Process for Designing and Implementing Provably Verifiable Voting Systems

Kholud Alghamdi

Follow this and additional works at: https://repository.fit.edu/etd

Part of the Computer Sciences Commons
The author grants permission to make single copies.
We the undersigned committee hereby recommend that the attached
document be accepted as fulfilling in part the requirements for the degree of
Doctor of Philosophy in Computer Science.

“Process for Designing and Implementing Provably Verifiable Voting Systems”
a dissertation by Kholud Alghamdi

Marius C. Silaghi, Ph.D.
Professor, Department of Computer Engineering and Sciences
Major Advisor

William Allen, Ph.D.
Associate Professor, Computer Science
Committee Member

Michael King, Ph.D.
Associate Professor, Computer Science
Committee Member

Veton Kepuska, Ph.D.
Associate Professor, Computer Engineering
Committee Member

Philip Bernhard, Ph.D.
Associate Professor and Department Head
Department of Computer Engineering and Sciences
ABSTRACT

Process for Designing and Implementing Provably Verifiable Voting Systems

by

Kholud Alghamdi

Thesis Advisor: Marius C. Silaghi, Ph.D.

This research aims to explore processes for designing verifiable voting systems in which certain properties can be proven, exemplified with systems applicable to election processes in Saudi Arabia. The electronic government model has become a substantial channel for governments to connect to businesses and citizens, to develop services, and provide general information. E-voting and in particular online voting is one of the important tools that can be used in political and administrative places where information and communications technology devices and tools are utilized to simplify people’s lives and facilitate the election process and decision making. Election processes let a population reveal their preferences regarding how they would like business to be conducted and help provide feedback to the upper administration. As might be expected, the integrity of the election procedures is the backbone of modern state management, so the voting system should be adequately strong in order to resist a variation of deceitful conduct. Moreover, the system should also be appropriately
explicit and understandable such that voters and candidates agree on the outcomes of the election. The integrity and confidentiality of this type of system will not be preserved unless restricted measures on security and authentication are enforced.
Table of Contents

Abstract iii

List of Figures viii

List of Symbols ix

Acknowledgements x

Chapter 1 Introduction 1

1.1 Motivation ........................................... 3

1.2 Significance of the Study .......................... 6

1.3 Contribution ........................................... 7

1.4 Dissertation Structure ............................. 8

Chapter 2 Background and Related Work 10

2.1 Blockchain Technology ............................. 11

2.1.1 Concepts in Blockchain Technology ............ 14

2.1.2 Proof-of-Work Versus Proof-of-Stake ............ 17
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>Benaloh Election Protocol</td>
<td>19</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Homomorphic Encryption</td>
<td>21</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Voting Process</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>Merritt’s Election Protocol</td>
<td>23</td>
</tr>
<tr>
<td>2.4</td>
<td>Verifiability Mechanisms</td>
<td>25</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Analyzing Implementations of Election Technologies</td>
<td>25</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Epistemic Policies for Voting Systems</td>
<td>32</td>
</tr>
<tr>
<td>2.5</td>
<td>Official Saudi Authentication Systems</td>
<td>36</td>
</tr>
<tr>
<td>2.5.1</td>
<td>ABSHER</td>
<td>37</td>
</tr>
<tr>
<td>2.5.2</td>
<td>NAFAD</td>
<td>39</td>
</tr>
<tr>
<td>2.6</td>
<td>Formal Verification</td>
<td>42</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Model Checking</td>
<td>43</td>
</tr>
<tr>
<td>2.7</td>
<td>Voting System Properties</td>
<td>46</td>
</tr>
</tbody>
</table>

**Chapter 3  Concepts and Problems**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Contribution Concepts</td>
<td>50</td>
</tr>
<tr>
<td>3.2</td>
<td>Research Problem</td>
<td>56</td>
</tr>
<tr>
<td>3.3</td>
<td>The Proposed Approach</td>
<td>57</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Designing for Verifiability</td>
<td>58</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Experiment Overview</td>
<td>60</td>
</tr>
<tr>
<td>3.4</td>
<td>System Architecture</td>
<td>61</td>
</tr>
<tr>
<td>3.5</td>
<td>Attacker Model</td>
<td>64</td>
</tr>
</tbody>
</table>
Chapter 4  Analysis and Results  66

4.1  Steps in Design for Verifiability  66

4.1.1  LTL Models of Voting Processes Properties  67

4.1.2  Experimental Validation of the Design Model Process Step  69

4.1.3  Formal Proof and Analysis of the Properties  74

4.2  Implications of Required Limitations  76

4.2.1  Models of Voting Schemes  77

4.2.2  Formal Proof and Analysis  78

Chapter 5  Conclusions  80

Appendix A  Experiments Details and Code  89

Appendix B  Call Graphs  100

References  103
List of Figures

2.1 Blockchain Content .................................................. 12
2.2 Blockchain Schema ..................................................... 12
2.3 NAFAD Login Authentication .......................................... 41
2.4 NAFAD IAM Logout Authentication .................................... 41
2.5 NAFAD Service Provider Logout Authentication ......................... 42

3.1 Sample Property Logic Representation with Büchi Automata ............. 54
3.2 Data Path During Ballot Casting ...................................... 61

4.1 First Automata ......................................................... 69
4.2 Second Automata ....................................................... 70
4.3 Third Automata ......................................................... 70
4.4 Call Graph .............................................................. 74

B.1 First (LTL) Formula Related Functions .................................. 100
B.2 Second (LTL) Formula Related Functions ............................. 101
B.3 Third (LTL) Formula Related Functions ................................ 102
## List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NGO</td>
<td>Non-Governmental Organization</td>
</tr>
<tr>
<td>VCVS</td>
<td>Verifiable Casting Voting System</td>
</tr>
<tr>
<td>IAM</td>
<td>Identity Authentication Management</td>
</tr>
<tr>
<td>LTL</td>
<td>Linear Temporal Logic</td>
</tr>
<tr>
<td>MD</td>
<td>Message Digest</td>
</tr>
<tr>
<td>SHA</td>
<td>Secure Hash Function</td>
</tr>
<tr>
<td>RIPEMD</td>
<td>Race Integrity Primitives Evaluation Message Digest</td>
</tr>
<tr>
<td>PLTL</td>
<td>Past Linear Temporal Logic</td>
</tr>
<tr>
<td>FLTL</td>
<td>Future Linear Temporal Logic</td>
</tr>
<tr>
<td>BA</td>
<td>Büchi Automata</td>
</tr>
<tr>
<td>EFOTL</td>
<td>Epistemic first-order temporal logic</td>
</tr>
<tr>
<td>PROMELA</td>
<td>Process Meta Language</td>
</tr>
<tr>
<td>ICT</td>
<td>Information and Communications Technology</td>
</tr>
</tbody>
</table>
Acknowledgements

I would like to first thank His Excellency Eng. Abdullah Alsawaha, the minister of Saudi Ministry of Communications and Information Technology (MCIT), for his inspiration and support when I met His Excellency at his office before I started my PhD studies, he encouraged me to take this step and to preserve my job at the same time. As His Excellency said, "we do not want to lose someone like you. You can do it!". His interest in my research area and his words were enough to inspire and push me forward during my studies.

I am indebted and thankful to Dr. Marius Silaghi, my major advisor, for his endless support, guidance, time, and extreme patience. Also, for his confidence in my abilities. He always welcomed me when I had any problems or questions about my writing or research. He was always appreciating my research strengths, patiently encouraging me to improve in some weak areas, supporting my own ideas and research directions. Without his appreciation and encouragement in my research area, I may not have ever pursued this PhD degree.

I am grateful to Dr. Philip Bernhard, Head of Department of Computer Engineering and Sciences, and to my respected committee members Dr. William Allen, Dr. Michael King, and Dr. Veton Kepuska for their valuable feedback and precious time.
I extend sincere gratitude to the cybersecurity general department team in the Saudi Ministry of Communications and Information Technology (MCIT) for their encouragement and kind wishes, especially Eng. Almuthna Aloqla, the general manager, for his continuous support, encouragement, and advice.

Finally, I am grateful and thankful to the Saudi Arabia government, represented by the Ministry of communications and Information technology, for their financial support.
Chapter 1

Introduction

Based on the importance of voting processes, traditional voting systems have evolved to become electronic. Electronic voting, which is a digital process whereby voters submit their ballots electronically instead of on paper, involves a wide range of systems, frameworks, and techniques. The electronic voting context is classified into two major types of environments, controlled and uncontrolled [22]. Controlled environments refer to physically supervised environments under the control of representatives of government, authorities, or other official observers from Non-Governmental Organization (NGOs) or international bodies, such as in polling stations. Uncontrolled environments refer to contexts not physically supervised and typically relying on the Internet to cast votes [43].

In general, Internet-based voting systems, specifically remote voting or online voting, have both advantages and disadvantages. The major advantage of online voting is convenience for people who have a good Internet connection and any smart
device. Online voting systems remove the requirement of going to poll stations and waiting in long queues. Furthermore, voters can cast their votes from anywhere in the world. Voters find using electronic ballots less complex to unmark, write off, or void the ballot, and with appropriate security, the outcomes are more accurate because of automated calculations. If the voting process is only a little click on a device, people will likely perform their voting duties gladly, so online voting increases the turnout, especially among the youth due to their knowledge of technology and smart devices. Online voting also removes the need of special machines, papers, ink, votes-counting volunteers, election day laborers. In addition, with the use of appropriate online voting systems, outcomes convey on time with higher accuracy, increase efficiency, and save more money.

