog engage in nullification if $\mathsf{pk_N}$ is in the provisional tally.
2. Using the Roster and set of valid signatures from the provisional tally, the EA reformats the election data into two lists. The first list establishes, in arbitrary order, the set of $\mathsf{pk_N}$ keys from voters who cast valid votes for YES (call it $\mathsf{yesVotes}$). The second list contains $\mathsf{pk_Y}$ from voters who voted NO.
3. For example, assume YES received six votes in the provisional tally. $\mathsf{yesVotes}$ consists of six $\mathsf{pk_N}$ keys. If the hedgehog wants to nullify the fourth key, it prepares a list of encrypted "requests" marking the ballot it wants to nullify: $\langle [\![0]\!], [\![0]\!], [\![0]\!], [\![1]\!], [\![0]\!], [\![0]\!] \rangle$. See section 4.1.5.
4. The first encrypted flag corresponds to the first $\mathsf{pk_N}$ in $\mathsf{yesVotes}$. The hedgehog adds a proof to this list using the nullification $\Sigma-\mathsf{Protocol}$. Concisely, the proof statement is: [(this flag is an encryption of 0) or (this flag is an encryption of 1 and I know $\mathsf{sk_N}$ corresponding to this $\mathsf{pk_N}$)]. This is detailed in section 4.1.2.

Final Tally. After the nullification period is over, the EA produces a verifiable final tally.

1. The EA takes all the encrypted flags for the first $\mathsf{pk_N}$ key in $\mathsf{yesVotes}$ and computes its xor under encryption (described in section 4.1.5). It repeats this process for the remaining $\mathsf{pk_N}$ keys.
2. The EA takes the list of encrypted xored flags, sums them under encryption, and verifiabily threshold-decrypts the result. The EA subtracts this value from the number of YES votes in the provisional tally to produce the final tally for YES votes.
3. The EA repeats Steps 1-2 for each $\mathsf{pk_Y}$ key in $\mathsf{noVotes}$.

Protocol 3: Tallying Protocol (including nullification).

The VoteXX protocol assumes a number of cryptographic primitives that are common in the voting literature. All operations are performed in the same elliptic curve group, where the decisional *Diffie-Hellman (DDH)* problem (and by extension, the discrete logarithm problem) is hard. Digital signatures are performed with the Schnorr signature scheme. Encryption is performed with ElGamal [26], which can be augmented with *distributed key generation (DKG)* and threshold decryption (for m out of n key holders [61]). We use standard $\Sigma-\mathsf{Protocols}$ to prove knowledge of discrete logarithms (Schnorr [71]), knowledge of Diffie-Hellman tuples (Chaum-Pedersen [19]), which also corresponds to ElGamal re-randomizations and decryptions, and knowledge of representations (Okamoto [56]). We also use techniques to allow the trustees to compute jointly and verifiably (*i.e.,* produce $\Sigma-\mathsf{Protocol}$ proofs), and to compute privately, on ElGamal ciphertexts the following: (i) a random shuffle of ciphertexts (Verificatum), and (ii) the evaluation of an *exclusive-or (xor)* operation based on its logic lookup table (mix and match [39]).

Protocol 1 describes registration. Registration can be re-opened by re-running setup. Once a voter key is registered, it can be used in later registration periods. Voting performs a straightforward signature, using a registered key (see Protocol 2). At the end of registration, voter keys are unlinked from their identity. Until the election closes, votes are encrypted to preserve the secrecy of the tally, and ballots are submitted through the *ACS* to unlink them from the voter network and communication metadata.

The tallying process (Protocol 3) includes our novel nullification technique. Consider a list of public keys that voted YES and assume the hedgehog wants to nullify one of them. It cannot point out which key it wants to nullify or the coercer will know the voter is working with (or is personally acting as) a hedgehog to intervene. So the hedgehog must hide its flag ($[\![1]\!]$) in a set of false flags ($[\![0]\!]$) for each YES key in the tally. We could allow the hedgehog to choose a fixed-sized subset of β keys at random to serve as an anonymity set, which improves performance but sacrifices full anonymity (*cf.* [20]).

As the nullification procedure is the main contribution of this thesis, we will describe how it works and why step-by-step in the next chapter. The core idea is the nullification $\Sigma-\mathsf{Protocol}$ that lets a hedgehog hide its intent (see Section 4.1.2) and the aggregation of nullification requests (see Section 4.1.5)

Once a set of flags (each real or false with its own proof Π) is computed and submitted by a hedgehog, Protocol 3 simplifies the description by having the EA wait to perform Steps 1–2 after the nullification period. In practice, it should not wait—it is quadratic work (number of hedgehogs times number of voters) and subject to "board flooding" attacks [45]. It must process the nullifications as they arrive ("concurrent authorization" [28]). Doing so is possible. When a new set of flags arrives, each proof is checked and the xor between the submitted flag and the accumulation of previous flags is computed (both are parallelizable for each flag). Thus, when nullification closes, the only remaining task is to threshold decrypt the accumulation of flags, which is linear in number of votes.

3.6 Security Analysis (Sketch)

In this section, we sketch a security analysis for the major security concerns of the system.

3.6.1 Registration

For the *Registration* phase:

- **Proof-of-knowledge of the passphrase.** The voter is challenged by the

EA to reveal partial information about her committed passphrase, $\mathsf{PassCommit}$, such as the character at a particular location. The EA's challenges are based on any information that can be disclosed with an efficient $\Sigma-\mathsf{Protocol}$ including character ranges, conjunctions/disjunctions of characters, *etc.* The challenges build confidence but reduce the entropy of the passphrase, so a balance must be struck. For example, say a voter is given a passphrase to register by a coercer but only told 10 of the 20 the characters in it, which the adversary believes is enough to respond to challenges but not enough to use it to vote. The EA asks for three characters at random. The voter will be caught 87.5%. The security of the passphrase is reduced to 17 characters.

Additionally, assuming one honest trustee, the relative discrete logarithms in the base of the commitment are not known, resulting in a commitment that is binding. Opening such a commitment (a Pedersen multi-commitment) to two different passphrases is exactly equivalent to breaking the discrete logarithm problem [61]. So, the voter cannot provide answers that correspond to a different passphrase than the one they know.

- **Passphrase confidentiality.** The commitment scheme is computationally hiding under the discrete logarithm assumption. However, as it is deterministic, passphrases can be broken through exhaustive search once they are decrypted and posted on the bulletin board. Human-chosen passwords are known to be weak [9, 76] and even if a small percentage (*e.g.,* 5%) of voters chose weak passwords, it could be well-within the margin of victory to exploit them. Adding randomness to the commitment improves its hiding property but the randomness would have to be remembered by the voter and it effectively reduces to the voter using a key instead of passphrase. Thus this is a theoretical limitation: the security of any passphrase-derived key is the unguessability of the passphrase.

We do not affirm the use of passphrases in every situation, but we believe our registration protocol is an interesting option to have documented in the literature. Any protocol that allows a public key to be stored on the bulletin board is sufficient for VoteXX (*e.g.,* voter generates one with their device, stores it, and uses it during voting).

- **Registration Disruption by the EA trusties.** A non-cooperative trustee can be eliminated from the registration set-up process however we rely on every trustee that participates in set-up to return and finish the registration finalization step, otherwise voters cannot convert their passphrase into a signing key pair. A single trustee can disrupt the registration process which is not ideal. Future work will explore enabling recovery of the values when only a smaller threshold of trustees participate. This seems possible to implement directly from a distributed key generation protocol [61]. All other aspects of the voting system require only a threshold of trustees to participate. So non-colluding trustees cannot disrupt the registration process.

- **Voters' anonymity.** Voter's registration public keys would be only decrypted after the mix. Assuming one honest trustee (mixing), it is computationally infeasible to determine which public key corresponds to which voter.

- **Ballot Stuffing attack.** We assume the EA only registers voters that are eligible to participate in the election, like most cryptographic voting systems. Ballot stuffing in our system could be conducted by signing a ballot with an unregistered public key. However any ballot stuffing attack would be detectable as all keys are made public, and the mixing of the keys proves that the outputs match the inputs (in both the number of them, as well as the plaintext contents under encryption).

3.6.2 Voting

For the *Voting* phase:

- **Impersonation.** The system avoids impersonation by using digital signatures.
- **Replay/modification Attack.** Two different levels of replay attacks can be considered. The votes can not be replayed for the next election, because each election has a unique nonce. However the replay attack inside an election can cause denial-of-service of the bulletin board. For mitigating this, the EA should only accept ballots that have a new (fresh) signature. Modification of a ballot would be infeasible due to the use of the digital signature.
- **Anonymity.** We assume that a coercer can know that a voter voted, however for other passive adversaries, the network-level anonymity of the voter is being preserved by the ACS channels. The unlinkability between a voter and their key is protected by the verifiable shuffle after registration.
- **Running tally.** Assuming at least one honest trustee, the ballots are only decrypted (by threshold decryption scheme) after the voting period has finished. So the tally will not be visible to anyone before the provisional tally phase.
- **Disruption.** We assume a widely-used ACS channel that is not possible to disrupt at scale for a long period of time.

3.6.3 Provisional Tally

For the *Provisional Tally* phase:

- **EA cannot perform timing attacks.** Due to mixing, decrypted ballots have no order, assuming at least one honest trustee.

- **E2E Verifiability.** Voters can check the bulletin board to see their ballot (encrypted) in included unmodified. Even if all trustees collude, the zero knowledge proof of a correct shuffle and decryption assure anyone of the result.

- **Provisional Tally Disruption.** Since we are using a threshold encryption scheme, EA members can not disrupt the calculation of tentative tally, unless they collude and compromise the threshold.

3.6.4 Nullification

For the *Nullification* phase:

- **Eligibility.** Nullifier can nullify if and only if she knows the key. For that purpose a $\Sigma-\mathsf{Protocol}$ has been designed and implemented that it would be described in the next chapter.

- **Replay attack resistance.** We are using non-interactive zero-knowledge proofs that are not divertable. The freshness of the random value that has been used to generate nullification proofs would be checked by the decentralized auditable bulletin board.

- **Disruption resistance.** The decryption of the encrypted signal can not be disrupted as long as a *threshold* number of the trusties are honest.

- **Coercion resistance.** The coercer can act as a nullifier but does not have a dominant strategy—how many times should it nullify? No, number is guaranteed to yield the coercer's preference because it does not know if the voter has one or more other nullifier simultaneously nullfiying the ballot.

3.6.5 Final Tally

For the *Final Tally* phase:

- **E2E integrity.** Hedgehogs can check the bulletin board to see their requests (encrypted) are included unmodified. Even if all trustees collude, the zero knowledge computation is publicly verifiable by any observer. The next chapter will prove the nullification proof is sound.
- **Nulification secrecy.** Due to the final mixing and non-interactive zero-knowledge proof, which ballot to hedgehog is modifying remains secret. We prove the zero knowledge property of the proof (where the ballot is a witness) in the next chapter.
- **Final Tally Disruption**. Because of threshold decryption, the final tally cannot be disrupted by a single or small set of non-colluding trustees.

3.7 Implementation

The VoteXX protocol has been fully implemented with four main objectives in mind:

1. The software must be open-source, period.
2. Software Maintenance was a crucial priority in the choices that have been made over the implementation language, package selection and design.
3. Modules have been partitioned based on the *Principle of Least Privilege*.
4. Test-Driven implementation to reduce corner case bugs. The Cryptographic core has been implemented in both *Mathematica* and *Java* and the random test set have been checked and verified on both implementations.

There have been nine modules that have been designed.

1. Mixnet Bridge Library. This library has be used in all the softwares in the system and is responsible to connect to a mixnet gateway.

2. TCP/IP Bridge Library. As as previous library, this library works as a bridge to Web as secondary communication channel between the election entities.

3. CryptoCore Library. Every cryptography-related functions have been packaged in this library. The functions have been enumerated below.

   (a) $\mathsf{Sig}(\mathsf{msg}, \mathsf{secretKey})$ is the Elgamal digital signature function with message recovery and $\mathsf{Verify}_{\mathsf{sig}}(\mathsf{signature}, \mathsf{publicKey})$ is the corresponding verification function.

   (b) $\mathsf{Enc}()$ and $\mathsf{Dec}()$, Exponential Elgamal encryption and decryption functions that has been already explained in Section 2.3.2.

   (c) $\mathsf{Prf}_{\mathsf{Sch}}()$ is the function to construct a fiat-shamir Schnorr proof (Section 2.3.6) and $\mathsf{Verify}_{\mathsf{Sch}}()$ is the verification function.

   (d) $\mathsf{Prf}_{\mathsf{Cha}}()$ is the function to construct a fiat-shamir Chaum-Pedersen proof (Section 2.3.6) and $\mathsf{Verify}_{\mathsf{Cha}}()$ is the verification function.

   (e) $\mathsf{Prf}_{\mathsf{Shuffle}}()$ is Σ-protocol *proof-of-shuffle* construction function.$\mathsf{Verify}_{\mathsf{shuffle}}()$ is the verification function that will be described later in Section 4.3.2.

4. Bulletin Board software. This software will be launched on the public, immutable and unstoppable mechanism [3] and based on the Election Timetable, it will grant write access to different election entities. The integrity checks have been implemented in this software including verification of the signatures and

[3]such as Ethereum

proofs. It is also responsible to deliver election's public information to election entities.

5. Registrars' software. This is a thin piece of code to allow registrars to sign the registration packets and send it toward the bulletin board software.

6. Election Authority software. This software is responsible to handle EA's actions, such as signing the votes and tally and also performing multi-party processes such as Threshold Elgamal and Plaintext Equality Test.

7. Voting software. This software lets the voter to generate new keys for the election, participate in the registration and voting phases and also to export her keys for the Nullification phase.

8. Nullification software. Constructing Nullification proofs occurs here. We will talk more about nullification in Chapter 4.

9. Observer software. This software is to retrieve any public information from the bulletin board.

Chapter 4

Nullification

"*He that has eyes to see and ears to hear may convince himself that no mortal can keep a secret. If his lips are silent, he chatters with his fingertips; betrayal oozes out of him at every pore.*" - Sigmund Freud

Nullification is the novel idea to give the voter a trust-worthy re-voting opportunity to *flip* her vote. Two main consequences of this situation would be

- *voter* will be able to flip the vote in the case of strong (but not full) coercion during the voting period.
- *voter* will be able to gain money from deceiving a Coercer. This property incentivizes the voters to guard the integrity of the election.