On the other hand, in term of drawbacks, difficulty of ensuring security is considered the obvious disadvantage in online voting systems. The Internet is an uncontrolled environment, which means exposed to hacking, attacking, and tampering. Moreover, make sure of the voters’ identities through this environment is difficult. Also, controlling a massive number of viruses or attacks is a costly task, and organizers would never guarantee 100 percent safety. Poor people are unable to vote because they do not have computers or Internet connection, which are available only to those who can afford the cost. All these drawbacks create kind of distrust, especially for people who do not like technology or computers.
1.1 Motivation

**Saudi Vision 2030**  The first motivation behind choosing this topic is the digital transformation program, which is one of the important Saudi Vision 2030 programs. This program aims to build a digital society, economy, and homeland in a manner that ensures transformation into a digital society based on creating digital platforms to enrich interaction and effective community participation in a manner that contributes to improving the experience of the citizen, resident, tourist, and investor in the Kingdom and a digital economy to develop the industry, improve competitiveness, and positively impact the economy, creating knowledge jobs and providing better services to beneficiaries and a digital home to stimulate creativity by attracting local and global investments and partnerships in the fields of technology and innovation.

**Estonia History of Electronic Governance**  Estonia was the first country worldwide which implemented Internet voting throughout the nation with its first complete use in 2005 when Internet voting was used for local government council elections. In 2007, Estonia used Internet voting for parliamentary elections, which was the first such event in the world [40]. The 23.4 percent of the total voters of Estonia casting their votes through the Internet in 2011 shows the popularity of the Internet voting system in the country: “Estonia is probably the only country in the world where 99 percent of the public services are available online 24/7” [16, 19]. In addition, the online services led to “844 years of working time saved” [16].
Considering the successful implementation of the Internet voting system and the enthusiasm of the public to adopt the method, this system has much potential to be a default mechanism for voting. The advantages of Internet voting over traditional methods back this claim. However, the high security risks still remain a threat [23].

**Blockchain Technology** According to [20], using blockchain for digitizing the balloting system and deploying emerging technology for voting can ensure the public trust in democracy and the electoral process and boost the maximum participation of voters. Moreover, digital technology minimizes the chances of rigging if adequately implemented, and much recent research follows this technology. Nonetheless, online voting systems are still exposed to many security and operational risks and exposures.

Several factors are involved in undermining the participant’s confidence in the electoral process and lowering the turnout, including under-staffing, outdated-equipment, inaccessibility, and changes in the addresses of the voters from election to election.

The transparency and durability in the electoral process will only be achieved by addressing the vulnerabilities associated with the conventional balloting system. Therefore, the online voting systems must provide a comprehensive solution to all issues. Also, most of the first online voting systems failed to provide authenticated results, integrity, and verification.

In blockchain technology, each block is interlinked cryptographically, thus forming a secure chain that cannot be hacked or manipulated by any entity. Digital IDs are used by voters that are secured cryptographically with the private key, incorporating
different factors like voter registration, biometric data, residential and citizenship. By implementing blockchain technology, the electoral process purportedly becomes more secure, efficient, and reliable; less expensive; and easily accessible.

**Iowa Democratic Caucus**  A key historical point was the Iowa Democratic caucus 2020 polls and the fiasco related to inconsistencies in counting [30].

The application designed and utilized for this electoral process collapsed when the backup center failed to work, so the local officials were unable to record the results effectively. The application turned out to be chaotic, and the results had to be phoned in. Lengthy security and quality checks were completed on the results to examine their authenticity with the actual figures, which further delayed the results. The officials used paper trails and photos to match and validate the findings, thus claiming that the declared results were accurate.

Furthermore, the officials also denied that the application failed or that the process was hacked, although the results were not trusted and supported by many participants and candidates. Iowa officials had already criticized the application earlier. Moreover, for the first time, the underlying data and the final number of delegates each candidate was receiving were released by the party. Nonetheless, the inconsistencies were not intentional but raised concerns about all the precise accounting since the application was designed. More people participated in 2020 for the first time as compared to 2016.
1.2 Significance of the Study

- Online voting systems increase users’ participation and provide accessibility by allowing them to vote through the Internet from any place in the world and granting them access from various computer systems and from any machine that has a connection to the Internet, which make the election process convenient and fast.

- Online voting systems help in cost effectiveness, decreasing the election financial plan by shortening production expenses. Pens, paper, printing, and ballot boxes are all considerably less costly for online voting systems than for traditional voting method.

- Online voting systems eliminate the need of collecting paper ballots and manually classifying the votes so employees save time and efforts.

- In terms of security and confidentiality, well-designed online voting systems have certain protection guarantees in order to protect privacy, information secrecy, and the election process. For instance, they utilize secure servers to host the voting system websites that grant access only for the authorized responsible through unique credentials.

- Online voting systems are eco-friendly, saving resources by using a web-based system.
• Due to using technology and electronic methods during every single step in the election process, the systems are fast and accurate. No invalid, rejected, or mismarked ballots are counted, and all votes count automatically, giving a chance to announce the results quickly.

• In online voting systems, the entire voting process is auditable end to end, which allows the administrators to assure voters that their vote are properly issued and counted for the voting process.

1.3 Contribution

This study covers the authentication part by using an Identity Authentication Management (IAM) system called NAFAD, which identifies users’ identities by verifying them through the ABSHER system.

This study shows that the security requirements of the internal multi-agent system processes can be tested based on liveness system properties when they are modeled with linear temporal logic (LTL) representations. We showed that the corresponding properties of the security requirements can be verified with an automatic system.

This study proposes the integration of the verifiability at the design level by defining all the critical system’s processes and checking and verifying them as part of the system development.

Moreover, this study proves that one of the new voting schemes on the proposed election website for the Saudi e-government. In a version of approval voting in which
the size of the ballot is bounded, a set of the properties of Approval Voting are lost, in particular, Pareto efficiency, Condorcet criteria winner support, Condorcet criteria loser, and independence to irrelevant alternatives.

The implementation further opens the window to improve weaknesses people still point out in online voting. Online applications can be started with a smaller population and extended to a larger population with improved schemes at every use. Further, the fact that a vote should not only be verifiable but also private must be taken into account.

Furthermore, this study will contribute to spread the culture of transparent management among all organizations and responsible persons and also will contribute to encouraging the participation in voting process of people aged above 18. This study will help enhance the existing voting systems with verifiability of important properties, like authentication, majority, democracy, and correctness.

1.4 Dissertation Structure

Chapter 2 introduces the background and related work needed for understanding the dissertation. Chapter 3 formally defines the current concepts that are used in the dissertation and the problems that address, including verifiability issues, implications of individual voting mechanisms, and the attacker model. Chapter 4 formalizes the proposed illustrative models and details the experiments. The chapter also presents the results and analysis of this work. Finally, Chapter 5 summarizes the research
conclusions.
Chapter 2

Background and Related Work

This chapter introduces related research on voting, including common concepts, approaches, solutions, and background techniques and technologies. The chapter starts with a review of the recent trends on voting with blockchain technology, which got much focus recently due to the large amount of funding available from bitcoin and ethereum trading, but advertised voting solutions based on technology suffer from lack of energy efficiency and lack of underlying democratic representation of the relevant constituency in the mining community responsible for robustness and correctness. Subsequently, the chapter presents other famous cryptographic protocols with known merits and drawbacks, which help calibrate expectations. Finally, the chapter introduces verifiability mechanisms based on software analysis, as well as tools and databases available for voter authentication at national level in Saudi Arabia before laying out the most relevant criteria for evaluating voting mechanisms.
2.1 Blockchain Technology

According to [27], blockchain is a distributed secure database that consists of a string of blocks that contains information where they are cryptographically linked together. Blockchain technology was originally described in 1991 by a group of researchers [17] and was intended to timestamp digital documents, tamper with them is not possible. However, the technology had not been widely used until it was adapted by Satoshi Nakamato in 2009 to create the digital cryptocurrency Bitcoin. Blockchain, a distributed ledger that is completely open to anyone, has an interesting property in which the data becomes very difficult to change once recorded inside the blockchain. Each block contains the following:

- The block number is a serial number used to keep the chain organized.
- A nonce is a number used only once.
- Data stored inside the block depends on the type of the blockchain. For example, the bitcoin contains data about a transaction such as the sender, receiver, and number of coins.
- A previous hash links a block with a previous block, effectively creating a chain of blocks, and the technique makes the block chain secure. For instance, block number 2 points to block number 1, and block number 3 points to block number 2 and so on. The first block in the chain cannot point to a previous one since
it is a genesis block where its previous hash is all zeros.

- The hash identifies the block and all its contents, much like as a fingerprint. Once a block is created, its hash is calculated. Changing data inside the block generally causes the hash to change; in other words, hashes are very useful when users want to detect changes in the blocks. If a fingerprint matches a block, then the fingerprint is assumed to be of that block.

Tampering with the second block highly likely causes the hash of the block to change as well. In turn, tampering makes block 3 and all following blocks invalid.
because they no longer correspond to a valid hash of the previous block, so changing a single block makes all following blocks invalid. Using SHA-256 hashes is considered enough to prevent tampering. Moreover, computers these days are very fast and calculate hundreds of thousands of hashes per second.

Some blockchain verifiability is based on a proof-of-work mechanism that controls the speed of creation of new blocks. For example, to add a block, one has to generate and try values for the nonce component of the block until a value is found which yields a hash with a required number of trailing zero bits. The computational equipment, time, and energy to generate a block has to be sufficiently significant so that the work to change the block includes redoing all the blocks considered impractical to attackers because the blocks are continuously chained after any change.

In the case of bitcoin, the number of required trails of 0s in the hash is calibrated dynamically to calculate the required nonce as proof-of-work when adding a new block to the chain, which takes about 10 minutes.

The mechanism makes the tampering with the blocks very hard because hackers must recalculate the proof-of-work of all the following blocks when they tamper with one block. Basically, the security of the blockchain comes from its creative use of hashing and proof-of-work mechanism. Also, one way blockchains secure themselves is through distribution. Instead of using a central entity to manage the chain, the blockchain uses peer-to-peer networks, and in some cases, any user is allowed to join. When people join the network, they can receive a full copy of the blockchain called an
open ledger or immutable ledger. Ideally, the node does use the ledger to verify that the data is still in order, which on long ledgers, unfortunately, is currently executed only by nodes with high stakes in the given ledger. When created, the block is sent to everyone in the network. Each node then verifies the block to make sure that it has not been tampered with. If all data check out, each node adds the block to its own blockchain, and the ledger is updated. All the nodes in the network build consensus in time, which means they eventually agree what blocks are valid and which are not. Inconsistencies due to lack of synchronization are assumed to converge based on majority adoption. Blocks that were tampered with are rejected by other nodes in the network, so to tamper successfully with the blockchain, the hacker has to take control of more than 50 percent of the peer-to-peer network or break the hash function.