In this chapter, we will look into the rationale, design, proof sketch and discussions on the *Nullification* add-on—the main contribution of this thesis—and we would apply it to the VoteXX voting protocol that has been explained in the previous chapter.

4.1 Design

In this section, the details of the *Nullification Protocol* have been described. For that, we assume that all the previous phases of the election, namely *Setup*, *Registration* and *Voting*, have been completed without any issue. First, we formally and shortly describe the initial state of the nullification.

4.1.1 Initial State

We assume that at the time t_{nul},

- The tentative tally has been published. So that, The *Bulletin board* contains (and not limited to):

$$\{\mathsf{BB}_{\mathsf{reg}}, \mathsf{BB}_{\mathsf{TT}}, \mathsf{BB}_{\mathsf{nul}}\} \tag{8}$$

  where $\mathsf{BB}_{\mathsf{reg}}$ includes two sets of public keys, BB_{Y} and BB_{N}. BB_{Y} is the list of registered voters' Y public keys. Formally,

$$\mathsf{BB}_{\mathsf{Y}} = \{\mathsf{pk}(\mathsf{v}_1, \mathsf{Y}), \mathsf{pk}(\mathsf{v}_2, \mathsf{Y}), \cdots, \mathsf{pk}(\mathsf{v}_i, \mathsf{Y})\} \tag{9}$$

  Similarly, BB_{N} is the list of N public keys of all registered voters.

$$\mathsf{BB}_{\mathsf{N}} = \{\mathsf{pk}(\mathsf{v}_1, \mathsf{N}), \mathsf{pk}(\mathsf{v}_2, \mathsf{N}), \cdots, \mathsf{pk}(\mathsf{v}_i, \mathsf{N})\} \tag{10}$$

  where $\mathsf{pk}(\mathsf{v}_i, \mathsf{Y})$ is the i^{th} voter's identity (public key) that had registered to the election. Also, $\mathsf{BB}_{\mathsf{TT}}$ contains the signed tentative tally and $\mathsf{BB}_{\mathsf{nul}}$ would be an empty set, since no Nullification packet has been submitted yet.

- Depending on the voter's choice, voter v_i has submitted the voting request containing the ballot, $\mathsf{ballot}(\mathsf{v}_i)$, in the form of

$$\mathsf{ballot}(\mathsf{v}_i) = \mathsf{Sig}(\mathsf{EID}, \mathsf{sk}(\mathsf{v_i}, \mathsf{Y})) \tag{11}$$

OR

$$\mathsf{ballot}(\mathsf{v}_i) = \mathsf{Sig}(\mathsf{EID}, \mathsf{sk}(\mathsf{v_i}, \mathsf{N})) \tag{12}$$

where $\mathsf{sk}(\mathsf{v}, \mathsf{Y})$ and $\mathsf{sk}(\mathsf{v}, \mathsf{N})$ are the *privatekey* that v can use to sign the ballot $\mathsf{ballot}(\mathsf{v}_i)$, depending on her choice.

- The voter, v_i also has shared the corresponding secretkey the key she had signed the ballot with (i.e. $\mathsf{sk}(\mathsf{v}, \mathsf{Y})$ or $\mathsf{sk}(\mathsf{v}, \mathsf{N})$) with a trusted *Nullifier*[1].

4.1.2 Single-Voter Construction

For the simplicity, we assume a single-voter voting system. Therefore, $\mathsf{BB_Y}$ and $\mathsf{BB_N}$ would be

$$\mathsf{BB_Y} = \{\mathsf{pk}(\mathsf{v}, \mathsf{Y})\} \tag{13}$$

Also, same as $\mathsf{BB_Y}$, $\mathsf{BB_N}$ contains only one public key.

$$\mathsf{BB_N} = \{\mathsf{pk}(\mathsf{v}, \mathsf{N})\} \tag{14}$$

We want to design a cryptographic primitive to empower a voter's proxy (*Nullifier*) to anonymously flip [2] her vote when she is under coercion. We assume that the coercion level is below *Over-the-Shoulder* and voter v had a small period of booth-level privacy to contact a trustee (Nullifier) and share the voting *secretkey* $\mathsf{sk}(\mathsf{v}, \mathsf{Y})$ or

[1]Note that the voter herself can play the role of the Nullifier.

[2]We will discuss different versions of Nullification that support other actions such as *Cancel*. Here, for simplicity, we focus on the Flip nullification design.

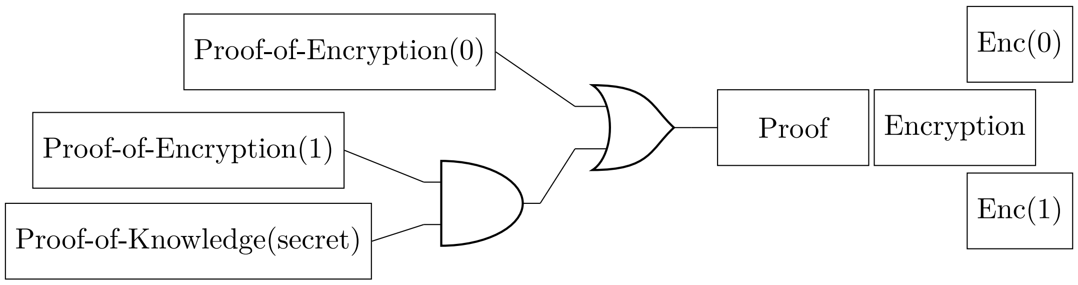

Figure 2: Nullification Receipt Construction

$\mathsf{sk}(\mathsf{v}, \mathsf{N})$.

By the functionality perspective, *Nullifier*'s target is to flip the vote when needed. Note that performing this action itself can reveal the voter's identity. So we design the *Nullification* logic circuit to enable the *Nullifier* to perform one of these two actions in zero-knowledge fashion.

1. *Nullifier* can flip voter's vote.

2. *Nullifier* can fake flipping voter's vote.

In Section 2.3.6, basic Σ-protocols such as *Schnorr* and *Chaum-Pedersen* have been described. We use these primitives to deliver the desired functionality.

To construct a valid Nullification receipt,$\mathcal{N}_{\mathsf{v}}$, (Figure 2), Nullifier sends an encryption of 0 or 1 beside the Nullification proof, Γ. We used $\mathsf{Enc}(x, [\alpha])$ notation for the *Exponential Elgamal* encryption function, So for instance, $\mathsf{Enc}(0)$ would be

$$\mathsf{Enc}(0, \alpha) = \langle \mathsf{C}_1, \mathsf{C}_2 \rangle = \langle g^{\alpha}{}_{mod\,p}\ ,\ g^0 \cdot h^{\alpha}{}_{mod\,p} \rangle \tag{15}$$

where x is the plaintext and α is the random value in $Z_{\{2,\,q-2\}}$ that has been used to generate the *ciphertext.* Note a few points:

1. All encryptions would be under EA's public key, h.

2. α would be used as the *witness*, ω that *Nullifier* will try to prove knowing it through the Chaum-Pedersen part of nullification circuit.

3. Using *Exponential Elgamal* enable *multi-Nullifier-single-Voter* and *multi-Nullification-single-Nullifier* mechanisms that give more freedom to voter/Nullifier to lure the *coercer*[3].

For the construction of the receipt, *Nullifier* must be able to prove to an arbitrary *honest Verifier* whether

1. She had sent $\mathsf{Enc}(1)$ by constructing a valid Chaum-Pedersen proof of $\mathsf{Enc}(1)$ **AND** she knows the voter's secretkey by constructing a valid Schnorr proof-of-knowledge of voter's secretkey).

OR

2. She had sent $\mathsf{Enc}(0)$ by constructing a valid Chaum-Pedersen proof of $\mathsf{Enc}(0)$.

So we design the module so that a valid Nullification receipt would be in the form of

$$\mathcal{N}_{\mathsf{v}} = \langle \Gamma_{\mathsf{OR}},\ \ \mathsf{Enc}(b_{\mathsf{nul}}) \rangle \tag{16}$$

where

$$\Gamma_{\mathsf{OR}} = \mathsf{Prf}_{\mathsf{OR}}(\mathsf{sk}, \alpha) = \langle \Gamma_{\mathsf{AND}},\ \Gamma_{0},\ e_{\mathsf{OR}} \rangle \tag{17}$$

and

$$\Gamma_{\mathsf{AND}} = \mathsf{Prf}_{\mathsf{AND}}(\mathsf{sk}) = \langle \Gamma_{\mathsf{sch}},\ \Gamma_{1},\ e_{\mathsf{AND}} \rangle \tag{18}$$

Since we want to hide the *Nullifier*'s intent, we construct an OR circuit proof inside the Nullification Σ-protocol. To construct a valid Nullification proof, $\mathsf{Prf}()$, *Nullifier*

[3]There are other choices to design a additively homo-morphic cryptosystem and we will discuss and compare them in the extension and also discussion sections.

follows the algorithm described below.

$$\mathsf{Prf}_{\mathsf{OR}}(\mathsf{sk}, b_{\mathsf{nul}}) = \begin{cases} \text{for} \quad b_{\mathsf{nul}} = 1 \begin{cases} \text{1. Compute } \ \mathsf{Enc}(1, \alpha) \\ \text{2. Simulate } \ \Gamma_{\mathsf{0}} \\ \text{3. Compute } \ e_{\mathsf{OR}} \\ \text{4. Construct } \ \Gamma_{\mathsf{AND}} \end{cases} \\ \\ \text{for} \quad b_{\mathsf{nul}} = 0 \begin{cases} \text{1. Compute } \ \mathsf{Enc}(0, \beta) \\ \text{2. Simulate } \ \Gamma_{\mathsf{AND}} \\ \text{3. Compute } \ e_{\mathsf{OR}} \\ \text{4. Construct } \ \Gamma_{\mathsf{0}} \end{cases} \end{cases} \tag{19}$$

Basically, *Nullifier* simulates the part that she wants to fake proving, computes the overall challenge of the *OR statement*, e_{OR} and then completes the other side of the statement. In the other words, *Nullifier* commits to both of them in a way that the *Verifier*can just perceive that one of these two statements are correct, without revealing which. Here the computational details of the Nullifier and Verifier have been described for each of these two cases.

Usecase #1, Nullifying the vote

To flip Voter's vote, a *Nullifier*, knowing voting secretkey sk_{v}, election's public parameters $\langle p, q, g, h\rangle$ and the lists of all registered *voters' public keys* constructs *Nullification proof* Γ_{v} by following these steps:

1. **Calculating** $\mathsf{Enc}(1, \alpha)$

   Nullifier picks a random value $\alpha \in Z_{\{2,\, q-2\}}$ and constructs $\mathsf{Enc}(1)$ in the form of

   $$\langle \mathsf{C}_1, \mathsf{C}_2 \rangle = \mathsf{Enc}(1, \alpha) = \langle g^{\alpha}{}_{mod\, p} \,,\; g^{1} \cdot h^{\alpha}{}_{mod\, p} \rangle \tag{20}$$

   where the *Nullification bit*, b_{nul} equals to 1, enciphering the intent of the *Nullifier* to actually nullify the vote.

2. **Simulating** Γ_{0}

   Now, chosen α, since the *Nullifier* wants to fake encrypting 0, she *simulates* Γ_{0} by starting from a random challenge e_{0}, a random response z_{0}, and a random value r_{0} such that $e_{\mathsf{0}}, z_{\mathsf{0}}, r_{\mathsf{0}} \in Z_{\{2,\, q-2\}}$. Formally,

   $$\Gamma_{\mathsf{0}} = \mathsf{Prf}_{\mathsf{Cha}}(\mathsf{C}_1, \mathsf{C}_2, \alpha) = \langle a_{01}, a_{02}, e_{\mathsf{0}}, z_{\mathsf{0}} \rangle \tag{21}$$

   Then she *rewinds* the construction algorithm to calculate a_{01} and a_{02}, meaning

   $$a_{01} = g^{z_{\mathsf{0}}} \cdot (g^{\alpha})^{-e_{\mathsf{0}}}{}_{mod\, p} \tag{22}$$

   and

   $$a_{02} = h^{z_{\mathsf{0}}} \cdot (h^{\alpha})^{-e_{\mathsf{0}}}{}_{mod\, p} \tag{23}$$

3. **Committing to** e_{OR}

   At this state, *Nullifier* has completed the simulation of $\mathsf{Prf}_{\mathsf{Cha}}(0)$ proof by the computation of all four needed values $\langle a_{01},\ a_{02},\ e_{\mathsf{0}},\ z_{\mathsf{0}} \rangle$. We recall that for binding two Σ-protocols in a conjunction form, *Nullifier* needs to *simulate* or *rewind* one proof and then commit to a *Fiat-Shamir* commitment, e_{OR} that from *Verifier*'s perspective, it would be in the from of

$$e_{\mathsf{OR}} = e_{\mathsf{0}} + e_{\mathsf{AND}} \tag{24}$$

where e_{0} is the Fiat-shamir challenge for the proof $\mathsf{Prf}_{\mathsf{Cha}}(0)$ that *Nullifier* has already simulated it, and e_{AND} is the Fiat-shamir challenge for the proof $\mathsf{Prf}_{\mathsf{AND}}(\mathsf{sk})$. However, this is not how *Nullifier* actually does it. After calculating $a_{01}, a_{02} and e_{\mathsf{0}}$, she computes e_{OR}, the overall commitment for $\mathsf{Prf}_{\mathsf{OR}}(\mathsf{sk}, b_{\mathsf{nul}})$ by

$$e_{\mathsf{OR}} = \mathcal{H}(a_{01} \mid a_{02} \mid a_{11} \mid a_{12} \mid a_{\mathsf{sch}} \mid p \mid q \mid g \mid h)_{\ mod\, q} \tag{25}$$

where a_{11}, a_{12} would be blinded random values to complete Chaum-Pedersen in the form of

$$a_{11} = g^{r_1}{}_{mod\, p}, \quad a_{12} = h^{r_1}{}_{mod\, p} \tag{26}$$

and a_{sch} is the blinded random value to construct the proof of knowing the voter's secretkey, $\mathsf{sk_v}$,

$$a_{\mathsf{sch}} = g^{r_{\mathsf{sch}}}{}_{mod\, p} \tag{27}$$

where $r_1, r_{\mathsf{sch}} \in Z_{\{2,\, q-2\}}$ are the **same** random value that will be used to construct $\mathsf{Prf}_{\mathsf{Cha}}(1)$.