2.1.1 Concepts in Blockchain Technology

- **Secure Hashes.** A Secure Hashing function is a cryptographic one-way function that accepts any sized input array of bytes and returns a fixed-sized array of bytes called a digest. Since the cardinality of the input space is much higher than the cardinality of the fix-sized output space, multiple inputs can be mapped to the same output. From the resulting digest, users cannot retrieve the original input. The hashing function is not encryption but is a one-way method, and the result cannot be inverted, which makes the hash number suitable for comparing the hashed results version of the texts rather than decrypting the text to gain
the original. Furthermore, to generate collisions, namely to find any two inputs with the same digest, is assumed hard, even if the collisions are theoretically known to exist.

All digital media like documents, movies, and music are only strings of binary digits such as zeros and ones. A hash function takes any digital media and runs an algorithm to produce a fixed-length digital output, known as a hash or digest, which may be much smaller than the original input. The function is deterministic, producing the exact same digital output or hash every time the same digital media is put through the hash function. A hash function has an avalanche effect, which means any simple change in the original text is expected to affect about half of the result bits. When even a single bit in the digital media is changed and then put through the hash function, the digital output (hash) is completely different from the original one. The mathematics behind the hash function ensures that deriving the original digital media content from its generated hash is impossible. The technique makes the hash function a one-way function. Many hashing functions are proposed, such as Message Digest (MD5), Secure Hash Function (SHA), RACE Integrity Primitives Evaluation Message Digest (RIPEMD), and Whirlpool. The blockchain technology in bitcoin uses SHA256 hashing algorithm, which is 256 bits, and the result shows as 64 hexadecimal characters.

- **Immutable Ledger.** A ledger is a database of records that simplifies the
process by registering all transactions and known assets in the whole peer-to-peer network, and the record cannot be changed or altered other than by appending new transactions. Also, each peer in the network has copies of the ledger.

- **Distributed Peer-to-Peer Networks.** The ledger transactions are created using a consensus process among a set of peer software agents.

- **Mining.** In a blockchain, the mining requires the miners to solve a problem with a known portion of input derived from the latest state of the blockchain to create a hash target. The miners must try to guess the digital input, then create the hash target, and solve the problem. As a hash is one way, the miners must try many combinations of input to create the hash target and solve the problem which uses computer resources like CPUs and memories. The fastest miner to answer the problem is the most likely to win.

- **Merkel Tree.** The Merkel tree is an important part of a blockchain for data integrity; each block in the blockchain contains multiple transactions that contain data. For instance, of transactions1, transactions2, transactions3, and transactions4, each of the full transaction data is passed through a hash function generating full unique hashes, one for each transaction. The transaction passes through hashes and then combines and passes through the hash function again. The process generates two separate and unique hashes, each based upon the combination of two hashes relating to the two transactions. The two hashes
are then combined and pass through the hash function again; the result is a single root hash forming a complete Merkel tree. The Merkel tree allows for the detection of any changes to any data within the transaction of a block by simply rerunning through the process for each transaction and comparing the result to the original root hash.

- **Consensus Protocol.** The protocol means all nodes in the decentralized network must come to the agreement on what the truth is. All nodes must agree on the transaction history. For example, proof-of-work is one way to archive consensus and can be based on mining.

### 2.1.2 Proof-of-Work Versus Proof-of-Stake

**Proof-of-Work**  A mechanism is used to reach consensus between many nodes in the network and to secure bitcoin blockchain. However, the proof-of-work algorithm operates by having all nodes solve a cryptographic puzzle with controlled difficulty. The puzzle is solved by miners, and the first one to find the solution earns the miner reward. Proof-of-work uses an election process in which one node is randomly chosen to validate the next block. To increase chances even further, miners can come together and create a miners’ pool, and they combine their hashing power across disjointed input sets and distribute the rewards evenly among everyone in the pool. Proof-of-work causes miners to use massive amounts of electricity, and miners’ pools turn the process into a less centralized one than originally supposed. [24]
Proof-of-Stake  A mechanism that allows users to compete against each other by mining. The miners are called validators, and the mining process is called minting or forging. Validators are not chosen completely randomly. To become a validator, a node has to deposit certain number of coins into the network as a stake, like a security deposit. The size of the stake determines the chances of a validator to be chosen to forge the next block. The process is like a linear correlation. If $X$ deposits 100 dollars at the network and $Y$ deposits 1,000 dollars, $Y$ has 10 times more of a chance to be chosen to forge the next block. In order for the process not to favor only the wealthiest nodes in the network, more unique methods are added to this process. The two most commonly used methods are randomized block selection and coin age selection. In the randomized block selection, the validators are selected by looking for coming nodes with a combination of the lowest hash value and the highest stake since the sizes of the stakes are public, and the forger can be predicted by other nodes. The coin age selection method chooses nodes based on how long their token has been staked. Coin age selection is calculated by multiplying a number of days staked by a number of coins staked. Once a node has forged a block, its coin age reset to zero and must wait a certain period of time to forge another block, which prevents a large stake node from dominating the blockchain.

All cryptocurrency owners using the proof-of-stake algorithm have their own set of rules and methods for what they think is the best possible combination for the owners and their users. When a node is chosen to forge a next block, it checks if the
transactions sending the block are valid, signs the block, and adds the block to the blockchain. As a reward, the node receives the transaction fee that is associated with the transaction in the block. If owners want the node to stop being a forger, the stake along with the earned rewards are released after a certain period, giving the network time to verify that no blocks are added to the blockchain by the node. [4]

2.2 Benaloh Election Protocol

One of the main voting process properties is privacy, which means the following:

- No election authorities or anyone else can link any ballot to the voter who cast it.
- No voter can prove that they voted in a particular way.

In “Receipt-Free Secret-Ballot Elections,” the authors describe verifiable secret-ballot election protocols, which prevent voters from issuing any receipt that would permit them to prove a vote was cast in a particular way. [1]

Traditional physically based election systems are commonly believed to satisfy privacy and uncoercibility, but no technique allows voters to be assured that their votes are accurately counted. Various proposed electronic election systems appear to satisfy privacy and accuracy, but the systems fail to satisfy uncoercibility.

By using cryptographic means such as the Rivest–Shamir–Adleman (RSA) or Byzantine agreement protocols, private channels are produced from public channels.
Several subsets of the processes are specified as voters, and voters have two independent processes symbolized as $V_0$ and $V_1$ for the two possible voting intentions.

The election’s tally is known as the number of voters who run the $V_1$ protocol. A tally $T$ is defined to be valid if $T_L < T < T_H$, where

- $T_L$ is the number of voters who ran the $V_1$ protocol.
- $T_H$ is the total number of voters less the number of voters who ran the $V_0$ protocol.

According to the authors, “The voter can record the bit stream arriving on any channel, public or private, but can also forge a record of any raw bit stream arriving on a private channel” [1]: While the voter is in the voting booth, no other process may observe the voter’s actions or the information arriving on the voter’s input channels.

We also assume the existence of a beacon which produces an unlimited supply of unpredictable bits. Such a beacon is either constructed physically, perhaps as a gauge monitoring the activity of a natural random source or synthetically by XORing bits supplied by agents of the election system. For coercibility $1/2^N$, an election system with $M$ voters is correct if a specially designated output common to all participants who follow correct protocols gives a common correct tally with a probability of at least $1 - M/2^N$ for a given security parameter $N$.

Furthermore, an election system is private if no dishonest protocol participants enable any other participant to distinguish between a voter running $V_0$ and another voter running $V_1$ with a probability greater than $(1/2 + 1/2^N)$.
Next, a protocol $P'$ is publicly consistent with a protocol $P$ if $P'$ running on input $(s)$ and $x$ produces publicly observable output $(s)$, which could have been produced by $P$ running on $x$. A protocol $P$ is coercible if some protocol $P'$ is publicly consistent with $P$ and provides an additional private receipt, which is used to prove that $P'$ is consistent with $P$. Finally, an election system is uncoercible if neither of the two voting protocols $V_0$ and $V_1$ is coercible. [1]

### 2.2.1 Homomorphic Encryption

The principal tool for the Benaloh technique is homomorphic encryption, which builds encryption functions that have certain properties. For example, if $A$ is an encryption of $a$ and $B$ is an encryption of $b$, the product of the two encryptions yields an encryption of the product of the plain text. An alternative construct, multiplying two encryptions yields an encryption of the sum of the plain text.

### 2.2.2 Voting Process

The general idea of the Benaloh protocol is as follows:

- A pair of an encrypted 0 and an encrypted 1 in random order is created by an authority as ballot = $(x_i, y_i)$.
- A homomorphic encryption is $(+, \times)$ and $E(m, k) = g^k || y^k \ast a^m \mod p$.
- The authority uses a beacon to provide an interactive proof to any observer that the ballot is valid.
• In the voting booths, voters receive extra information.

Thus, with extra knowledge, voters declare publicly which half of the ballot they cast as vote.

Vote-Casting Protocol

• An authority produces $N + 1$ pairs $(2N + 2)$ of encrypted ballots, where $N$ is a parameter for the confidence level of the Zero Knowledge proofs and $(a_i, b_i)$, where $a_i = min[E(x_i), E(y_i)]$ and $b_i = max[E(x_i), E(y_i)]$.

• Voters enter a booth.

• The authority transmits $c_i = D(a_i)$ to voters for $i$ in $0...N$ over a private channel.

• A beacon generates $N$ random bits $r_i$ over a public channel.

• Voters leave the booth.

• The authority publishes $[D(a_i), D(b_i)]$ for each $i$ such that $r_i = 0$, along with a $[D'(a_i), D'(b_i)]$ certificate.

• For $r_i = 1$, authority proves that $a_i$ is equal to either $a_0$ or $b_0$.

• To cast a 0 vote, voter $k$ publicly transmits $v_k = a_0$ if $c_0 = 0$; otherwise, $b_0$.

• To cast a 1 vote, voter $k$ publicly transmits $v_k = b_0$ if $c_0 = 0$; otherwise, $a_0$. 
Tallying and Publishing the Results

- All observers can compute the compound vote for \( m \) voters: \( V = v_1 \times v_2 \times ... \times v_m \).

- The authority reveals the election tally by computing and exposing the \( D(V) \) along with the certificate \( D'(V) \), where \( D(V) = \) number of 1 votes cast, and \( D'(V) = \) the proof of the correctness of the tally.

At a closer look of the privacy and uncoercibility properties, privacy, which assures that no one can detect how voters vote, is totally subsumed by the uncoercibility property, which assures that no one can detect how voters vote even with the voter’s collaboration. In a common condition, uncoercibility is hard to show because no exchange voter protocol appeared to permit coercion. However, as described in the election system, the voters only monitor or record information and finally transfer a single bit to single choice. “The voter does not interact other than to select which of two (encrypted) votes it wishes to cast! Thus, a greatly simplified proof approach is possible. By showing that no third party, even given the information held and revealed by a voter, can distinguish a 0 vote from a 1 vote, the uncoercibility will be attained.” [1]

2.3 Merritt’s Election Protocol

To demonstrate the main idea of Merritt’s protocol, an example of four voters, \( W, X, Y, \) and \( Z \), are required to have public keys that are supposed to be distributed
before the election starts. According to [39], the voting process is as follows:

- Voters pick a random number \(a\) and compute \(E_W (E_X (E_Y (E_Z (v, a)))) = R\).

- Voters pick a random number \((a_1, a_2, a_3, a_4)\), and compute \(E_W (E_X (E_Y (E_Z (R, a_1), a_2), a_3), a_4)\).

A verification step consists of two rounds. In the first round, decryption occurs in the order \(W, X, Y,\) and \(Z\); in this round, all votes are sent to \(W\), who decrypts the votes with a private key and eliminates all the random numbers at that phase. \(W\) mixes up the order of all the votes and sends the result to \(X\) and so on for all voters.

For instance, after \(W\), each vote is like \((E_X (E_Y (E_Z (E_W (E_X (E_Y (E_Z (v, a))))), a_1), a_2), a_3)\). At the end of this round, after \(Z\) completes the same process, each vote is like \((E_W (E_X (E_Y (E_Z (v, a))))\)). In the second round, decryption occurs in the order of \(W, X, Y\) and \(Z\) one more time. All votes are sent to \(W\), who decrypts all votes with a private key, examines to see that his vote is among the group of votes, signs all votes, and passes the message onward. For instance, after \(W\), each vote is like \((S_W (E_X (E_Y (E_Z (v, a)))))\). At end of this round, after \(Z\) completes the process, each vote is like \(S_Z (v, a)\).

**Tallying Process** All voters verify and delete \(Z\)'s signature and examine to assure that their votes are among the group of votes by searching for their random number. All eliminate the random numbers and tally the votes.
In term of advantages, the voters know immediately when someone tries to cheat, using the protocol allows for anonymity and decentralization. As drawbacks, the protocol is impracticable due to demanding all the participants to be online at the same time. Also, $Z$ knows the outcome of the election before anyone else, and $W$ has the ability to copy anyone else’s vote, even though $W$ does not know the vote. [38]

2.4 Verifiability Mechanisms

This section reviews verifiability mechanisms for online voting systems that enable voters and auditors to check properties of the voting process based on evidence from logs generated during the voting process. For example, the mechanisms employ the trace of cryptographic, logical, or statistical proofs.

2.4.1 Analyzing Implementations of Election Technologies

One of the main requirements and properties of online voting systems is verifiability, a property characterized by the capacity of voters and observers to check evidence produced through the implementation of the voting process and to attest that the results are correct. In other words, all votes can be verified to make sure they are correctly accounted for in the final stage. The evidence consists of cryptographic, logical, or statistical proof. Computation-based evidence such as formal certificates and log files is checkable by third parties in which they utilize evidence checking tools, like log analyzers or certificate checkers.
Evidence Flow  Evidence supports a proof that the execution of the online voting process handles the vote ballot correctly, and flow refers to the path from the beginning point, which is the source to the finishing point for the ballot processing. End-to-end verifiability refers to three properties: the ballot must be cast as intended, recorded as cast, and counted as recorded. For example, each property has corresponding evidence such as an authentication code through Short Message Service (SMS) or a receipt generated after the ballot cast. To combine evidence flow, which means the source and end point flow could be merged, is feasible.

Two approaches are applied to evidence flow checking. The first solution follows the forward path of the ballot from the start of the process until the evidence is produced. Utilizing the approach assures that the evidence has been produced correctly by applying the online voting system’s procedures; for instance, SMS authentication and receipt issuance.

The second approach analyzes the evidence flow in a backward manner, which means the evidence produced in the end point must match a particular input. In other words, the evidence must be traceable back to a valid ballot input. Moreover, the evidence flow the execution of online voting system has much content, like class hierarchy, function calls, and data flows.

Policy Language  The system’s behavior is the mechanism by which one action leads to another. During the designing of online voting systems, the system’s behavior is usually predefined in specifications. Linear temporal logic (LTL) is a logic system
commonly used to reason with complex dynamic systems [34, 13, 37].

Based on a finite set of atomic propositions, LTL is equipped with logical or temporal operators such as the temporal connectives. until denoted $\mathcal{U}$, since denoted $\mathcal{S}$, release denoted $\mathcal{R}$, next denoted $\mathcal{X}$ or $\Diamond$, previously denoted $\Box$, globally or always in the future denoted $G$ or $\blacksquare$, always in the past denoted as $\blacklozenge$. Also, eventually will happen in the future is denoted $\Diamond$ while eventually happened at some time in the past is denoted $\blacklozenge$.

For example, next $\mathcal{X} \pi$ means that $\pi$ is true in the next time step after the current one, and $\pi \mathcal{U} \Psi$ means that either $\Psi$ is true now or $\pi$ is true now and $\pi$ remains true at least until $\Psi$ holds. On the other hand, $\pi \mathcal{R} \Psi$ means $\pi$ releases $\Psi$, specifies that $\Psi$ should be true now and remains true until $\pi$ is true, inclusively. Furthermore, LTL, commonly used for formally describing properties of dynamic systems, allows representation of many verbal English language requirements with unambiguous logic expressions.

The basis of LTL is propositional logic, the main difference lying in the addition of the temporal operators. Temporal logic allows for making deductive arguments about not only what the process is, but also what it was, has always been, and will always be.

Common LTL-provable attributes of a system are phrased as liveness, safety, correctness, and invariants [34]. Liveness, a family of properties stating that a given predicate keeps happening past any specific time in the future, is used to prove lack

27
of deadlocks by showing that desired reactions keep occurring in a server.

An LTL property \( \phi \) is used to specify a system requirement such that the requirement can be formally reasoned with or proven true in certain time steps. For each time step, a property is assumed to be either true or false, and the LTL cannot reason with time resolutions smaller than the selected time discretization.

Every valid LTL formula can be translated to a corresponding Büchi automaton that accepts all and only the infinite traces that satisfy the formula [42]. The Büchi automaton obtained by conversion from an LTL formula \( \phi \) is denoted by \( B \phi \). To show that a system model satisfies an LTL formula \( \phi \), the formula can be converted to a Büchi automata and proven using corresponding tools.

Based on [44], in temporal logic, the specifications can be redefined as an executable model, which has structures like the following: “If \( A \) holds in the past, then do \( B \)” [44], where \( A \) and \( B \) are actions of the system. Moreover, each state in the system models is handled differently depending on the point of time.

Approaches to the use of LTLs splits into two families, past linear temporal logic (PLTL) and future linear temporal logic (FLTL). Each one has several temporal forms. The syntax used in the two approaches to temporal logic is expressed as follows:

- Syntax of PLTL

\[
\Phi, \Psi ::= true \mid p \mid \neg \Phi \mid \Phi \land \Psi \mid \Phi S \Psi \mid \Phi \mid \Box \Phi
\]

- Syntax of FLTL

\[
\Phi, \Psi ::= true \mid p \mid \neg \Phi \mid \Phi \land \Psi \mid \Phi U \Psi \mid \Phi \mid \Box \Phi
\]
For instance, after the voting server receives the ballot message from a voter who votes, the server must validate the ballot message. The property is described with FLTL like: $\Box (receiveBallot \rightarrow ◇ validateBallot)$.

**Büchi Automata.** According to [14], a Büchi automaton is a finite automaton, which means it is nondeterministic and can be defined as $A= (Q, Σ, I, T, F)$, where

- $Q$ is a finite set of states.
- $Σ$ is a finite set of literals constituting the input alphabet.
- $δ \subseteq Q \times Σ \times Q$ is called the transition relation.
- $Qs \subseteq Q$ is the set of initial states.
- $F \subseteq Q$ is the set of accepting states.

Deterministic Büchi automata define transition functions as $δ : Q \times Σ \rightarrow Q$. Generalized Büchi automata are nondeterministic Büchi automata with a set of initial states and sets of final accepting states where a state from each such set need to appear infinitely often in an accepted run. Very weak alternating automata [36, 14] modify the Büchi automata to be a set of partially ordered states and a set of possible initial states, in which the transition function is defined from a state to a relation between conjunctions of multiple input alphabet literals and multiple resulting states in the partial order on states, enabling modeling of nondeterminism.
Büchi automata conversion from future-time linear temporal logic is supported by Gastin and Oddoux’s [14] algorithm, which is composed of three procedures as follows:

- Preprocessing the original formula to a noncanonical negation normal form $\phi$, namely, where negations are moved inside parentheses and double negations are cancelled.