To understand what really is happening, note that *Nullifier* has already injected a fake blinded values a_{01} and a_{02} into the process of committing to e_{OR}. Then, she continues the *rewinding process* by injecting e_{0} into the overall commitment of the Nullification Module as well. Therefore, to find a valid challenge for $\mathsf{Prf}_{\mathsf{AND}}(\mathsf{sk})$, *Nullifier* uses the below equation.

$$e_{\mathsf{AND}} = e_{\mathsf{OR}} - e_{\mathsf{0}} \tag{28}$$

4. **Constructing** Γ_{AND}To remind, we are constructing a Σ-protocol that is a composition of one Chaum-Pedersen proof-of-encryption of *One* and one *Schnorr Proof-of-knowledge* of the *voter's* privatekey. Saying to more formally

$$\mathsf{Prf}_{\mathsf{AND}}(\mathsf{sk}) = \langle \Gamma_{\mathsf{sch}},\ \Gamma_1,\ e_{\mathsf{AND}} \rangle \tag{29}$$

where e_{AND} is strong fiat-shamir commitment of $\mathsf{Prf}_{\mathsf{AND}}(\mathsf{sk})$. Γ_{sch} is a *strong Fiat-Shamir heuristics* powered Schnorr proof. We recap the calculation of *strong Fiat-Shamir heuristics* powered Schnorr proof here.

Schnorr proof Construction

We recall that to construct a *Fiat-Shamir* powered Chaum-Pedersen proof the Prover has to send a proof containing

$$\langle a,\ e,\ z \rangle \tag{30}$$

where

$$a = g^r{}_{\mathit{mod}\, p} \tag{31}$$

and

$$e = H(a \mid p \mid q \mid g \mid h)_{\mathit{mod}\, q} \tag{32}$$

is the *strong Fiat-Shamir heuristics* challenge and H is a secure hash function. Also, the Σ-Protocol's response, z is in the form of

$$z = r + \omega \cdot e \tag{33}$$

Nullifier knowing the steps, constructs $\mathsf{Prf}_{\mathsf{Sch}}(\mathsf{sk})$ in the form of

$$\Gamma_{\mathsf{sch}} = \langle a_{\mathsf{sch}},\ e_{\mathsf{sch}},\ z_{\mathsf{sch}} \rangle \tag{34}$$

where a_{sch} has already been computed and z_{sch} is the response to the challenge(or fiat-shamir commitment) e_{sch}. For Composing two Σ-protocol, *Nullifier* uses the same challenge for the both Σ-Protocols. Moreover, from equation 28, *Nullifier* is forced to use e_{AND} in order to construct a valid nullification receipt. Therefore, Nullifier uses

$$e_{\mathsf{sch}} = e_{\mathsf{AND}} \tag{35}$$

in the construction of Γ_{sch}. Also, trivially, z_{sch} would be

$$z_{\mathsf{sch}} = r_{\mathsf{sch}} + \omega \cdot e_{\mathsf{sch}} = r_{\mathsf{sch}} + \mathsf{sk} \cdot e_{\mathsf{AND}} \tag{36}$$

where ω is the voter's secretkey that has been already shared with the *Nullifier*. Also, *Nullifier* will construct the Chaum-Pedersen proof to prove that she is sending an encryption of one in the nullification receipt, Γ_v. Here we recap the normal construction of a single Chaum-Pedersen proof of knowledge with Fiat-shamir heuristics commitment/challenge.

Chaum-Pedersen proof construction

To construct a *Fiat-Shamir* powered Chaum-Pedersen proof , the *Prover*, $\mathcal{P}$ has to send a proof containing

$$\langle a_1,\ a_2,\ e,\ z\rangle \tag{37}$$

where

$$a_1 = g^r{}_{\,mod\, p},\ \ a_2 = h^r{}_{\,mod\, p} \tag{38}$$

and

$$e = H(a_1 \mid a_2 \mid p \mid q \mid g \mid h)_{\,mod\, q} \tag{39}$$

is the *strong Fiat-Shamir heuristics* challenge where H is a hash function. Also, the Σ-Protocol's response z is in the form of

$$z = r + \omega \cdot e \tag{40}$$

Nullifier has already computed a_3 and a_4. As we have explained earlier, she uses e_{AND} to compute z_1 in the form of

$$z_1 = r_1 + \omega \cdot e_1 = r_1 + \alpha \cdot e_{\mathsf{AND}} \tag{41}$$

At this point, *Nullifier* has finished the construction of Γ_{v}. ■

Usecase #2, Faking the Vote Nullification

To fake the nullification, *Nullifier* follows the below steps.

1. **Calculating $\mathsf{Enc}(0, \beta)$**

   In this case, *Nullifier* constructs $\mathsf{Enc}(0, \beta)$ in the form of

$$\mathsf{Enc}(0, \beta) = \langle \mathsf{C}_1, \mathsf{C}_2 \rangle = \langle g^{\beta}{}_{mod\,p}\ ,\ g^0 \cdot h^{\beta}{}_{mod\,p} \rangle \tag{42}$$

   where $\beta \in Z_{\{2,\,q-2\}}$ is a freshly chosen random value. Note that *Nullifier* is sending *Nullification bit*, $\mathsf{b}_{\mathsf{nul}}$ equals to 0, encrypting the intent of the *Nullifier* to fake the nullification.

2. **Simulating Γ_{AND}**

   Same as the previous case, *Nullifier* starts the construction of Γ_{v} by simulating the side of disjunction that she does not want to complete, meaning $\mathsf{Prf}_{\mathsf{AND}}(\mathsf{sk})$. For that purpose, she rewinds the construction protocol to calculate a_{11}, a_{12} by

$$a_{11} = g^{z_{\mathsf{AND}}} \cdot (g^{\beta})^{-e_{\mathsf{AND}}}{}_{mod\,p}\ ,\ \ a_{12} = h^{z_{\mathsf{AND}}} \cdot (h^{\beta} \cdot g^{-1})^{-e_{\mathsf{AND}}}{}_{mod\,p} \tag{43}$$

where $e_{\mathsf{AND}}, z_{\mathsf{AND}} \in Z_{\{2,\, q-2\}}$ are two random numbers that respectively play the role of the challenge and response to $\mathsf{Prf}_{\mathsf{AND}}(\mathsf{sk})$. Also, for simulating $\mathsf{Prf}_{\mathsf{Sch}}(\mathsf{sk})$, *Nullifier* computes a_{sch} by

$$a_{\mathsf{sch}} = g^{z_{\mathsf{AND}}} \cdot \left(g^{\mathsf{sk}}\right)^{-e_{\mathsf{AND}}}{}_{mod\, p} \tag{44}$$

3. **Committing to e_{OR}**

   Same as previous usecase, here, *Nullifier* computes the overall Fiat-Shamir commitment for $\mathsf{Prf}_{\mathsf{OR}}(\mathsf{sk}, b_{\mathsf{nul}})$ by

$$e_{\mathsf{OR}} = \mathcal{H}(a_{01} \mid a_{02} \mid a_{11} \mid a_{12} \mid a_{\mathsf{sch}} \mid p \mid q \mid g \mid h)_{\,mod\, q} \tag{45}$$

   Contrary to the previous usecase, here *Nullifier* have computed a_{11}, a_{12} and a_{sch} and has to compute valid a_{01} and a_{02} values by

$$a_{01} = g^{r_0}{}_{mod\, p}, \quad a_{02} = h^{r_0}{}_{mod\, p} \tag{46}$$

   where $r_0 \in Z_{\{2,\, q-2\}}$ is the random value that will be used in the proof construction of $\mathsf{Prf}_{\mathsf{Cha}}(0)$. In this case *Nullifier* has simulated $\mathsf{Prf}_{\mathsf{AND}}(\mathsf{sk})$ and wants to construct a valid proof for $\mathsf{Prf}_{\mathsf{Cha}}(0)$, she calculates a valid challenge, e_0 in the form of

$$e_0 = e_{\mathsf{OR}} - e_{\mathsf{AND}} \tag{47}$$

   where e_{AND} is the challenge she calculated in the previous step.

4. **Constructing Γ_0**

In the last step, *Nullifier* constructs the receipt for $\mathsf{Prf}_{\mathsf{Cha}}(0)$ in the form of

$$\langle a_{01}, a_{02}, e_{\mathsf{0}}, z_{\mathsf{0}} \rangle \tag{48}$$

where a_{01}, a_{02}, e_{0} have already been calculated and z_{0} would be

$$z_{\mathsf{0}} = r_{\mathsf{0}} + \omega \cdot e_{\mathsf{0}} = r_{\mathsf{0}} + \beta \cdot e_{\mathsf{0}} \tag{49}$$

At this point *Nullifier* has finished constructing a valid receipt to fake the vote cancellation.■

4.1.3 Receipt Verification

At this state, assume an honest *Verifier* $\mathcal{V}$, receiving $\mathcal{N}_{\mathsf{v}}$ in the form of equation 16 verifies the validity of it by checking the conditions below.

1. Γ_{0} must be a valid *Chaum-Pedersen* proof in the form of

$$\Gamma_{\mathsf{0}} \stackrel{?}{=} \langle a_{01}, a_{02}, e_{\mathsf{0}}, z_{\mathsf{0}} \rangle \tag{50}$$

   To check that, For that $\mathcal{V}$ must verify whether

$$a_{01} \stackrel{?}{=} g^{z_{\mathsf{0}}} \cdot \mathsf{C_1}^{-e}{}_{mod\, p} \tag{51}$$

   and

$$a_{02} \stackrel{?}{=} h^{z_{\mathsf{0}}} \cdot \mathsf{C_2}^{-e}{}_{mod\, p} \tag{52}$$

2. Γ_{AND} must be a valid Conjunction proof of one chaum-pedersen and one schnorr

Σ-protocol in the form of

$$\Gamma_{\mathsf{AND}} \stackrel{?}{=} \langle \Gamma_1, \Gamma_{\mathsf{sch}}, e_{\mathsf{AND}} \rangle \tag{53}$$

and these conditions must fulfilled.

(a) Γ_1 must be another valid *Chaum-Pedersen* commitment in the form of

$$\Gamma_1 \stackrel{?}{=} \langle a_{11}, a_{12}, e_1, z_1 \rangle \tag{54}$$

$$a_{11} \stackrel{?}{=} g^{z_1} \cdot (\mathsf{C}_1)^{-e}{}_{mod\, p} \tag{55}$$

and

$$a_{12} \stackrel{?}{=} h^{z_1} \cdot (g^{-1} \cdot \mathsf{C}_2)^{-e}{}_{mod\, p} \tag{56}$$

(b) Γ_{sch} is a valid *Schnorr* proof in the form of

$$\Gamma_{\mathsf{sch}} \stackrel{?}{=} \langle a_{\mathsf{sch}}, e_{\mathsf{sch}}, z_{\mathsf{sch}} \rangle \tag{57}$$

and

$$a_{\mathsf{sch}} \stackrel{?}{=} g^{z_{\mathsf{sch}}} \cdot \mathsf{pk}(\mathsf{v}_i, \mathsf{Y}/\mathsf{N})^{-c}{}_{mod\, p} \tag{58}$$

(c) To verify that the *Nullifier* has bound $\mathsf{Prf}_{\mathsf{Cha}}(1)$ and Γ_{sch} in a proper way, $\mathcal{V}$ check whether

$$e_1 \stackrel{?}{=} e_{\mathsf{sch}} \tag{59}$$

3. The commitment e_{OR} must binding $\mathsf{Prf}_{\mathsf{Cha}}(0)$ and $\mathsf{Prf}_{\mathsf{AND}}(\mathsf{sk})$ properly. Meaning, $\mathcal{V}$ must check whether

$$e_{\mathsf{OR}} \stackrel{?}{=} e_{\mathsf{AND}} + e_0 \tag{60}$$

Also, $\mathcal{V}$ checks whether e_{OR} is a valid Fiat-Shamir challenge or not. Therefore, she will check

$$e_{\mathsf{OR}} \overset{?}{=} \mathcal{H}(a_{01} \mid a_{02} \mid a_{11} \mid a_{12} \mid a_{\mathsf{sch}} \mid p \mid q \mid g \mid h)_{\mathit{mod}\, q} \tag{61}$$

4.1.4 Expanding to N Voters

In the previous section, the Nullification circuit for a single-voter election has been explained. We can see that there are still weak spots in the protocols.

1. The intent of the Nullifier of sending the Nullification receipt is still visible to the Coercer.
2. Based on pre-tally and tally states of the bulletin board, the nullification bit would be revealed to the Coercer.

A trivial way to anonymously nullify a vote is to send a Nullification receipt per all the voters that had registered in the election. For N voter election, we assume that the election with public values $\langle p, q, g, h\rangle$ and n registered voters has been completed successfully to pre-tally state. Therefore the $\mathsf{BB_Y}$ would be in the state

$$\mathsf{BB_Y} = \{\mathsf{pk}(\mathsf{v}_1, \mathsf{Y}), \mathsf{pk}(\mathsf{v}_2, \mathsf{Y}), \cdots, \mathsf{pk}(\mathsf{v}_i, \mathsf{Y}), \cdots, \mathsf{pk}(\mathsf{v}_n, \mathsf{Y})\} \tag{62}$$

Also, same as $\mathsf{BB_Y}$, $\mathsf{BB_N}$ contains only one public key.

$$\mathsf{BB_N} = \{\mathsf{pk}(\mathsf{v}_1, \mathsf{Y}), \mathsf{pk}(\mathsf{v}_2, \mathsf{Y}), \cdots, \mathsf{pk}(\mathsf{v}_i, \mathsf{Y}), \cdots, \mathsf{pk}(\mathsf{v}_n, \mathsf{Y})\} \tag{63}$$

We assume two cases

1. *Nullifier* knows the i^{th} voter's secretkey and intents to nullify her vote. To

nullify a vote, a *Nullifier* constructs an array of $\mathcal{N}_{\mathsf{v}_i}$ in the form of

$$\overrightarrow{\mathcal{N}} = \{\mathcal{N}_1, \mathcal{N}_2, \cdots, \mathcal{N}_i, \cdots, \mathcal{N}_n\} \tag{64}$$

where she constructs

$$\mathcal{N}_i = \langle \Gamma_{\mathsf{v}_i}, \mathsf{Enc}(1) \rangle \tag{65}$$

and Γ_{v_i} would be in the form of use-case#1. Also, *Nullifier* constructs

$$\mathcal{N}_{j \neq i} = \langle \Gamma_{\mathsf{v}_j}, \mathsf{Enc}(0) \rangle \tag{66}$$

using the instruction that has been explained in usecase#2.