- Translate $\phi$ into a generalized Buchi automaton $A'_\phi$ [44] with the help of very weak alternating automata.

- Post-processing $A'_\phi$ simplifies and translates it into a Buchi automaton $A_\phi$.

**An Abstraction Language**  The process of checking programs with the help of verification automata is performed in two steps: developing the automaton and preparing the input for the automaton. In [44], the author showed how to extract the input to the automaton from the system that needs to be verified, which is the Vvote system. “The abstraction language is an abstraction from practical programming languages including only some essential components”: the grammar of an abstraction language represents the content of a function. The abstract input to the automaton is represented by a set of paths. For example, the word `identifier` is used to represent a sequence of letters, digits, and underscores. The word `integer literal` is used as a sequence of decimal digits that denotes the corresponding integer value, including negative signs. The symbol `*` is utilized to represent repetition and `()` to represent
group of items together. Also, the loops like for or do..while, are reformed to be while loop. Based on the abstraction language, a mapping between the logic predicates and the actual function names is generated.

**Application: Source Code Analyzer**  Static analysis is an automated testing of a source code. The idea of the static analysis of computer software is to perform the analysis without executing the program. Source code analysis translates the source code into an intermediate model, optimized for the analysis step. during this step, the code scanner tool produces an abstract model. Many abstraction levels start from the implementation source code to the abstraction language program and then from the abstraction language program to the call graph. The correctness and termination of the abstraction process from the implementation language to the abstraction language relies on the code scanner tool that is used. In some source code analyzing tools, several security policies are predetermined, so users can examine the software by matching patterns of predetermined conducts in the source code. [44], [5], [25]

Source code analyzer tools include the following:

- Coverity is a code scanning tool computing the set of all abstract execution paths. The advantage of using such tools is that they work with real programming languages. During translations, the Coverity tool generates an abstract model from the source code, called an abstract syntax tree, and then provides the execution path of the implementation during the traversal from root to end nodes.
• SPIN, a model checker tool introduced by Gerard J. Holzmann [18], is focused on proving the correctness of the process and tries to abstract as much as possible from internal sequential computations. The SPIN tool deals with formulas in LTL forms and generates a Büchi automata and provides the Büchi automata in PROMELA syntax.

• Sonarqube, a code scanning tool that inspects the code for quality purposes, does the static analysis scan with focus on bugs, code smells, vulnerabilities, and code duplication.

• Doxygen is a static code analysis tool that generates documentation from a source code. The tool supports high level-programming languages, such as C, PHP, Java, and Python, scans a source code, and visualizes the relations between the various elements by means of dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.[11]

2.4.2 Epistemic Policies for Voting Systems

The properties that can be verified depend on log checking. In modern systems, when controversies occur concerning claims about wrong results during an election, the results are sometimes stabilized by examining the logs, which are the only preserved records of the processes that happened during operations. Typical questions are whether a vote was not properly counted, whether a vote was altered by a malicious one, whether the election commission permitted voters to vote more than one time,
or whether the secrecy of a vote was compromised.

In modern voting systems, voter verifiability may be provided with extra techniques, letting voters examine if their own vote was cast as intended and counted as cast. Moreover, verifiability relies on logs when settling claims. For instance, was a server compromised, or did the software have a temporary malfunction, causing a system to fail? A way to cope with the problem is by developing an accurate model of what a valid log must look like, and the development can be accomplished by representing its characteristics in a mathematical succinct way. The representations are typically part of the security policies. Security policy languages are typically customized to the implementation field.

A model checker for the epistemic first-order temporal logic (EFOTL) to convey security protocols for communicating and distributed systems is proposed in [44]. EFOTL is an expansion of first-order linear time temporal logic with an epistemic connective representing knowledge for specific agents. The distinctive features of EFOTL is a group of formal connectives, also named knowledge modalities, which allow conveying epistemic security policies in terms of what each representative knows.

Each point in a distributed system, such as the server, client computer, and printer, are considered a representative, and by utilizing several knowledge modalities, they convey security policies that include many representatives briefly but comprehensively. Conveying universal security policies in EFOTL is represented through user-defined predicates, like $knows(a, P)$, which capture the fact that a represen-
tative \( a \), such as representing a voter, a database, or a printer, knows \( P \). Despite the fact that the conveyance of universal security policies are feasible, they lead to complicated and convoluted security policies that are hard to maintain.

The collection of epistemic and temporal characteristics of EFOTL enables drawing up a large category of significant policies for voting systems. Examples of epistemic security policies are the authentication between two or more representatives, commitments to shared secrets, examination if thresholds were reached before decisions were made, and protocol complying with communication patterns.

Every sent instruction started by one representative should be matched by a received instruction of another. Ballot papers should be created, stored, retrieved, completed, and submitted or audited, and all events should occur in a specific order. The knowledge modalities of EFOTL allow conveyance of security policies for the whole system while pointing to local knowledge logs of each representative.

For the purpose of the study in [44], the author presumed that each log entry is uniquely identified as pertinent to a specific session. Particularly for voting systems, each log has a reiterated series of the similar proceedings due to execution of the same protocol over and over. To exploit the similarities between several runs of the same protocol, to think about equivalent sessions is reasonable, which means those sessions cannot be then distinguished by examining the logs.

**A Language for System Properties** EFOTL is expanded by epistemic connectives ordered by representatives, often referred to as a specific contributor in a commun-
nication, an agent, a machine in a distributed system, or a process. Representatives are symbolized as $a, b, ...$ and the set of all representatives as $A = \{a, b, ...\}$. The set of all time points are denoted as $I = \{i, j, ...\}$. Every log is commonly time stamped with the moment the data was written and can be verified for matching to a time point related to the other logs and time points. Many logs are produced by one protocol running a session. Also, sessions are denoted with modal worlds which are identified by $W = \{w, v, ....\}$. The algebra expression of EFOTL is finite and produced from variables $V := \{x_0, x_1, ...\}$ and uninterpreted constants $C$. The set of expressions is $T := \{t_0, t_1, ...t_n\}$, and the set of predicates is $P := \{P_0, P_1, ...\}$. The predicates can be extracted from the log to be examined.

**Finite Model Checking** A result is that model checking is recursive, meaning that a determination is made about whether a formula is accepted in a specific model or not.

**Model Generation** A Kripke structure is $\mathcal{M} = (W, D, \mu, (Ra)_{a \in A})$, where $D$ is the domain, and each $Ra \subset W \times W$ describes the reachability relation on worlds that relate to representative $a$. Various representatives have different visions of the worlds.

The following tasks are used to rebuild a Kripke structure:

- Translate the plain text into pairs of EFOTL predicates and worlds.
- Partition the logs according to the world.
- Map real time into discrete time points that are understood by EFOTL.
• Update the reachability relation according to the symmetries among sessions.

**Basic Definitions** The basic concepts regarding the voting schemes are defined in [44] as the tuple “Voting scheme, ballot, ballot box, election results,” where given a non-empty set \( C \) of candidates, a voting scheme \([B, T]\) is characterized by the following:

- A set \( B \) of possible ballots that can be cast by voters.

- A tallying function \( T \) assigned to each possible ballot box as an election result, in which a ballot box is a finite multi-set of ballots and an election result, is a finite duplicate-free sequence of elected candidates. Given a non-empty set \( V \) of voters, every voter is presumed to cast only one ballot in the election. Tallying functions are deterministic.

- A ballot for a preferential voting scheme orders a list of candidates according to the voters’ preferences.

### 2.5 Official Saudi Authentication Systems

In Saudi Arabia, two platforms are officially supported for providing authentication services, namely the ABSHER database and the NAFAD access portal to ABSHER.
2.5.1 ABSHER

ABSHER (Arabic for “help with pleasure”) is an electronic platform for the Saudi Ministry of Interior that serves citizens and residents in the Kingdom of Saudi Arabia and facilitates a very large number of government procedures. The platform helps users to avoid personal trips to the official state institutions to avoid crowding and to facilitate quick transactions, keeping pace with technological developments. ABSHER is a completely free system for users, who need only to register, and then they can take full advantage of many services provided by the ministry and its various sectors.

The ABSHER registration process for residents and individual citizens is not complicated. Visitors of the official website of the Saudi Ministry of Interior portal ABSHER can sign in and register an account in three steps. The first step involves user data, in which users are required to enter nine fields:

- Identity Number of 10 digits
- Valid email address
- Confirmation of the email address
- Username in English letters
- Mobile number of the user
- Password that contains at least 8 characters (uppercase and lowercase, a number, and a special character)
• Confirmation of the password

In the second step, users must click on the box to confirm that the terms and conditions have been read and then click on the button Next in order to receive a verification code as a phone message. Users enter the verification code and click on Register to create an account.

After completing the second step, users find the following message: “Your account has been successfully registered.” After completing registering an account with the portal, users must activate their accounts to benefit fully from the platform.

**ABSHER Service Activation for Individuals.** The three official ways to activate ABSHER accounts on the Ministry of Interior’s electronic portal all rely on fingerprinting the users and relating the fingerprint with the users’ accounts. The first is to appeal to the staff offices for activation in the Ministry of Interior, such as the General Department of Passports or Civil Affairs, which are accredited offices for registration and activation found throughout all Saudi cities. The second is to activate the accounts through approved banks. The third is through self-service registration and activation machines, in which users can activate their accounts without visiting the aforementioned offices. The registration can be completed by fingerprinting through the self-activation device kiosks made available for citizens and residents at various locations like stores or hospitals. The devices are distinguished by ease of use and safety in completing electronic transactions. Upon registration, a confirmation message arrives on the users’ phones linked to their accounts.
Integrated with many sectors in Saudi Arabia, ABSHER supports electronic services and electronic queries for each of the following:

- General Directorate of Passports
- General Department of Traffic
- Ministerial Agency of Civil Affairs
- General Department of Expatriates Affairs
- Directorate General of Prisons
- Public Prosecution
- Power of Attorney

2.5.2 NAFAD

NAFAD is an official interface to ABSHER, thereby providing a secure access to identities for general business applications besides the previously mentioned government services. Creating a new account in NAFAD redirects users to the ABSHER portal to create an account there also if the users do not yet have an account with ABSHER. Then, the users can use the ABSHER credentials to sign in at the NAFAD portal to have the advantage of all offered services. Table 1 indicates the attributes that NAFAD transmits by default to service providers.