2. *Nullifier* wants to fake nullifying a voter's vote. So, without any private knowledge, she constructs

$$\overrightarrow{\mathcal{N}} = \{\mathcal{N}_1, \mathcal{N}_2, \cdots, \mathcal{N}_n\} \tag{67}$$

where

$$\mathcal{N}_i = \langle \Gamma_i, \mathsf{Enc}(0) \rangle \tag{68}$$

for $i \in \{1, 2, ..., n\}$ and sends $\overrightarrow{\mathcal{N}}$.

Note that each *Nullifier* may attempt to nullify more than one vote, if she knows the corresponding secretkeys.

4.1.5 Aggregation

To this point we have explained how *Nullifier* can anonymously nullify the vote by sending a Nullification request in the form of Eq.64. On the EA side, at the end of voting, EA would sign and publish all the received Nullification requests on the bulletin board, $\mathsf{BB}_{\mathsf{nul}}$ where

$$\mathsf{BB}_{\mathsf{nul}} = \{\overrightarrow{\mathcal{N}}_1, \overrightarrow{\mathcal{N}}_2, \cdots, \overrightarrow{\mathcal{N}}_m\} \tag{69}$$

where m is the number of received Nullification packets. Note that all the Nullification packets are in the same size and contain a list of $\overrightarrow{\mathcal{N}}_{\mathsf{v}_i}$ in the same order. Therefore, by expanding each Nullification packet, we get Table 1.

Table 1: Bulletin Board Nullification table

	v_1	v_2	$\cdots$	v_i	$\cdots$	v_n
$\overrightarrow{\mathcal{N}}_1$	$\mathcal{N}_{[1,1]}$	$\mathcal{N}_{[1,2]}$	$\cdots$	$\mathcal{N}_{[1,i]}$	$\cdots$	$\mathcal{N}_{[1,n]}$
$\overrightarrow{\mathcal{N}}_2$	$\mathcal{N}_{[2,1]}$	$\mathcal{N}_{[2,2]}$	$\cdots$	$\mathcal{N}_{[2,i]}$	$\cdots$	$\mathcal{N}_{[2,n]}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\overrightarrow{\mathcal{N}}_j$	$\mathcal{N}_{[j,1]}$	$\mathcal{N}_{[j,2]}$	$\cdots$	$\mathcal{N}_{[j,i]}$	$\cdots$	$\mathcal{N}_{[j,n]}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\overrightarrow{\mathcal{N}}_m$	$\mathcal{N}_{[m,1]}$	$\mathcal{N}_{[m,2]}$	$\cdots$	$\mathcal{N}_{[m,i]}$	$\cdots$	$\mathcal{N}_{[m,n]}$

Here, we describe the naive Nullification Aggregation idea; However, various aggregations can be implemented that have been explained in Section 4.3.2. Each $\mathcal{N}_j(\mathsf{v}_i)$ contains an *Exponential ElGamal* encryption is the form of

$$\mathcal{N}_j(\mathsf{v}_i) = \langle \Gamma(j,i) \, , \, \mathsf{Enc}(\mathsf{flip}_{[j,i]}) \rangle \tag{70}$$

where $\mathsf{flip}_{[i,j]} \in \{0,1\}$ is the nullification bit that has been sent by j^{th} *Nullifier* for the ballot corresponding to v_i. Note that the bit is concealed through the Nullification phase. Also, by the new indexing, $\mathsf{Enc}(\mathsf{flip}_{[j,i]})$ would be

$$\mathsf{Enc}(\mathsf{flip}_{[j,i]}) = \langle \mathsf{C}_1(j,i), \mathsf{C}_2(j,i) \rangle = \langle g^{\mathsf{rand(j,i)}} \ , \ g^{\mathsf{flip}_{[j,i]}} \cdot h^{\mathsf{rand(j,i)}} \rangle \tag{71}$$

Table 2: Bulletin Board Nullification sub-table of encryption

	v_1	$\cdots$	v_i	v_n
$\overrightarrow{\mathcal{N}}_1$	$\langle \mathsf{C}_1(1,1), \mathsf{C}_2(1,1) \rangle$	$\cdots$	$\langle \mathsf{C}_1(1,i), \mathsf{C}_2(1,i) \rangle$	$\langle \mathsf{C}_1(1,n), \mathsf{C}_2(1,n) \rangle$
$\overrightarrow{\mathcal{N}}_2$	$\langle \mathsf{C}_1(2,1), \mathsf{C}_2(2,1) \rangle$	$\cdots$	$\langle \mathsf{C}_1(2,i), \mathsf{C}_2(2,i) \rangle$	$\langle \mathsf{C}_1(2,n), \mathsf{C}_2(2,n) \rangle$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\overrightarrow{\mathcal{N}}_j$	$\langle \mathsf{C}_1(j,1), \mathsf{C}_2(j,1) \rangle$	$\cdots$	$\langle \mathsf{C}_1(j,i), \mathsf{C}_2(j,i) \rangle$	$\langle \mathsf{C}_1(j,n), \mathsf{C}_2(j,n) \rangle$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\overrightarrow{\mathcal{N}}_m$	$\langle \mathsf{C}_1(m,1), \mathsf{C}_2(m,1) \rangle$	$\cdots$	$\langle \mathsf{C}_1(m,i), \mathsf{C}_2(m,i) \rangle$	$\langle \mathsf{C}_1(m,n), \mathsf{C}_2(m,n) \rangle$

The unwrapped Nullification table has been shown in Table 4.1.5. Based on additively homomorphism of $\mathsf{Enc}()$, we know that by multiplying C_1 and also C_2 we get valid encryption of the summation of all bits that have been sent by any *Nullifier*. Formally

$$\overrightarrow{\mathcal{N}}_{\mathsf{total}} = \prod_{j=1}^{m} \overrightarrow{\mathcal{N}}_j = \langle \prod_{j=1}^{m} \mathcal{N}_{[j,1]}, \prod_{j=1}^{m} \mathcal{N}_{[j,2]}, \cdots, \prod_{j=1}^{m} \mathcal{N}_{[j,n]} \rangle \tag{72}$$

So, we will have

$$\begin{aligned}
\mathsf{Enc}(\mathsf{total}, i) &= \langle \mathsf{C_1(total, i)}, \mathsf{C_2(total, i)} \rangle \\
&= \langle \prod_{j=1}^{m} \mathsf{C_1}(\mathsf{v}_i), \quad \prod_{j=1}^{m} \mathsf{C_2}(\mathsf{v}_i) \rangle \\
&= \langle \prod_{j=1}^{m} g^{\mathsf{rand(j,i)}}{}_{mod\, p}, \quad \prod_{j=1}^{m} g^{\mathsf{flip}_{[j,i]}} \cdot h^{\mathsf{rand(j,i)}}{}_{mod\, p} \rangle \\
&= \langle g^{\sum_{j=1}^{m} \mathsf{rand(j,i)}}{}_{mod\, p}, \quad g^{\sum_{j=1}^{m} \mathsf{flip}_{[j,i]}} \cdot h^{\sum_{j=1}^{m} \mathsf{rand(j,i)}}{}_{mod\, p} \rangle \\
&= \langle g^{\mathsf{rand(total,i)}}{}_{mod\, p}, \quad g^{\mathsf{flip_{[total,i]}}} \cdot h^{\mathsf{rand(total,i)}}{}_{mod\, p} \rangle
\end{aligned} \tag{73}$$

where $i \in \{1, 2, \cdots, n\}$. $\mathsf{Enc}(\mathsf{total}, i)$ is a valid *Exponential ElGamal* output and $\mathsf{flip_{[total]}}$ is the summation of number of successful flip per voter. The decryption would happen in the Tally phase.

$$\mathsf{Dec}(\langle \mathsf{C_1}(\mathsf{total}, v_i), \mathsf{C_2}(\mathsf{total}, v_i) \rangle) = g^{\mathsf{flip}_{[\mathsf{total}, v_i]}}{}_{mod\, p} \tag{74}$$

Since there is no efficient way to find $\mathsf{flip}_{[\mathsf{total}, v_i]}$ from the above equation, the system would fail to compute the real number of total flips per that specific voter. The problem is the decrypted value would be g^{total} and there is no known efficient way to compute total from g^{total} as it has been described in *Decisional Diffie Helman* problem, Section 2.3. Note that this is a general issue with additively homomorphic Elgamal and the solution is to ensure any additions do not get too large to find through exhaustive search (*e.g.,* a trillion or less). Our bounds the value by the number of hedgehog nullifications which we can assume in practice to be small enough.

One approach is to pre-compute a Lookup table in the form of Table 3, before the election. In the table, we can see that every encryption has been paired with a plaintext. In Table 3, the computation of the number of the flips for each voter has been described. For each voter, v_i, a check would be performed per each row of the

Table 3: Nullification Lookup table, a closer look

IF	$[g^{\mathsf{flip}_{[\mathsf{total},i]}} \overset{?}{=} g^0]$	THEN	$\mathsf{flip}_{[\mathsf{total},i]} = 0$	$\Rightarrow$	NO	
IF	$[g^{\mathsf{flip}_{[\mathsf{total},i]}} \overset{?}{=} g^1]$	THEN	$\mathsf{flip}_{[\mathsf{total},i]} = 1$	$\Rightarrow$	YES	
...	...	...	...	...	...	
IF	$[g^{\mathsf{flip}_{[\mathsf{total},i]}} \overset{?}{=} g^j]$	THEN	$\mathsf{flip}_{[\mathsf{total},i]} = j$	$\Rightarrow$	YES	$(j \in \mathbf{N}_{\mathsf{odd}})$
...	...	...	...	...	...	
IF	$[g^{\mathsf{flip}_{[\mathsf{total},i]}} \overset{?}{=} g^k]$	THEN	$\mathsf{flip}_{[\mathsf{total},i]} = k$	$\Rightarrow$	NO	$(k \in \mathbf{N}_{\mathsf{even}})$

look-up table. Note that the size of the look-up table is a constant k, denoting the limitation on the number of flips for a voter. More formally,

$$k = max[\mathsf{flip}_{[\mathsf{total},\mathsf{v}_i]}] \;\; where \;\; i \in \{1, 2, ..., n\} \tag{75}$$

where n is the number of the registered voters. So this design would be vulnerable to *Denial-of-Service* attack when a voter or Nullifier tries to send successful Nullification packet for an arbitrary voter for an unusually number of times. We will discuss the possible mitigations in the following sections.

4.1.6 Complexity

To calculate the complexity of each phase of the nullification protocol, we count the number of *modular exponentiations*, the most expensive operation through all the proposed processes. In Table 4, we have listed the complexity for each sub routines.

Table 4: Number of exponentiations for each Nullification phase

Phase	# of exponentiations
Construction of flipping nullification packet	$O(N_{voter})$
Construction of non-flipping nullification packet	$O(N_{voter})$
Verification of Nullification packet	$O(N_{voter})$
Aggregation of Nullification Packets	$O(1)$

4.2 Proof Sketch

In the previous section, we have described the basic design of the *Nullification Protocol.* In this section, we prove that the proposed *Nullification module* is an *Honest-Verifier Zero-Knowledge Proof-of-knowledge* protocol. In order to formally prove that, we have to prove the designed nullification module is a $\Sigma - protocol$. We followed Lindell's Efficient secure two-party protocols [34] notation.

4.2.1 Terminology

First, we prove the interactive version of the Nullification Protocol, dubbed $\hat{\mathsf{Prf}}()$ is a Σ-protocol. To prove that, we have to prove three basic properties in designed protocol that have been explained in Section 2.3, namely, *Completeness*, *Special Soundness* and *Special Honest Verifier Zero-knowledge.* To start the journey, we have to translate some notations.

Prover

Prover, $\mathcal{P}$, would be the *Nullifier* that is constructing the package. Malicious prover has been denoted by $\mathcal{P}^*$.

Verifier

Verifier, $\mathcal{V}$, would be the Election Authorities or a third trusted party to handle the received Nullification Packets. We denoted the malicious verifier by $\mathcal{V}^*$.

Problem

Problem, x, would be proving to the Verifier, $\mathcal{V}$ that the sent Nullification Packet is a valid packet.

$$x_{\mathsf{nul}} = \langle p, q, g, h, \mathsf{BB_Y}, \mathsf{BB_N}, \mathsf{Enc}(\mathsf{flip}_{[\mathsf{v}_i]}) \rangle \tag{76}$$

where $\mathsf{BB_Y}, \mathsf{BB_N}$ have been described in Eq.9 and Eq.10 respectively. $\mathsf{flip}_{[\mathsf{v}_i]} \in \{0, 1\}$ shows the bit that has been sent by the *Nullifier*.

Witness

Witness, ω, the information that $\mathcal{P}$ wants to prove knowing it without revealing any additional information. ω would be voter's secretkey, $\mathsf{sk}(\mathsf{v}_i, \mathsf{chc}(\mathsf{v_i}))$, $\mathsf{flip}_{[\mathsf{v}_i]}$ and α.

$$\omega_{\mathsf{nul}} = \langle \mathsf{sk}(\mathsf{v}_i, \mathsf{chc}(\mathsf{v_i})), \mathsf{flip}_{[\mathsf{v}_i]}, \alpha \rangle \tag{77}$$

where $\mathsf{chc}(\mathsf{v_i}) \in \{\mathsf{Y}, \mathsf{N}\}$ denotes the voter key of choice for voting.