Table 1
**NAFAD Users’ Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>lang</th>
<th>arabicName</th>
</tr>
</thead>
<tbody>
<tr>
<td>nationalID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>englishName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arabicNationality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gender</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arabicFirstName</td>
<td></td>
<td>arabicFatherName</td>
</tr>
<tr>
<td>arabicFamilyName</td>
<td></td>
<td>englishFatherName</td>
</tr>
<tr>
<td>arabicGrandfatherName</td>
<td></td>
<td>englishGrandfatherName</td>
</tr>
<tr>
<td>assuranceLevel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>issueLocationAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iquamaExpirationDataG</td>
<td></td>
<td>iquamaExpirationDateH</td>
</tr>
<tr>
<td>IDExpiryDateH</td>
<td></td>
<td>IDExpiryDateG</td>
</tr>
<tr>
<td>versionNumber</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Other custom attributes are obtained by the service providers based on special request as long as they are available in ABSHER and approved. All messages between users, service providers, and NAFAD are based on the Security Assertion Markup Language (SAML2), in which each message is timestamped, digitally signed, and passed over HTTPS with optional encryption as supported by IBM WebSphere.

NAFAD establishes sessions in which authentication is performed by redirection when users log in, and the session may end either when the users log out directly from the NAFAD’s IAM system or through logout requests to service providers.
Figure 2.3: NAFAD Login Authentication

Figure 2.4: NAFAD IAM Logout Authentication
2.6 Formal Verification

Formal verification is the procedure to prove or disprove some of the properties of the voting system or of a particular voting process. Formal verification techniques [33] are executed in three different ways:

- Manually, where a human tries to produce a proof of correctness
- Semiautomatically, where theorem-proving techniques are employed but some manual processing is still required
- Automatically, where an algorithm takes as input a model (program) and a property and then decides whether the model satisfies the property or not
2.6.1 Model Checking

Model checking, a research area concerned with verifying whether the model of a system satisfies given specifications or not, requires two components: a mathematical or an abstract model of the system under testing and specifications of the desired behaviors or requirements. Specifications refer to formal definitions of properties we want the model to satisfy.

The main goal of model checking is to analyze the abstract system model when reasoning about properties of the infinite sequences of states generated. Some checked properties are described as invariants. If an invariant is not violated, it holds for at least the infinite states in the future of the model evolution. If a single violation of a given property occurs, then the counterexample is found [10].

A mathematical model of a system in this context refers to the specification of system state transitions as a tuple \( T = \langle S, I, A, \delta, AP, L \rangle \), where

- \( S \) is the set of possible states.
- \( I \) is the initial state distribution.
- \( A \) is a set of inputs.
- \( \delta \) is a relation modeling potentially nondeterministic transitions between any two states given an input.
- \( AP \) is a set of atomic propositions which are either true or false at any given time.
• $L$ is a labeling function telling which members of $AP$ hold true for each state.

The automata defined by $T$ accepts inputs from $A$ as specified in $\delta$ and nondeterministically produces sequences of states, which by virtue of $L$ translate into sequences of corresponding sets of atomic propositions from $AP$, denoted $\Omega$ [26]. An abstract model [15] of a system in this context refers to representation with a language that skips certain details.

Static code analysis is an automatic program checking process commonly implemented with two components: an automaton and the inputs for the automaton. Static analysis tools extract the input from the source code of the system that needs to be verified and feeds it into the automaton. The tools take the source code and translate it into an intermediate model for analysis, such as a call graph. Each path contained in the call graph is considered as an input path for the automaton for property checking.

To start the process of model checking for a property $\phi$, the property must be converted to the Büchi automata $B \phi$ such that the inputs to the $B \phi$ are from the set of inputs $A$ as specified in transition model $\delta$. If all the inputs are accepted in $B \phi$, then the model satisfies the property $\phi$ by the definition of the Büchi automata. If at least one counterexample exists that is not accepted in $B \phi$, then the model does not satisfy the property.

**LTL2BA** The tool LTL to Büchi automata (LTL2BA) [14] is an available software used to translate an LTL formulas into a Büchi automaton that automatically draws
the resulting automaton and generates code in the PROMELA language, which is
given to a model checker tool for verifying properties of a system.

Process meta language (PROMELA) is used to specify abstract models. A modeling language introduced by Gerard J. Holzmann is the SPIN model checker tool [18]. A program in the language is composed of global declarations such as process types, shared variables, communication channels, and sequence of statements. PROMELA is used to build verification models, which represents the abstraction of a design and contains only the properties relevant for the desired verification. The following is a sample of PROMELA:

```plaintext
int Y;
proctype N( )
{
}
proctype A( )
{
    int X = 2;
    run N( );
    Y = 1;
    Y > 2 && X == 0;
}
```
In the sample, run \( N \) means execute \( N \) if \( N \) can be created. The test \((Y > 2 \&\& X == 0)\) is executable and allows passing to the next statement if some other process makes \( Y \) greater than 2 and \( X \) zero. The current process assigned \( Y \) as 1. [7]

### 2.7 Voting System Properties

**Approval Voting** is a voting mechanism consisting of a ballot format and winner selection method that have been researched in the context of voting and is typically used when a high number of voters and a small number of choices are in the election. [9, 8]

With approval voting, voters vote for multiple candidates, as many as they desire, in an election. Under common implementations, approval voting satisfies different sets of criteria, such as Pareto efficiency, Condorcet criterion, Condorcet loser criterion and irrelevance to irrelevant alternatives. [9]

**Pareto Efficiency Criterion** means candidate \( A \) is elected only if at least one voter is disadvantaged by selecting someone else or if every voter is indifferent. [31]

**Condorcet Winner Criterion** refers to the candidate who receives a majority of the votes in every one-to-one election against each other candidate, that is, a candidate who is preferred by more voters to all other candidates, taken separately in pairs. Condorcet winners do not exist in all instances. In a plurality vote, the Condorcet winner is the individual who wins a two-candidate election against each of
the other candidates. The Condorcet winner is always the same for a set of candidates, regardless of the voting system used, and may be found by utilizing pairwise counting on voters’ ranked choices. [45]

**Condorcet Loser Criterion** is a candidate who is defeated in a one-to-one race against each other candidate [21].

**Independence of Irrelevant Alternatives Criterion** reveals whether the introduction of alternative candidates who lose changes the winner with a given voting mechanism. [2] For example, with a mechanism that is irrelevant to alternatives, if $C$ is the irrelevant alternative, namely, not a Condorcet winner, $C$ participates in a competition between $A$ and $B$ with $A$ (the leader) liked better than $B$ (not the leader), then the individual voters who like $C$ less than $A$ do not switch their vote from $A$ to $B$.

**Monotonicity Criterion** is a voting system criterion describing single and multiple winner ranked voting systems. Voters rank their candidates in a list as 1\textsuperscript{st}, 2\textsuperscript{nd}, and 3\textsuperscript{rd}. [3]

Rank polling is monotonic if it is not feasible to avoid the election of a candidate by ranking them better on some polls or to elect an unelected candidates by ranking them less on some ballots while their preference is not modified on any other ballot.

Formally, according to [12], if $f(\pi) = \{ x \}$ is a winner determination function where $\pi$ is a set of votes and if choices $\pi'$ are obtained from $(\pi)$ by moving $x$ up in
Majority Criterion means that a candidate is considered as a winner only when a majority of voters voted for him, satisfying that the winner’s received votes are > 50% of the total votes [29].

On the other hand, the plurality rule means that the candidate is considered a winner if he gains the most first place votes among all the competitive candidates [32]. According to the majority criterion, if a candidate receives a majority of the 1st choice votes, that person should be declared the winner of the election. The majority standard will not apply if no majority option is available.

The majority criteria has a comparison criterion for single-person winner voting methods. If a candidate is having a majority, meaning more than fifty percent of voters have put that candidate first, then that contender must win as per the criterion of majority [29].

The system may fail if A is favored, meaning that voters offer higher votes than all other candidates. A is favored by a large majority of voters, but practically all of them have the majority. However, B’s median is good while A’s is just fair, so as a result, B comes out on top and wins as per majority criterion.

The majority criteria are used in instances where a majority of voters favor one candidate over all others. There is rarely a candidate ranked top by such a majority in a contest with three or more significant candidates.

As a result, the majority criteria is typically meaningless in elections involving
more than two major parties. The majority criteria were first described in terms of voting systems based on only preferential order.
Chapter 3

Concepts and Problems

This chapter introduces the voting-related concepts employed in our research, starting with definitions for mechanisms and interactions. Next, the research problem is formally defined based on these research concepts. Subsequently, the research method and analyzed potential system architectures are introduced and reviewed. The attacker model is also detailed to specify the limits of the research.

3.1 Contribution Concepts

In the context of our work, we define a voting system as follows:

Definition 1 (Voting System) A voting system consists of a tuple $(S, D, L, B, V, A, C, \Omega)$, where $S$ is the voting web server, $D$ is a cast ballot database, $L$ is a set of logs for the server transactions, $B$ is a set of browsers used as agents by voters, $V$ is a set of voters using browsers to vote, $A$ is an authentication
provider, and \( C \) is a set of communication channels between system components. \( \Omega \) is the authority including system administrators, owners and other organizers.

To illustrate, server \( S \) consists of a computer system running appropriate software. A laptop running Tomcat Apache on Windows is an example. An example of database \( D \) is a computer system running dedicated software such as MYSQL that can be utilized to store voting ballots. The logs database \( L \) is a computer system storing records of execution transactions such as connection attempts and failures, authentication steps, timestamped message data exchanges, administrator accesses, and database modifications. The set of browsers \( B \) consist of all computer systems used by users and running agent software able to communicate to server \( S \). Such software is restricted to tools like Chrome, Firefox, or Edge.

The set of voters \( V \) consists of the expected users. In an example related to the studied voting system, this set includes expected users who have business licenses and record numbers, but also potential attackers outside this group.

The authentication providers \( A \) in our case are ABSHER and NAFAD as described in Chapter 2. These providers can be configured to authenticate voters based on certain requirements, such as business licences, home addresses, and voting locations.

Communication channels \( C \) consists of SMS and the Internet over government-approved communication company routers and inter-process communication. A set of authorities \( \Omega \) consist of the actors who have or can influence or tamper directly or indirectly with the processes taking place in server \( S \) and databases \( D \) and \( L \). While
IT administrators are commonly given authority, employers and actors can act as authorities and directly replace or command the administrators without full transparency. In certain realizations, server $S$, logs database $L$, and even some browsers in $B$ share computer systems, whereby the communication channels $C$ between these components consist of an inter-process communication mechanism.

**Concepts Related to Voting Stages** During the process of voting, voters pass through multiple states, such as eligible but not yet registered to authenticated and having cast a ballot.

**Definition 2 (Eligible Voter)** Voter $v$ is eligible in a given election if his or her identity is approved by the authentication provider based on the requirements configured for that election.

For example, the potential eligible voters must be citizens or residents of Saudi Arabia, and the address or business license must conform to the election configuration.

Ballots cast in a sequence of message exchanges or transactions that start by an initial authentication and in which all subsequent interactions are safeguarded by the initial authentication using some security mechanisms or tokens until the procedure is finalized. A sequence of messages forms a communication session and in our case is formalized in the definitions of the words *communication session* and *authenticated voter*.

**Definition 3 (Communication Session)** A *communication session* is a sequence
of message exchanges that occur between voter \( v \) and server \( S \), during which \( v \) is identified by a single interaction with authentication provider \( A \).

In other approaches to system development, each message self-authenticates, and each session is restricted to handle one message. Eligible voters can only cast ballots when they are in the authenticated state.

**Definition 4 (Authenticated Voter)** A voter \( v \) is authenticated if he or she is in a communication session with server \( S \) that is approved by authentication provider \( A \).

In the experiment examples, authenticated voters are established as users with ABSHER credentials and NAFAD accounts, in case malicious users falsely claim eligibility in the given election.

**Concepts Related to Proving System Properties**

**Definition 5 (Property Logic Representation)** A logic representation of a property is a definition of that property using a formal language.

For example, using LTL formulas as a logic representation of a property of a voting system stating that any user presenting \((\text{Signin})\) a valid username \((\text{Valid-Username})\) and valid password \((\text{Valid-Password})\) is \((\rightarrow)\) eventually \((\Diamond)\) authenticated successfully \((\text{Verified})\) through accepted credentials by authentication provider \( A \) is represented as follows:

Formula: \( \Box (\text{Signin} \ (\text{Valid-Username}, \ \text{Valid-Password}) \ \rightarrow \ \Diamond \ \text{Verified} \ (\text{Valid-Username}, \ \text{Valid-Password})) \)
Figure 3.1: Sample Property Logic Representation with Büchi Automata

As an alternative, a logic representation for this formula is the Büchi automata as shown in Figure 3.1.

**Definition 6 (Security Requirements)**  Security requirements of a system consist of a set of property logic representations that together can be used to prove the expected behavior of the system given selected assumptions and an attacker model.

Later, we introduce the expected behaviors and attacker model. Attackers are various capabilities, and due to costs, a system may not be designed to resist attackers who are unlikely or unexpectedly strong. For instance, an attacker model is defined as a set of assumptions that help to quantify the strength of a given voting system realization.

**Definition 7 (Attacker Model)**  Attackers model is a set of restrictions on what actions an attacker can perform against a given system.

For example, an attacker model specifies whether attackers can or cannot eavesdrop on given secure communication channels or whether they can access or actively create or modify messages, system processes, or database and log records.
Concepts Related to the Voting Mechanisms  Voting mechanisms or systems, which refer to sets of rules used for conducting elections, include a method of counting the ballots and computing the winner. For example, in an approval voting system, voters choose more than one candidate, and a winner is the candidate who receives maximal approval or the largest number of votes. In a plurality system, voters choose one candidate only, and a winner is the candidate who receives the largest number of votes.

Definition 8 (Single Casting Mechanism) A voting method is said to be single casting if each voter \( V \) is allowed to vote only one time and only for one specific candidate \( C \) among the listed candidates.

In the United States presidential election round, voters \( V \) have one attempt to cast their votes for one candidate \( C \), qualifying them as single-casting mechanisms.

Definition 9 (Limited Casting Mechanism) A voting method is said to be limited casting if each voter \( V \) is allowed to cast only exactly \( N \) ballots, each of them for a different candidate from the candidates list.

For example, in one of the mechanisms we analyzed, voters can cast exactly three approval ballots for three different candidates from the candidates list.

Definition 10 (Unlimited Casting Mechanism) A voting method is said to be unlimited casting if each voter \( V \) is allowed to cast as many ballots as he wishes, each of them for a different candidate from the candidates list.
For example, with standard approval voting as well as with Borda preferential elections [41, 6], voters can cast a separate ballot for each candidate on the list.

### 3.2 Research Problem

As a way to interact with businesses and citizens and to provide services, the electronic government model becomes prevalent. Hence, electronic voting systems and particularly online voting systems are important tools used in information and communications technology (ICT) to simplify lives and provide decision-making assistance. Online voting systems must be designed to resist the use of deceptive conducts and also to be transparent enough to ensure that candidates and voters trust the outcomes of the elections. In this type of system, verifiability plays an important role for social stability and can be maintained if suitable security and authentication measures are applied appropriately.

The addressed problem is how to design from scratch a verifiable voting system rather than how to verifiability after the design is ready. Verifiability of properties is applied not only to software implementation but also to the principles of involved voting mechanisms. In addition, electoral systems properties are considered performance indicators of a voting system. When presented with the same underlying voter preferences, different voting system constraints produces different results.

The following points summarize a proposed approach for checking voting system verifiability:
• Build a system model in which Büchi automata is used as a nondeterministic finite state to model system actions.

• Specify the desired properties in which linear temporal logic is used as formal logic to form logical formulas.

• Check whether the defined properties satisfy the system model or not by using an algorithm.

Previous researchers have used the approach for analyzing already implemented systems. We investigated the possibility of integrating the checks as steps during system development.

3.3 The Proposed Approach

Logic representations of properties are used to verify security requirements of multi-agent system processes, in particular, in the frame of voting systems. We propose a way to verify security requirements of critical multi-agent system processes by using logic representations and automatic reasoning. The typical multi-agent system considered in our work is an election system with agents representing their users and aiming to ensure security. Relevant processes are authentication, voting, re-voting, and election verification. The security requirements commonly addressed in such a voting system are no users can vote unless their credentials are authenticated, no invalid votes are counted, no votes are counted twice, and all valid votes are eventually
counted. We show a model of such security requirements by using system liveness properties and exemplify their verification on a real system that we implemented for this purpose. Moreover, we analyze and show the implications of required limitations on common voting system criteria for the three different versions of voting restrictions on the election website proposed for the Saudi e-government system.

Distinct mechanisms satisfy different requirements, and no particular method satisfies all of them. In the presented analysis, we prove that one of the new proposed voting schemes, a version of approval voting in which the size of the ballot is bounded, a set of the properties of approval voting are lost, in particular, Pareto efficiency, Condorcet criteria winner support, a Condorcet criteria loser, and independence to irrelevant alternatives.

3.3.1 Designing for Verifiability

The verifiability of the voting system is integrated at a formal level from the design stage, and the verifiability requirements are then continuously refined and propagated to implementation levels as tested by extension to code analysis as the system is developed.

The approach followed uses formal verification and logic representation to check security requirements with respect to the behavior and functionality of the main internal processes in a voting system. The method consists of three phases: modeling, which refers to designing both the internal system processes and desirable security
requirements; running, which consists of using a model checking technique to verify the validity of the properties of the model and eventually to find any counterexample if one exists; and analysis, which uses the results of the running phase to either accept the design or to change it and return to phase one.

In terms of the criterion, the approach followed analyzes election system criteria with respect to some restrictions applied in a studied voting system. The studied versions are methods supported by the proposed system for the e-government of Saudi Arabia: single, limited, and unlimited casting.

Basically, we ran three checks respectively as mentioned in the following points:

- In the voting choice function level, we analyzed single, limited, and unlimited casting mechanisms and showed the implications of some required limitations.

- In the formal specification level, we utilized LTL to specify security requirements with respect to the behavior and functionality of the main internal processes in a voting system and then translate them to an automaton.

- In the implementation level, we created the source code with respect to the predefined security requirements and then conducted the analysis which requires extracting the input from the source code and feeding it into the automaton for program execution checking.
3.3.2 Experiment Overview

In order to evaluate the proposed method’s potential for verifying security requirements, we implement a realistic voting system as a reference. The main processes supported by this voting system are authentication, voting, and verification of the election. We modeled these processes individually. Also, to model the security requirements, we started by defining them unambiguously in a natural language. Based on the definitions of the requirements, LTL specifications were formulated. All LTL formulas obtained were converted to nondeterministic Büchi automata models [35, 14], namely, models based on nondeterministic finite state machines that accept sequences with infinitely occurring states respecting the acceptance condition with transitions of the inputs. We observed that security requirements could be modeled with liveness properties that map easily in the acceptance conditions.