Relation

Relation, $\mathcal{R}_{\mathsf{nul}}$, would be a relation over $\{0, 1\}^t$ to $\{0, 1\}^t$ where includes x to ω.

$$\mathcal{R}_{\mathsf{nul}} = \{\mathcal{R} \mid (x_{\mathsf{nul}}, \omega_{\mathsf{nul}}) \in \mathcal{R}\} \tag{78}$$

Random Oracle

In cryptography, a random oracle is an oracle (a theoretical black box) that responds to every unique query with a (truly) random response chosen uniformly from its output domain. If a query is repeated, it responds the same way every time that query is submitted. It is equivalent to flipping n ideal coins, in response to a challenge, to generate a n-bit unique response per challenge.

Σ-Protocol

Σ-Protocol that we are proving would be the interactive version of the Nullification protocol. So we replace the prover-generated fiat-shamir commitment with a Verifier-generated challenge $e_{\mathcal{V}}$ to convert the protocol to a standard public-coin interactive protocol. So in the step 3 of the construction of the Nullification receipt Γ, the prover commits to the blinded random values $(a_{01}, a_{02}, a_{11}, a_{12}, a_{\mathsf{sch}})$ by sending them to the Verifier. Then, Verifier sends back a random value $e_{\mathcal{V}}$ which would be used to complete the proof by the Prover. The interactive version has been depicted in Figure 3.

Transcript

Transcript, Γ would be in the form of

$$\Gamma_{\mathsf{nul}} = (a_{\mathsf{nul}}, e_{\mathsf{nul}}, z_{\mathsf{nul}}) \tag{79}$$

where a_{nul} would be

$$a_{\mathsf{nul}} = (a_{01}, a_{02}, a_{11}, a_{12}, a_{\mathsf{sch}}) \tag{80}$$

$\mathcal{P}$ | $\mathcal{V}$

Computes a_{nul}

$a_{\mathsf{nul}} = (a_{01}, a_{02}, a_{11}, a_{12}, a_{\mathsf{sch}})$

$\xrightarrow{a_{\mathsf{nul}}}$

Generates challenge e_{nul}

$\xleftarrow{e_{\mathsf{nul}}}$

Generates the response z_{nul}

$z_{\mathsf{nul}} = (z_0, z_1, z_{\mathsf{sch}})$

$\xrightarrow{z_{\mathsf{nul}}}$

Verifies $(a_{\mathsf{nul}}, e_{\mathsf{nul}}, z_{\mathsf{nul}})$

Figure 3: Interactive version of Nullification Protocol

where a_{01} and a_{02}, the blinded values for $\mathsf{Prf}_{\mathsf{Cha}}(0)$ and would be in the form of

$$a_{01} = g^{r_0}{}_{mod\,p}\ ,\ a_{02} = h^{r_0}{}_{mod\,p} \tag{81}$$

Also, a_{11} and a_{12} are the blinded values needed to construct $\mathsf{Prf}_{\mathsf{Cha}}(1)$ in the form of

$$a_{11} = g^{r_1}{}_{mod\,p}\ ,\ a_{12} = h^{r_1}{}_{mod\,p} \tag{82}$$

a_{sch} would be the Schnorr's blinded value in the form of

$$a_{\mathsf{sch}} = g^{r_{\mathsf{sch}}}{}_{mod\,p} \tag{83}$$

and e_{nul} would be the challenge of the overall disjunction in the Nullification circuit.

$$e_{\mathsf{nul}} = e_{\mathsf{OR}} \tag{84}$$

The response z_{nul} would be

$$z_{\mathsf{nul}} = (z_0, z_1, z_{\mathsf{sch}}) \tag{85}$$

Algorithm

Algorithm, $\mathcal{A}$, would be a polynomial-time algorithm that given any x_{nul} and any pair of accepting transcripts $(a_{\mathsf{nul}}, e_{\mathsf{nul}}, z_{\mathsf{nul}})$ and $(a_{\mathsf{nul}}, e'_{\mathsf{nul}}, z'_{\mathsf{nul}})$ for x_{nul} where $e_{\mathsf{nul}} \neq e'_{\mathsf{nul}}$, outputs ω_{nul} such that $(x_{\mathsf{nul}}, \omega_{\mathsf{nul}}) \in \mathcal{R}_{\mathsf{nul}}$.

Simulator

Simulator, $\mathcal{M}$, a probabilistic polynomial-time simulator $\mathcal{M}$, which on input $x_{\mathsf{nul}}, e_{\mathsf{nul}}$ outputs a transcript Γ with the same probability distribution as transcripts between the honest $\mathcal{P}$ and $\mathcal{V}$ on common input x_{nul}. Formally, for every x and ω such that $(x, \omega) \in \mathcal{R}$ and every $e \in \{0,1\}^t$ it holds that

$$\{\mathcal{M}(x_{\mathsf{nul}}, e_{\mathsf{nul}})\} = \{\langle \mathcal{P}(x_{\mathsf{nul}}, e_{\mathsf{nul}}), \mathcal{V}(x_{\mathsf{nul}}, e_{\mathsf{nul}})\rangle\} \tag{86}$$

where $\mathcal{M}(x, \omega)$ denotes the output of simulator $\mathcal{M}$ upon input x and e, and $\mathcal{P}(x, w), \mathcal{V}(x, e)$ denotes the output transcript of an execution between $\mathcal{P}$ and $\mathcal{V}$, where $\mathcal{P}$ has input (x, ω), $\mathcal{V}$ has input x, and $\mathcal{V}$'s random tape (determining its query) equals e.
To prove that the interactive version of the*Nullification Protocol* is a Σ-protocol, we have to prove that it suffices *Completeness*, *Special Soundness* and *Special Honest Verifier zero knowledge.*

4.2.2 Completeness

As we have described in section 4.1.2, by following the steps. honest $\mathcal{V}$ has not choice but accepting the transcript that has been constructed properly. It is complete based on the Group theories that have been explained in section 2.3. So the *Nullification Protocol* will always accept If $\mathcal{P}$ and $\mathcal{V}$ follow the protocol on input problem x_{nul} and private input ω_{nul} to $\mathcal{P}$ where $(x_{\mathsf{nul}}, \omega_{\mathsf{nul}}) \in \mathcal{R}_{\mathsf{nul}}$, then $\mathcal{V}$ always accepts.

4.2.3 Special Soundness

We propose a polynomial algorithm $\mathcal{A}$ for the problem x_{nul} that given two valid transcript $(a_{\mathsf{nul}}, e_{\mathsf{nul}}, z_{\mathsf{nul}})$ and $(a_{\mathsf{nul}}, e'_{\mathsf{nul}}, z'_{\mathsf{nul}})$ where

$$e_{\mathsf{nul}} \neq e'_{\mathsf{nul}} \tag{87}$$

would be able to extract ω_{nul} such that $(x_{\mathsf{nul}}, \omega_{\mathsf{nul}}) \in \mathcal{R}_{\mathsf{nul}}$. Two challenges e_{nul} and e'_{nul}would be

$$\begin{aligned} e_{\mathsf{nul}} &= e_{\mathsf{AND}} + e_{\mathsf{0}} \\ e'_{\mathsf{nul}} &= e'_{\mathsf{AND}} + e'_{\mathsf{0}} \end{aligned} \tag{88}$$

From the two valid transcripts, we have

$$\begin{aligned} a_{01} &= g^{z_0} \cdot \mathsf{C_1}^{-e_0} \\ a_{01} &= g^{z'_0} \cdot \mathsf{C_1}^{-e'_0} \\ \Rightarrow g^{z_0} \cdot \mathsf{C_1}^{-e_0} &= g^{z'_0} \cdot \mathsf{C_1}^{-e'_0} \\ \Rightarrow g^{(z_0 - z'_0)} &= (g^{\alpha})^{(e_0 - e'_0)} \\ \Rightarrow \alpha &= \frac{e_{\mathsf{0}} - e'_{\mathsf{0}}}{z_{\mathsf{0}} - z'_{\mathsf{0}}} \end{aligned} \tag{89}$$

where z_{0} and z'_{0} are given and for the challenges, we have

$$\begin{aligned} e_{\mathsf{0}} &= e_{\mathsf{nul}} - e_{\mathsf{AND}} \\ e'_{\mathsf{0}} &= e'_{\mathsf{nul}} - e'_{\mathsf{AND}} \end{aligned} \tag{90}$$

So, an equation for missing values would be

$$\alpha = \frac{(e_{\mathsf{nul}} - e'_{\mathsf{nul}}) - (e_{\mathsf{AND}} - e'_{\mathsf{AND}})}{z_{\mathsf{0}} - z'_{\mathsf{0}}} \tag{91}$$

Similarly, from the other side of the disjunction, we have

$$\begin{aligned} a_{11} &= g^{z_1} \cdot \mathsf{C_1}^{-e_1} \\ a_{11} &= g^{z'_1} \cdot \mathsf{C_1}^{-e'_1} \\ \Rightarrow g^{z_1} \cdot \mathsf{C_1}^{-e_1} &= g^{z'_1} \cdot \mathsf{C_1}^{-e'_1} \\ \Rightarrow g^{(z_1 - z'_1)} &= (g^{\alpha})^{(e_1 - e'_1)} \\ \Rightarrow \alpha &= \frac{e_{\mathsf{1}} - e'_{\mathsf{1}}}{z_{\mathsf{1}} - z'_{\mathsf{1}}} \end{aligned} \tag{92}$$

By the design, we know that, for a transcript to be valid,

$$e_{\mathsf{AND}} = e_{\mathsf{1}} \tag{93}$$

So the second equation on α would be

$$\alpha = \frac{e_{\mathsf{AND}} - e'_{\mathsf{AND}}}{z_{\mathsf{1}} - z'_{\mathsf{1}}} \tag{94}$$

By Eq.91 and Eq.94 α can be computed by

$$\begin{aligned} \alpha &= \frac{(e_{\mathsf{nul}} - e'_{\mathsf{nul}}) - (\alpha \cdot (z_0 - z'_0))}{z_1 - z'_1} \\ \Rightarrow \alpha &= \frac{e_{\mathsf{nul}} - e'_{\mathsf{nul}}}{z_0 - z'_0 + z_1 - z'_1} \end{aligned} \tag{95}$$

By here, $\mathcal{A}$ has extracted α successfully. To validate the other equations of $\mathsf{Prf}_{\mathsf{Cha}}(0)$ and $\mathsf{Prf}_{\mathsf{Cha}}(1)$ the algorithm checks

$$\begin{aligned} a_{02} &\overset{?}{=} h^{z_0} \cdot \mathsf{C}_1{}^{-e_0}{}_{mod\, p} \\ a_{12} &\overset{?}{=} h^{z_1} \cdot (g^{-1} \cdot \mathsf{C}_2)^{-e_1}{}_{mod\, p} \end{aligned} \tag{96}$$

To extract $\mathsf{flip}_{[\mathsf{v}_i]}$, we use C_2.

$$\begin{aligned} \mathsf{C}_2 &= g^{\mathsf{flip}_{[\mathsf{v}_i]}} \cdot h^{\alpha}{}_{mod\, p} \\ \Rightarrow \mathsf{flip}_{[\mathsf{v}_i]} &= \frac{\mathsf{C}_2}{h^{\alpha}} \end{aligned} \tag{97}$$

Here, $\mathcal{A}$ has extracted $\mathsf{flip}_{[\mathsf{v}_i]}$ too. If $\mathsf{flip}_{[\mathsf{v}_i]} = 0$, it means that the secretkey is not part of ω_{nul} so the design of $\mathcal{A}$ is finished. However to check that, $\mathcal{A}$ iterates through the bulletin board to find a public key pk matching below equations.

$$\begin{aligned} a_{\mathsf{sch}} &= g^{z_{\mathsf{sch}}} \cdot \mathsf{pk}^{-e_{\mathsf{sch}}} \\ a_{\mathsf{sch}} &= g^{z'_{\mathsf{sch}}} \cdot \mathsf{pk}^{-e'_{\mathsf{sch}}} \\ \Rightarrow g^{z_{\mathsf{sch}}} \cdot \mathsf{pk}^{-e_{\mathsf{sch}}} &= g^{z'_{\mathsf{sch}}} \cdot \mathsf{pk}^{-e'_{\mathsf{sch}}} \\ \Rightarrow g^{(z_{\mathsf{sch}} - z'_{\mathsf{sch}})} &= \mathsf{pk}^{(e_{\mathsf{sch}} - e'_{\mathsf{sch}})} \\ \Rightarrow \mathsf{pk} &= g^{(\frac{e_{\mathsf{sch}} - e'_{\mathsf{sch}}}{z_{\mathsf{sch}} - z'_{\mathsf{sch}}})} \end{aligned} \tag{98}$$

In the case of finding the matching pk, sk can be calculated through

$$\mathsf{sk} = \frac{e_{\mathsf{sch}} - e'_{\mathsf{sch}}}{z_{\mathsf{sch}} - z'_{\mathsf{sch}}} \tag{99}$$

At this point $\mathcal{A}$ has extracted all the members of ω_{nul} set. ■

4.2.4 Special Honest Verifier Zero-Knowledge

The idea behind the *special honest Verifier zero knowledge* proof is that an honest Verifier can achieve no information except the intent of sending a valid nullification packet[4]. Note that if a $\mathcal{P}$ can generate a valid nullification packet by using only a valid challenge (with the same probability distribution of normal conversations between the $\mathcal{P}$ and the $\mathcal{V}$[5]), it means that $\mathcal{P}$ is sharing no private information; Thus, the protocol would be Zero-knowledge.

However, proving the Zero-knowledge property would be infeasible for real-sized challenge that is needed for Special Soundness. To get the idea, we show that for a single-bit challenge, the interactive version of Nullification protocol is Zero-knowledge. Then extend the challenge and end up proving special honest Verifier Zero-knowledge property. So sit tight.

In Figure 4, we expanded Verifier concept and utilized a *Random Oracle* $\mathcal{O}_{\mathsf{bit}}$, where by any request a_{nul}, it replies a unique random challenge $e_{\mathsf{bit}} \in \{0, 1\}$ and stores $(a_{\mathsf{nul}}, e_{\mathsf{bit}})$ for the next identical requests. In the figure, the single-bit-challenge Interactive Nullification protocol has been depicted. In this case, for each a_{nul}, $\mathcal{V}$ just honestly forwards the value to the ideal Random Oracle and delivers the response as challenge, e_{bit}.

Imagine the $\mathcal{P}$ wants to guess the challenge e_{bit} ahead of time. Since e_{bit} carries

[4]and of course some side channel information such as computational power that is out of the scope of this study

[5]Because the probability distribution itself could leak some information about the Prover

$\mathcal{P}$		$\mathcal{V}$		$\mathcal{O}_{\mathsf{bit}}$
Computes a_{nul}				
	$\xrightarrow{a_{\mathsf{nul}}}$			
		forwards e_{bit} **honestly** to $\mathcal{O}$		
			$\xrightarrow{a_{\mathsf{nul}}}$	
				Checks if $(a_{\mathsf{nul}}, e_{\mathsf{bit}}) \in \mathsf{dict}_{\mathcal{O}}$
				IF YES
				sends stored e_{bit}
				IF NO
				flips a coin $e_{\mathsf{bit}} \in \{0, 1\}$
				$\mathsf{dict}_{\mathcal{O}} \leftarrow \mathsf{dict}_{\mathcal{O}} \cup \ \{(a_{\mathsf{nul}}, e_{\mathsf{bit}})\}$
				sends stored e_{bit}
			$\xleftarrow{e_{\mathsf{bit}}}$	
		Sends back e_{bit} **honestly** to $\mathcal{P}$		
	$\xleftarrow{e_{\mathsf{bit}}}$			
Computes z_{nul}				
send z_{nul} to $\mathcal{V}$				
	$\xrightarrow{z_{\mathsf{nul}}}$			
		Verifies $(a_{\mathsf{nul}}, e_{\mathsf{bit}}, z_{\mathsf{nul}})$		

Figure 4: Nullification protocol using Single-bit Random Oracle

only single bit of information, the guessed challenge, $e_{\mathcal{P}}$ would be correct half of the time. More formally,

$$\mathsf{Pr}[e_{\mathcal{P}} = e_{\mathsf{bit}}] = 0.5 \tag{100}$$

By guessing e_{bit} ahead of time, $\mathcal{P}$ would be able construct the full nullification packet. For that purpose, $\mathcal{P}$ starting from a random bit e_{bit}, flips a coin to simulate the decision of completing on side of the conjunction. If the coin is head, it generates a random value e_{AND}. Then, it computes e_0

$$e_0 = e_{\mathsf{bit}} - e_{\mathsf{AND}} \tag{101}$$

Otherwise, it generates a random value e_0. Then, it computes e_{AND} by

$$e_{\mathsf{AND}} = e_{\mathsf{bit}} - e_0 \tag{102}$$

Then, by choosing random values $z_0, z_1, z_{\mathsf{sch}} \in Z_{\{2,\, q-2\}}$ and computes a_{nul} in the form of Eq.80 where

$$\begin{aligned} a_{01} &= g^{z_0} \cdot \mathsf{C_1}^{-e_0}{}_{mod\, p} \\ a_{02} &= h^{z_0} \cdot \mathsf{C_2}^{-e_0}{}_{mod\, p} \\ a_{11} &= g^{z_1} \cdot \mathsf{C_1}^{-e_1}{}_{mod\, p} \\ a_{12} &= h^{z_1} \cdot (\mathsf{C_2} \cdot g^{-1})^{-e_1}{}_{mod\, p} \\ a_{\mathsf{sch}} &= g^{z_{\mathsf{sch}}} \cdot \mathsf{pk}^{-e_{\mathsf{sch}}}{}_{mod\, p} \end{aligned} \tag{103}$$

where pk is a random public key from the bulletin board. So, at this point $\mathcal{M}$ has completed a transcript in the form of Eq.79 with the same probability distribution.

By here, we have proven that the single-bit-challenge interactive nullification protocol is zero-knowledge. Note that, by reducing the size of the challenge to a single bit, we have weaken the soundness property. Meaning that a malicious Prover $\mathcal{P}^*$ can fake knowing the witness, ω_{nul} by guessing the challenge with %50 probability. Also, the protocol will not be zero-knowledge with t-bit challenge, where t is sufficiently large, because it would be infeasible to guess the challenge ahead of time and classical rewinding techniques would fail to extract the witness, looking from the view of a malicious Verifier, $\mathcal{V}^*$. Therefore, we relax Zero-knowledge to Honest Verifier Zero Knowledge.

■

4.2.5 Fiat-Shamir Heuristics

To this point, we have proved that the interactive version of Nullification protocol is Σ-protocol. To prove the non-interactive version, we rely on studies of [12], that by the assumption that $\mathcal{H}()$ is an ideal hash function, the constructed proof will have three basic properties of a sigma protocol that has been mentioned in the previous sections. Also in [14], it has been thoroughly explained how to a non-interactive Σ-protocol works in the absence of an ideal random oracle.

4.3 Improvements

In the previous section, a baseline design for Nullification protocol has been described. In this section, we talk about the trade-offs and more complex designs.

4.3.1 Caveats

There were a few caveats in the baseline design that have been enumerated here.

Bounded number of flips per voter

In the baseline aggregation method, we are taking advantage of the additively homomorphic property of the Exponential Elgamal crypto-scheme and using a simple lookup table to solve discrete log problem.

Vulnerable to Pattern voting

Pattern Voting [66] is the act of sending a pattern through voting phases that make the ballot distinguishable. By fingerprinting a ballot, a coercer can check whether a voter followed his instruction or not. This attack has been mentioned, studied and mitigated in other proposed voting systems *e.g.,* [72] and [27]. This attack in our system would be interpreted to The coercer could identify the coerced voter by observing the number of Nullifications for that voter after the tally.

Vulnerable to Denial of Service

Other problem of this system is the vulnerability to denial-of-service attacks. Since, the is no strong authentication mechanism for the nullification phase, a malicious Nullifier or voter can send nullification packet containing $\mathsf{Enc}(0)$ per each voter only using the public information of the crypto-system. So by repeating sending the same packet, the malicious Nullifier can force the election authority (EA) to verify the whole Nullification packet.

Vulnerable to Forced Abstention

We don't solve "forced abstention" which means the coercer makes sure you never vote by checking that your key never shows up. On another case, if a voter isn't going to vote anyways, then they can be bribed and they probably won't nullify (since they weren't going to vote anyways). They just take the money happily. In this case, our designed nullification enables the voter to sell to both parties but if only one candidate is paying for votes, this won't happen.

4.3.2 Secure Nullification

We propose three improvements to mitigate the aforementioned vulnerabilities:

Decentralizing the Election Authority

There are a few practical and well-studied solutions to decentralize the Election Authority's power to avoid malfeasance vulnerabilities.

- **Threshold Cryptoscheme** Since [24], there have been many researches to share the decryption keys in a distributed fashion. In a $(k-n)$-threshold scheme, message (here, nullification bit) is encrypted using a public key, where the corresponding private key is partitioned among the participating parties. The reason is to decentralize the authority to decrypt and pre-mature tally. With a threshold cryptosystem, in order to decrypt an encryption, k out of n parties must participate in the decryption protocol.

- **Self-Tallying systems** Initially proposed by Kiayias and Yung [44], Self-tallying voting systems are those that without needing any central authority to act, it can generate the election results. The idea have been improved by various studies *e.g.,* [33] and [2].

True XOR

A solution for both Pattern attack and bounded flips caveats would be designing an *Exclusive OR* function that could perform over Exponential Elgamal encrypted values. Here, we propose two multi-party functions named *True XOR*, TXOR and

Input #1	Input #2	Output
$\mathsf{Enc}(b_1)$	$\mathsf{Enc}(b_2)$	$\mathsf{Enc}(b_1 \oplus b_2)$
$\mathsf{Enc}(0)$	$\mathsf{Enc}(0)$	$\mathsf{Enc}(0)$
$\mathsf{Enc}(0)$	$\mathsf{Enc}(1)$	$\mathsf{Enc}(1)$
$\mathsf{Enc}(1)$	$\mathsf{Enc}(0)$	$\mathsf{Enc}(1)$
$\mathsf{Enc}(1)$	$\mathsf{Enc}(1)$	$\mathsf{Enc}(0)$

(a)

Input	Output
$\mathsf{Enc}(b_1) + \mathsf{Enc}(b_2)$	$\mathsf{Enc}(b_1 \oplus b_2)$
$\mathsf{Enc}(0)$	$\mathsf{Enc}(0)$
$\mathsf{Enc}(1)$	$\mathsf{Enc}(1)$
$\mathsf{Enc}(2)$	$\mathsf{Enc}(0)$

(b)

Figure 5: Optimizing mix and match XOR. (a) Baseline Mix-n-Match XOR evaluation table. (b) True XOR evaluation table

Online XOR, OXOR that each has its own advantages. Both functions have been inspired by *Mix and Match* protocol by Juels et al.'s [39]. *Mix and Match* protocol constructs and executes an arbitrary function over encrypted values by multi-party shuffling and re-randomization techniques.

Note that we do not need the exact value of $\mathsf{nul}_{\mathsf{total}}$ at the end and only the parity of it would be enough to encode whether to flip the vote or not. Based on this observation, we design various XOR implementation over the arriving Nullification bit $\mathsf{bit}_{\mathsf{nul}}$ and current aggregated bit, $\mathsf{bit}_{\mathsf{agg}}$. Figure 5 illustrates two different implementations of XOR function. On the left side, the evaluation table of the baseline XOR function has been shown and on the right side, using the additively homo-morphic properties of Exponential Elgamal cryptosystem, we proposed a new function with the same functionality, named True XOR.

Figure 5 shows a baseline XOR function over encrypted values $\mathsf{Enc}(b_1)$ and $\mathsf{Enc}(b_2)$ where both are 0 or 1. As it has been mentioned, any function could be implemented to work under encryption using Mix-n-match technique. Therefore this baseline XOR could be a Using the additively homo-morphic properties of Exponential Elgamal, we can convert the function to TXOR that can be see in part (b). Later, we will see that it reduce the number of needed *Plaintext Equality Test*, PET, that has been described in Section 2.3.4, in the matching phase.

We define *True XOR* function, TXOR() in the form of

$$\mathsf{TXOR} : \mathbb{C}^2_{\{0,1\}} \to \mathbb{C}_{\{0,1\}} \tag{104}$$

where $\mathbb{C}_{\{0,1\}}$ is the set of all chipertexts in the predefined Exponential Elgamal cryptosystem, $\langle p,\ q,\ g,\ h\rangle$ that is an encryption of 0 or 1. Formally,

$$\mathbb{C}_{\{0,1\}} : \{\langle \mathsf{C}_1, \mathsf{C}_2\rangle \,|\, \forall\, r \in Z_{\{2,\, q-2\}}, m \in \{0,1\},\ \langle \mathsf{C}_1,\ \mathsf{C}_2\rangle = \mathsf{Enc}(m,r)\} \tag{105}$$

where $\mathsf{Enc}(m,r)$ is the Exponential Elgamal encryption function. In Figure 5, in the left side, the evaluation table of TXOR function has been illustrated.

We define TXOR's functionality as an exclusive OR over two ciphertext and outputs a new well-formed ciphertext. Formally,

$$\begin{aligned} \mathsf{TXOR}[\mathsf{Enc}(m_1,r_1),\mathsf{Enc}(m_2,r_2)] &= \mathsf{Enc}(m_\mathsf{X}, r_\mathsf{X}) \\ &= \langle \mathsf{C}_{\mathsf{X}1}, \mathsf{C}_{\mathsf{X}2}\rangle \\ &= \langle g^{r_X}{}_{mod\, p},\ g^{(m_1 \oplus\, m_2)} \cdot h^{r_X}{}_{mod\, p}\rangle \end{aligned} \tag{106}$$

where XOR is binary exclusive OR operator. Note the TXOR function's input and output has been designed to be used at the arrival of every new nullification packet.

To participate in Mix-and-match protocol, trustees that have read/write access to a shared memory, follow two steps

- **Mixing**. In Figure 6, the mixing phase of *Mix and Match* protocol has been illustrated. Each trustee, reads the current version of the function's evaluation table, shuffle rows are re-randomize input and output encryptions. We define

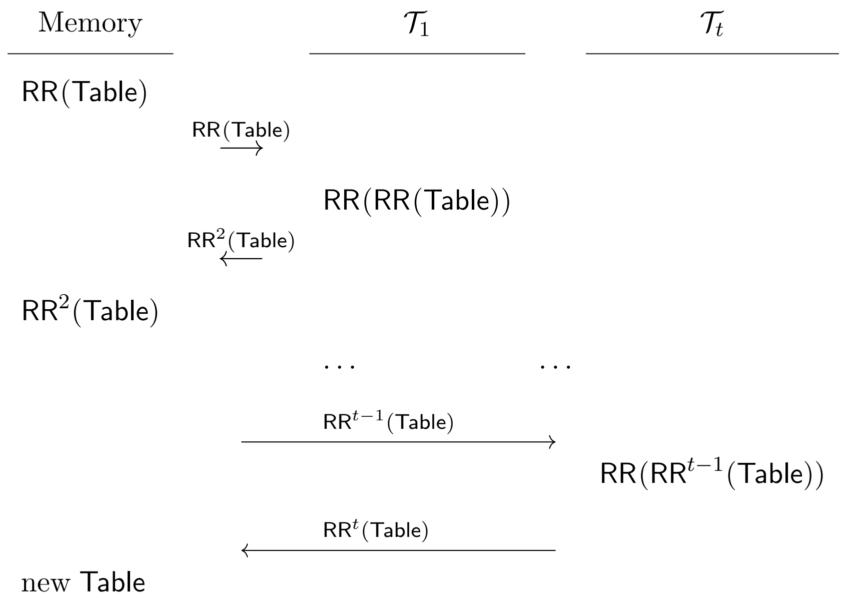

Figure 6: Mixing phase of Mix and Match protocol

RR() function as shuffling of row of the TXOR function's evaluation table and also re-randomization of encrypted values inside it.

- **Matching**. In this phase, trustees perform PET test per each Input cell of the evaluation table. In Figure 5, the input cells have been annotated with purple color. Per each row, trustees run two PETs and if for a row, both PETs returns true, the output of of the TrueXor would be the output of that row. In practice, the mixed evaluation table would be pre-computed before the election and the only online computation needed for this design on TXOR is executing three multi-party PETs.

Online XOR

Another proposed implementation of XOR over encryption is *Online XOR*, OXOR, where it would be less expensive but without pre-computation. The key idea is to

$\mathsf{Enc}(0)$	$\mathsf{Enc}(0)$
$\mathsf{Enc}(1)$	$\mathsf{Enc}(1)$
$\mathsf{Enc}(2)$	$\mathsf{Enc}(0)$

(a)

$$\xrightarrow[\boxed{\text{Mixing phase}}]{}$$

$\mathsf{RR}^t(\mathsf{Enc}(1))$	$\mathsf{RR}^t(\mathsf{Enc}(1))$
$\mathsf{RR}^t(\mathsf{Enc}(0))$	$\mathsf{RR}^t(\mathsf{Enc}(0))$
$\mathsf{RR}^t(\mathsf{Enc}(2))$	$\mathsf{RR}^t(\mathsf{Enc}(0))$

(b)

Figure 7: True XOR. (a) initial-state (b) after a sample mixing

Input	Output
$\mathsf{Enc}(b_{\mathsf{agg}}) + \mathsf{Enc}(b_{\mathsf{nul}})$	$\mathsf{Enc}(b_{\mathsf{agg}} \oplus b_{\mathsf{nul}})$
$\mathsf{Enc}(b_{\mathsf{agg}}) + \mathsf{Enc}(b_{\mathsf{agg}})$	$\mathsf{Enc}(0)$
$\mathsf{Enc}(1)$	$\mathsf{Enc}(1)$

Figure 8: The inner structure of Online XOR function

use the entropy of the input random variables inside the evaluation table. OXOR's domain, range and functionality would be identical to TXOR that have been explained in Eq.105 and Eq.106.

In Figure 8, the design of OXOR has been depicted. The idea was to detect that $\mathsf{Enc}(b_{\mathsf{agg}})+\mathsf{Enc}(b_{\mathsf{agg}})$ is encoding $\mathsf{Enc}(b_{\mathsf{agg}})+\mathsf{Enc}(b_{\mathsf{nul}})$ when $\mathsf{Enc}(b_{\mathsf{agg}}) = \mathsf{Enc}(b_{\mathsf{nul}})$. The advantage of OXOR over TXOR is that it is computationally cheaper (2 PET worst case, 1 PET expectation). The only drawback is the trustees, should participate in the mixing phase after each arrival of new nullification packet; Instead of pre-cumputing the evaluation table before election. Because the OXOR's evaluation table includes $\mathsf{Enc}(b_{\mathsf{agg}}) + \mathsf{Enc}(b_{\mathsf{agg}})$ that depends on the on-going state of the nullification board.

Now that we reduced the problem to be solved with two-row evaluation table *Mix and Match*, we reduced the mixing phase by Zero-knowledge proof of shuffles using previously explained circuit construction using Σ-protocols.

A generic binary proof-of-shuffle has been shown in Figure 9. The perform OXOR's

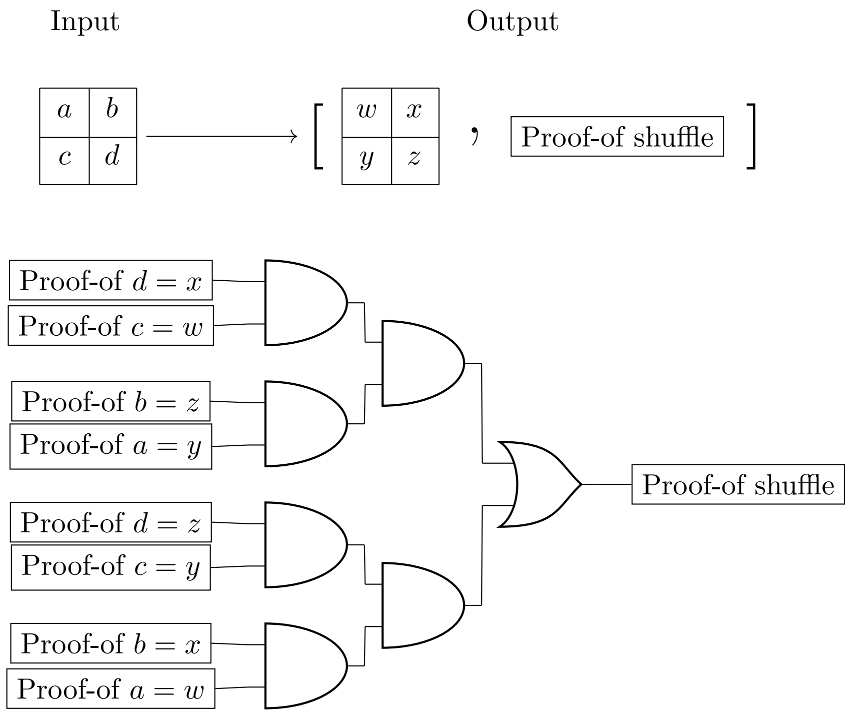

Figure 9: Proof of shuffle of Online XOR evaluation table

mixing phase, each trustee given the online version of the OXOR evaluation table (Figure 8) as $[a, b, c, d]$, computes the proof-of-shuffle receipt that contains proof of disjunction of proofs of all the possible permutations [6] and the rerandomized evaluation table as $[w, x, y, z]$. Formally

$$\begin{array}{cc} \text{Mixing input} & \text{Mixing output} \\ \begin{bmatrix} a = 2 \cdot \mathsf{Enc}(b_{\mathsf{agg}}) & b = \mathsf{Enc}(0) \\ \\ c = \mathsf{Enc}(1) & d = \mathsf{Enc}(1) \end{bmatrix} & \begin{bmatrix} w = \mathsf{RR}(2 \cdot \mathsf{Enc}(b_{\mathsf{agg}})) & x = \mathsf{RR}(\mathsf{Enc}(0)) \\ \\ y = \mathsf{RR}(\mathsf{Enc}(1)) & z = \mathsf{RR}(\mathsf{Enc}(1)) \end{bmatrix} \end{array} \tag{107}$$

The matching phase would be exactly the same as the classic Mix and Match protocol. Trustees perform PET tests (2 worst case, 1 expectation) on the rerandomized evaluation table's inputs. We reduced a PET with the expense of losing the advantage of Re-randomized evaluation table pre-computation.

[6]Note that the number of permutations for n objects is $n!$, so that this technique will be drastically less efficient for larger evaluation tables.

4.3.3 Efficient Nullification

We have mentioned that TXOR and OXOR technique in the previous sections that would lead to more efficient nullification design. In this section we will compare the efficiency of various extensions of the protocol. larger concurrent work is better than smaller one-shot calculation at the end (a comparison is needed).

Efficient Lookup table using Pollard's Rho

As it has been shown in Section 4.3.1, the baseline aggregation design misses an efficient way to get $\mathsf{flip}_{[\mathsf{total},v_i]}$ from $g^{\mathsf{flip}_{[\mathsf{total},v_i]}}$ where i is the index of an arbitrary voter in Nullification Bulletin board table, $\mathsf{BB}_{\mathsf{nul}}$. Here we discuss some solutions with different characteristics that can be used in different scenarios. **Pollard's RHO** method [63] is a randomized algorithm for computing the discrete logarithm. It generates a pseudo-random sequence by an iteration function . It is a way to pseudo-random walk in p space looking for a collision to break the a Discrete Log problem from that intersection. The expected number of evaluations before a match is $\pi|G_p|/2$, where G_p is the finite syclic group over p. This approach fully exponential in the problem size and would be more efficient than the naive look-up table design.

Concurrent Nullification Verification

In the design of Cobra [28] partitioning work over time, linear amount of work on each entry, concurrent ballot authorization. (COBRA) [23]

Anonymity sets

Anonymity is being defined in sets, where the actor of an action could not identified by efficient algorithms. However, in our case, since the total number of voters would be relatively large, smaller subsets could be used to offer sufficient but faster anonymity.

Clark et al. in [20] uses the concept of anonymity sets to speed up the designed e-voting system, Selections. For that purpose, Nullifier would select a set of n_{set} voters' public keys and construct the nullification proofs for that set, where n_{set} would be predefined in the $\mathsf{SignedElectionConfiguration}$.

Chapter 5

Wrap-up

"*The world can be such an unsparingly savage place. One can be forgiven for believing that evil will triumph in the end.*" - Red Reddington

In this study we tried to take one step further to solve insecure e-voting systems problem.

5.1 Discussions

5.1.1 Generalization

The idea of Nullification can get generalized through:

Compatibility

A key feature of the nullification approach is to separate the mechanism for mitigating the influence of from the main voting mechanisms namely, casting and counting the ballots. Consequently, our approach can be applied to a wide variety of voting systems, including precinct voting with paper ballots, voting by mail, and Internet voting; for instance, the Helios internet voting system [1] and Scantegrity [13] mail voting system.

K-Candidate Election

Based on Nullification system, each voter has an option to flip their vote using a "*flip code*," which they establish during registration. In a two-candidate race, flipping means computing the modulo-two sum of the original vote and any flipped votes for that ballot question. The idea can be generalized to a modulo-k sum for a k-candidate race.

Desired Actions

Nullification can be interpreted into two parts: signalling and the desired action. After the signal has been received privately to the Bulletin board, then it can be interpreted into three different actions[1]:

- **Flip**. In this case, vote gets flipped to other party if an odd number of nullifying nullification packets for a particular voter gets to $\mathsf{BB}_{\mathsf{nul}}$.
- **Cancel-toggle**. Vote gets cancelled if an odd number of nullification packets on $\mathsf{BB}_{\mathsf{nul}}$ nullifies the targeted voter's vote.
- **Cancel-nil**. In this case, vote gets cancelled if one or more entities cancel it, meaning that only one of the nullification packets should include $\mathsf{Enc}(1)$ for the targeted voter.

[1]The model comes from voting procedure. Casting a ballot can be assumed as signalling and how to count it can be the action.

5.1.2 Nullification or Voting

The difference between the nullification-only voting[2] and the proposed VoteXX protocol would be on efficiency. Nullification is so expensive, comparing to just signing the ballot. On the other hand, the coercion (hopefully) will not happen at scale. So just the opportunity to nullify would be sufficient to guard the voting system against tally-altering coercion situations. Therefore we have chosen a simpler and faster mechanism for voting.

5.1.3 Nullification, not Revoting.

Nullification is different from multiple voting. In multiple voting, only the voter can vote, and the coercer might be able to coerce the voter at the end of the voting period. By contrast, in nullification, there might be one or more hedgehogs, and the coercer does not know who they are or how many there are.

5.1.4 Nullification and Liquid Democracy

Nullification idea in practice would get into Liquid Democracy. Consider a binary voting situation that would decide for a two-party presidential election. We call the parties Red and Blue. Assume that there is an activist, Alice, that is campaigning for Red party. Putting the nullification protocol under scrutiny, Alice finds out that through this protocol, she can nullify all the votes of people who trust her in just one single nullification packet. So she starts advertising her contact for the other Red party supporters to become a mass Blue-vote-Nullifier. Red Supporter would share their Blue key with Alice. It can work as a proof of support to party's candidate.

One of the main objectives of the Liquid Democracy concept is to mitigate the

[2]meaning that the voting itself would be sent as a Nullification packet.

Voter Fatigue problem. The problem can be addressed by Nullification too for repetitive elections *e.g.,* internal party voting. As long as Alice plays fair and acts in the desired manner, party members continue trusting her and delegating the vote to her. We can consider a voting system that works without voting and only by the nullification.

5.1.5 Game Theoretic Analysis

The coercer can demand that the voter reveal all of the secret keys the voter possesses. However, revealing the secrets to the coercer would not prevent the voter from flipping their vote again. So by the existence of a coercion-free window, voter would be able to send a signal to another trusted Nullifier to flip her vote on behalf of her. The signal could be subtle and covert—such as moving a specified potted plant on a balcony or posting a picture of a specific object on Twitter. As we can see, the multiple Nullifier strategy are useful when a coercer closely monitors the voter. We assume that, at some time after registration and before the nullification period, the voter is able to communicate their flip code to their Nullifier(s) privately. We also assume that, at some time after any coercion, and before the nullification period, the voter is able to signal the hedgehog(s) privately.

To clarify the idea, Consider an election between Alice and Bob. The coercer instructs the voter: "cast a vote for Bob in my presence. After the electron, I will flip a coin. If it is heads, I will nullify your vote. If it is tails, I will do nothing."

From the voter's perspective, say they are an Alice supporter. If the voter does nullify, there is a 50% chance the coercer will do nothing and the vote will be nullified. There is a 50% chance the coercer will also nullify, which cancels out the nullification and the vote is cast for Bob. Now assume the voter doesn't nullify. There is a 50% chance the coercer will nullify which results in the vote being nullified. There is a

50% chance they do nothing which results in a vote for Bob. The probabilities are identical if the voter nullifies or not, so the voter is indifferent to nullifying. From the coercer's perspective, if they coerce 1000 voters, 500 will be nullified and 500 will be cast for Bob. So this does not stop coercion, it just doubles the price for the coercer.

5.2 Future Works

This study can evolve through

5.2.1 Improving the Design and Implementation

The efficiency of the parallel nullification receipts that would be sent all together by a single *Nullifier* can be optimized by theory. A thread of future works can be diving deeper into the ocean of Σ-protocols.

There is also an ongoing effort to fully implement the VoteXX voting protocol over XX network. The baseline implementation of Nullification has been successfully added. Future works on this direction would be implementing TXOR and OXOR functions. Also, holding a real election is on the plan.

5.2.2 Compulsory Lubrication

As it has been described in the previous chapter, the designed system would end up into a Liquid Democracy style game. So studying the game theoretic consequences is a must. This study mapped the problem of coercion to a more fair vote buying game but have not touched further. There is an open door for contributions to improve the game itself.

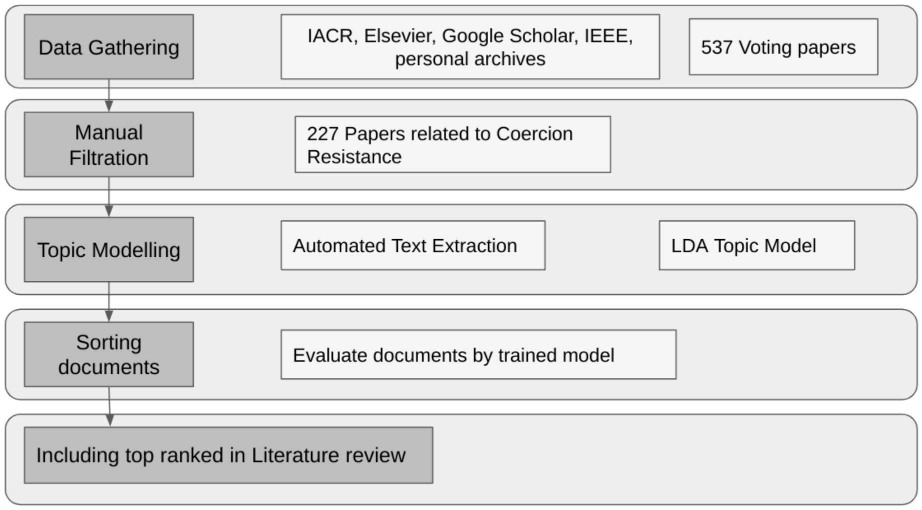
Data Gathering
IACR, Elsevier, Google Scholar, IEEE, personal archives
537 Voting papers
Manual Filtration
227 Papers related to Coercion Resistance
Topic Modelling
Automated Text Extraction
LDA Topic Model
Sorting documents
Evaluate documents by trained model
Including top ranked in Literature review

Figure 10: The process of Topic Extraction over academic e-voting papers

5.2.3 Literature Disambiguation

The literature ambiguition is not limited to coercion resistance or even voting problem. Many terms has been used for the same concepts. Based on the initial motivation of the author (Section 1.2), we tried to disambiguate the literature using Information Extraction and Natural Language Analysis techniques. In Figure 10, the steps to extract the most influential topics on the modern voting literature.

We started with 537 voting related academic papers using publicly available search engines and personal repositories of voting domain experts [3]. Figure 11 depicts four found topics and their corresponding word cloud. We hypothesize that Having access to a list of previously used terms would help the new academic writer to stick to the literature. So, it would lead to disambiguation of the whole literature.

[3]The search was not exhaustive and there might be missing papers that presenting voting requirements and protocols. The accuracy and fairness of this extraction must be verified in future works. We used keywords such as “vote”, “voting”, “ballot” and an expert read the title and the abstract to see whether it is roughly related to e-voting secure system design or not.

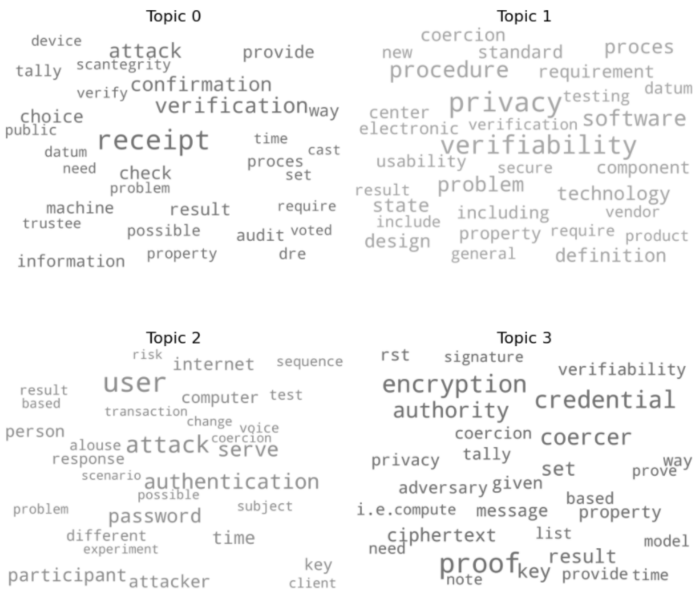

Figure 11: WordCloud of the topics extracted out of 537 voting paper, using LDA over manually curated repository

Bibliography

[1] B. Adida. Helios: Web-based Open-Audit voting. In *17th USENIX Security Symposium (USENIX Security 08)*, San Jose, CA, July 2008. USENIX Association.

[2] M. Arapinis, N. Lamprou, L. Mareková, and T. Zacharias. E-cclesia: Universally composable self-tallying elections. Cryptology ePrint Archive, Report 2020/513, 2020. `https://ia.cr/2020/513`.

[3] R. Araujo, S. Foulle, and J. Traoré. A practical and secure coercion-resistant scheme for internet voting. *Toward Trustworthy Elections*, LNCS 6000, 2010.

[4] R. Araujo, N. B. Rajeb, R. Robbana, J. Traoré, and S. Yousfi. Towards practical and secure coercion-resistant electronic elections. In *CANS*, 2010.

[5] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *ACM STOC*, 1994.

[6] M. Bishop. *Computer security : art and science.* Addison-Wesley Pearson, Boston, 2019.

[7] M. Bodner. Videos online show blatant ballot-stuffing in russia, 2018.

[8] D. Boneh. The decision diffie-hellman problem. In J. P. Buhler, editor, *Algorithmic Number Theory*, pages 48–63, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[9] J. Bonneau, S. Preibusch, and R. Anderson. A birthday present every eleven wallets? the security of customer-chosen banking pins. In *Financial Cryptography*, 2012.

[10] M. Bratton. Vote buying and violence in nigerian election campaigns. *Electoral Studies*, 27(4):621–632, 2008.

[11] T. Bronack. The problems with a paper based voting system. 2000.

[12] R. Canetti, Y. Chen, J. Holmgren, A. Lombardi, G. N. Rothblum, R. D. Rothblum, and D. Wichs. Fiat-shamir: From practice to theory. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC

2019, page 1082–1090, New York, NY, USA, 2019. Association for Computing Machinery.

[13] R. T. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P. S. Hernson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, A. T. Sherman, and P. L. Vora. Scantegrity II municipal election at Takoma Park: the first E2E binding governmental election with ballot privacy. In *USENIX Security Symposium*, 2010.

[14] P. Chaidos and J. Groth. Making sigma-protocols non-interactive without random oracles. In *Public Key Cryptography*, 2015.

[15] D. Chaum. Surevote: Technical overview. In *WOTE*, 2001.

[16] D. Chaum. Random-sample voting, 2016.

[17] D. Chaum, R. Carback, M. Yaksetig, J. Clark, M. Nejadgholi, A. T. Sherman, C. Liu, and F. Zagórski. Votexx: A solution to improper influence in voter-verifiable elections. In *7th International Joint Conference, E-Vote-ID 2022*, Lecture Notes in Computer Science. Springer, 2022.

[18] D. Chaum, D. Das, F. Javani, A. Kate, A. Krasnova, J. de Ruiter, and A. T. Sherman. cmix: Mixing with minimal real-time asymmetric cryptographic operations. In *ACNS*, 2017.

[19] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO*, 1992.

[20] J. Clark and U. Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. In *FC*, 2011.

[21] M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368, 2008.

[22] S. Coll. *Directorate S: The C.I.A. and America's Secret Wars in Afghanistan and Pakistan*. Penguin Random House, 2018.

[23] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.

[24] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '94, page 522–533, New York, NY, USA, 1994. Association for Computing Machinery.

[25] W. Diffie and M. E. Hellman. New directions in cryptography, 1976.

[26] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[27] A. Essex, J. Clark, and C. Adams. *Aperio: High Integrity Elections for Developing Countries*, page 388–401. Springer-Verlag, Berlin, Heidelberg, 2010.

[28] A. Essex, J. Clark, and U. Hengartner. Cobra: Toward concurrent ballot authorization for internet voting. In *EVT/WOTE*, 2012.

[29] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[30] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989.

[31] D. Graeber. *Debt : the first 5,000 years*. Brooklyn, N.Y. : Melville House, 2011.

[32] T. Haines and B. Smyth. Surveying definitions of coercion resistance. Cryptology ePrint Archive, Report 2019/822, 2019. `https://ia.cr/2019/822`.

[33] F. Hao and P. R. P. Zieliński. Anonymous voting by two-round public discussion. *IET Information Security*, 4:62–67(5), June 2010.

[34] C. Hazay and Y. Lindell. *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media, 2010.

[35] S. Heiberg, H. Lipmaa, and F. v. Laenen. On e-vote integrity in the case of malicious voter computers. In *ESORICS*, 2010.

[36] J. Helbach, J. Schwenk, and S. Schage. Code voting with linkable group signatures. In *EVOTE*, 2008.

[37] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'00, page 539–556, Berlin, Heidelberg, 2000. Springer-Verlag.

[38] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT*, 2000.

[39] M. Jakobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In *Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '00, page 162–177, Berlin, Heidelberg, 2000. Springer-Verlag.

[40] R. Joaquim, C. Ribeiro, and P. Ferreira. Improving remote voting security with CodeVoting. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*. Springer, 2010.

[41] D. W. Jones. problems with voting systems and the applicable standards. 2001.

[42] A. Juels, D. Catalano, and M. Jacobsson. Coercion-resistant electronic elections. In *ACM WPES*, 2005.

[43] A. Juels, D. Catalano, and M. Jakobsson. *Coercion-Resistant Electronic Elections*, page 37–63. Springer-Verlag, Berlin, Heidelberg, 2010.

[44] A. Kiayias and M. Yung. Self-tallying elections and perfect ballot secrecy. In *Public Key Cryptography*, 2002.

[45] R. Koenig, R. Haenni, and S. Fischli. Preventing board flooding attacks in coercion-resistant electronic voting schemes. In *SEC*, 2011.

[46] A. Kolbe, N. Cesnales, M. Puccio, and R. Muggah. Impact of perceived electoral fraud on haitian voter's beliefs about democracy. 2015.

[47] E. Kramon. Where is vote buying effective? evidence from a list experiment in kenya. *Electoral Studies*, 44:397–408, 2016.

[48] R. Kusters, T. Truderung, and A. Vogt. Accountability: Definition and relationship to verifiability. In *ACM CCS*, 2010.

[49] W. Lueks, I. Querejeta-Azurmendi, and C. Troncoso. Voteagain: A scalable coercion-resistant voting system. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[50] T. Moran and M. Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *CRYPTO*, 2006.

[51] A. A. Nasar. The history of algorithmic complexity. *The Mathematics Enthusiast*, 13(3):217–242, 2016.

[52] M. Nejadgholi and J. Yang. A study of oracle approximations in testing deep learning libraries. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pages 785–796. IEEE, 2019.

[53] M. Nejadgholi, N. Yang, and J. Clark. Short paper: Ballot secrecy for liquid democracy. In M. Bernhard, A. Bracciali, L. Gudgeon, T. Haines, A. Klages-Mundt, S. Matsuo, D. Perez, M. Sala, and S. Werner, editors, *Financial Cryptography and Data Security. FC 2021 International Workshops - CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers*, volume 12676 of *Lecture Notes in Computer Science*, pages 306–314. Springer, 2021.

[54] NIST. NIST election terminology glossary - draft. Accessed: 2020-07-05.

[55] N. Nyagudi. *Election Shenanigans - Kenyan Hybrid Warfare.* 07 2020.

[56] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO*, 1992.

[57] T. Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Workshop on Security Protocols*, 1997.

[58] R. Oppliger, J. Schwenk, and C. Lohr. Captcha-based code voting. In *EVOTE*, 2008.

[59] S. Park, M. A. Specter, N. Narula, and R. L. Rivest. Going from bad to worse: from internet voting to blockchain voting. *J. Cybersecur.*, 7, 2021.

[60] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[61] T. P. Pedersen. A threshold cryptosystem without a trusted party. In *EUROCRYPT*, 1991.

[62] J. Peralta and S. Gelera. 'massive' vote-buying continues on election day, 2019.

[63] J. M. Pollard. Monte carlo methods for index computation (). *Mathematics of Computation*, 32:918–924, 1978.

[64] S. Popoveniuc. Speakup: remote unsupervised voting. In *ACNS*, 2010.

[65] S. Popoveniuc, J. Kelsey, A. Regenscheid, and P. Vora. Performance requirements for end-to-end verifiable elections. 2010 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE '10), Washington, DC, US, 2010-08-09 00:08:00 2010.

[66] S. Popoveniuc and J. Stanton. Undervote and pattern voting: Vulnerability and a mitigation technique. In *In Preproceedings of the 2007 IAVoSS Workshop on Trustworthy Elections (WOTE 2007*, 2007.

[67] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Cornell University, 1962.

[68] A. D. Rubin, D. S. Wallach, D. Boneh, M. D. Byrne, D. Dean, D. L. Dill, P. G. Neumann, D. K. Mulligan, and D. A. Wagner. A center for correct , usable , reliable , auditable , and transparent elections (accurate). 2005.

[69] P. Y. A. Ryan and V. Teague. Pretty good democracy. In *Workshop on Security Protocols*, 2009.

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

[71] C. P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 239–252, New York, NY, 1990. Springer New York.

[72] W. D. Smith. New cryptographic election protocol with best-known theoretical properties. 2005.

[73] O. Spycher, R. Haenni, and E. Dubuis. Coercion-resistant hybrid voting systems. In *EVOTE*, 2010.

[74] S. C. Stokes. Perverse accountability: A formal model of machine politics with evidence from argentina. 2005.

[75] J. Surowiecki. *The Wisdom of Crowds*. Anchor, 2005.

[76] M. Vasek, J. Bonneau, R. Castellucci, C. Keith, and T. Moore. The bitcoin brain drain: Examining the use and abuse of bitcoin brain wallets. In *Financial Cryptography*, 2017.

[77] M. Volkamer and R. Grimm. Multiple casts in online voting: Analyzing chances. In *EVOTE*, 2006.

[78] F. Zagórski, R. Carback, D. Chaum, J. Clark, A. Essex, and P. L. Vora. Remotegrity: Design and use of an end-to-end verifiable remote voting system. In *ACNS*, 2013.