In terms of testing the online system criterion, the studied system versions were divided into three versions. The first version, which permits voters to vote for one unique candidate, corresponds to the common single plurality vote mechanism. The second version, which permits voters to vote for three unique candidates at a time, is a new method called approval voting with a bounded ballot size. The third version, which permits voters to vote for all candidates, corresponds to the common approval voting mechanism.
3.4 System Architecture

In order to generate a proof of concept and to validate our proposal, a full election system was designed and implemented. As proposed, the design integrates verifiability in early stages. After the election mechanisms were analyzed for general voting properties and accepted, the system architecture was built as described in Figure 3.2.

One of the most visible components of the studied voting system is the Verifiable Casting Voting System (VCVS) server, which accepts connections over the network from voters and administrators and checks their credentials or redirect them for credential acquisition. The server also maintains authenticated sessions in which voters
cast and verify their ballots while administrators audit the process. The server is also in charge of coordinating ballot databases and logs for verifying processes. In the diagram in Figure 3.2, the server denoted by a blue rectangle is connected with user browsers, databases, logs, and identity providers via secure channels depicted by tubes that cross the clouds.

The VCVS ballot database stores, collects, and processes both actual ballot data and metadata, and contains a voting system database and an NAFAD identity database. The voting system database consists of candidates, users, and ballot information. The NAFAD database consists of information about users who are authenticated successfully.

The logs database is a tamper-proof database that stores a record of each transaction with the purpose of supporting auditing of the system after the fact by third parties or administrators and generating verification proof for the correctness of past voting processes.

As system clients, users and administrators must have access credentials. Administrators’ credentials are created by the VCVS administrators themselves, stored in the database, and then distributed to all administrators to access the administrative pages. Users must use their ABSHER credentials in order to access the VCVS system and cast their votes.

An identity provider like NAFAD handles the authentication requests coming from the integrated system (VCVS). Then, it returns the request validation to users
to authorize their access.

The ABSHER database handles the validation part of the requests coming from NAFAD and verifies their existence. Otherwise, the request is rejected.

Users signs in to the VCVS system, using ABSHER credentials, and then the VCVS server redirects users to NAFAD, which in turn sends requests to the ABSHER database for validation. The validation results are transmitted to the users, who can access the system if the request is approved. Otherwise, they receives a rejection message. Once the users gain access to the system in an authenticated session, they can choose a candidate and cast their vote. Afterwards, a confirmation message is generated and rendered to the users. Logs are maintained in the VCVS logs database.

The Internet is considered an insecure channel that may traverse areas with routers controlled by attackers. The Intranet is considered a secure channel.

Part of the verifiability integration at this design level, offline verifiability is ensured by tamper-proof logs and the security of the Intranet channel, assuming security of the VCVS server and appropriate log structure.

Online verifiability of the fact that each process conforms to requirements at the execution time is ensured by the maintenance of invariants defined with LTL and tested dynamically.

The request initiated by voters via their users’ browsers must include valid usernames and passwords, which are ABSHER credentials. Otherwise, the request is rejected. The VCVS server redirects voters to NAFAD for validation. NAFAD validates
requests and authorizes only the requests containing ABSHER credentials checked against the ABSHER database. NAFAD responds to the users’ browsers, which either display the verification page to the users if the authentication were successful or an error message. At the same time, NAFAD sends the users’ tokens and the relevant parameters to the server. The VSVC server authenticates the entered tokens in the displayed verification page, and based on the result, the server grants access to the users. When the users access the VCVS and cast their votes, the VCVS interacts with the VCVS (MySQL) database to store data, and the VCVS are notified if the data is successfully stored, which means the votes are correctly counted. Then, the confirmation message request is sent from the VCVS server to the users’ browsers. The VCVS server also sends a record of each transaction to the Logs database.

3.5 Attacker Model

We assumed that the attackers cannot read or modify the data on the server and cannot control the behavior of the software on the server. In practice this assumption is enforced by up-to-date server software, using two-factor authentication as a security parameter and limits on authentication attempts.

We assumed that the attackers cannot eavesdrop on communication between clients and the server. This assumption is commonly achieved by end-to-end encryption that attackers cannot break.

We assumed that the attackers cannot control the user agent behavior and cannot
read or modify data on the client terminals of the voters. This assumption is enforced by supporting voter logins from multiple terminals that can verify each other.

We assumed that the attackers cannot impersonate clients, which means that the attackers cannot claim false identities. In our case, this assumption is enforced by exploiting the state-provided ABSHER identity database. An attack on identities in our system requires an attack on NAFAD.

We assumed that the attackers cannot gain the user ABSHER credentials by taking control of the VCVS server. This assumption is enforced by the fact that the VCVS redirects users to NAFAD when the access request is initiated, and users never provide their credentials other than to NAFAD as supported by the NAFAD API.

Attackers cannot make computer hardware fail because the Internet and SMS communication are reliable and private, and human participants are physically unconstrained and actively participate during the whole process.
Chapter 4

Analysis and Results

In this chapter, we report on the steps needed for integration of verifiability in design and development. We start by detailing the steps in design for verifiability and for LTL models of voting processes properties, followed by analyzing the details of the implications of requirement limitations, in which the models of the studied voting schemes are specified. Subsequently, we overview logic models of properties to be verified, and we close by showing how proof of these models is integrated into development.

4.1 Steps in Design for Verifiability

For the design phase, high level models were proposed first. The models were verified experimentally. Subsequently, formal proofs are generated, and the results are analyzed for design refinement.
### 4.1.1 LTL Models of Voting Processes Properties

In our approach, the internal processes are modeled individually: the authentication, validation, and confirmation.

We used temporal descriptions in the security requirements, such as the voters should be successfully authenticated as a precondition to casting votes, votes should be eventually counted after being cast, and voters should eventually receive confirmation messages after voting. We utilize words like *precondition*, *before*, and *after* to show that some actions happen in sequence.

We noted that types of temporal descriptions occurring in our cases of interest can be formulated as liveness properties, which state that a given predicate eventually holds at moments past some specific time in the future.

To illustrate the approach, we used linear temporal logic to formulate liveness properties describing system security requirements in the following formulas. The fact that system failures, such as power, Internet, or computer, result in an interruption of services identifiable to the users is not represented in the following expressions and are treated separately:

- **Formula 1:** \( \Box (\text{Signin (Valid-Username, Valid-Password)} \rightarrow \Diamond \text{Verified (Valid-Username, Valid-Password)}) \)

- **Formula 2:** \( \Box (\text{Verified (Valid-Username, Valid-Password)} \rightarrow \Diamond \text{Receive (Unique-Code)}) \)
• Formula 3: $\Box \text{(Cast (vote)} \rightarrow \Diamond (\text{Counted (vote)} \land \text{Confirmed(vote))})$

The current working assumptions that are included as limitations in the attacker model of this research step state that computer hardware does not fail, Internet and SMS communication is reliable and private, and human participants are physically unconstrained and actively participate during the whole process. Relaxation of these assumptions are the subject of future work.

The first formula sample LTL statement specifies that whenever a sign-in action happens for valid users, then sometime later, a verification action takes place and qualifies them. Furthermore, according to the second statement, whenever users are verified, then sometime later, they receive a unique code as a token for casting a vote. The third statement indicates that a confirmation message must be generated, and the vote is counted whenever cast.

Tools such as the LTL2BA is able to create a Büchi automaton from such LTL properties. The automaton generated in this way is specified in PROMELA.

The call graph of the source code is built and focuses only on function calls, and then execution paths of the implementation are also provided as the traversal from root to the end nodes.

For example, a verification piece of code corresponding to the automata generated for the first LTL property for the property checking is shown in Algorithm 1.

The function names are \{signin, getVerified, signin-completion\}.

The Büchi automata is fed by the set of the generated paths as inputs. Then, if
the paths from the input set end in the accepting state, which is the \textit{init} state, the property is achieved. In contrast, if the paths end up in a non-accepting state 1, then the property is not guaranteed.

### 4.1.2 Experimental Validation of the Design Model Process

**Step**

An experiment for the proposed processes is based on a voting system developed for Saudi Arabia’s business performance contests organized under the public administration.

Formulas $\phi$ are translated into a Büchi automata $B \phi$ as shown in Figures 4.1, 4.2, and 4.3 generated automatically by LTL2BA.

![First Automata](image)

**Figure 4.1: First Automata**

The function name $!\text{signin}$ represents any event except the sign-in process, and
( || ) is utilized not to match either the functions signin or getVerified.

The function name getVerified represents any event except getVerified, and ( || ) is utilized not to match either the functions getVerified or receiveUniqueCode.

The function name castVote means any event except castVote, and ( || ) is utilized not to match either the functions castVote or sendConfirm.

Each examples includes two states. As named by the LTL2BA tool generating the automata, the init state is the accepting state while both states of the automaton are initial states. The transition labels are the predicate names. Label 1 matches any predicate name.

For each Büchi automata, a PROMELA code is generated as shown below:
First Büchi Automata Code

never {

/* □ (signin → ◇ getVerified) */

accept-init : /* init */

if

:: (!signin) | | (getVerified) → goto accept-init

:: (1) → goto T0-S2

fi;

T0-S2 : /* 1 */

if

:: (getVerified) → goto accept-init

:: (1) → goto T0-S2

fi;

}

Second Büchi Automata Code

never {

/* □ (getVerified → ◇ receiveUniqueCode) */

accept-init : /* init */

if

:: (!getVerified) | | (receiveUniqueCode) → goto accept-init

:: (1) → goto T0-S2
Third Büchi Automata Code

never {

/* □ (castVote → ◇ sendConfirm) */
accept-init : /* init */

if

:: (!castVote) || (sendConfirm) → goto accept-init
:: (1) → goto T0-S2
fi;

T0-S2 : /* 1 */

if

:: (sendConfirm) → goto accept-init
:: (1) → goto T0-S2
fi;

}
Since we integrated verifiability within the design phase, static analysis was the appropriate method to test a studied program, in which the source code was analyzed simply without being run. The integration removes the need for creating and using test cases and gives opportunities to enhance and develop the code multiple times at the design level as needed.

We conducted our test by using the Doxygen source code analyzer to produce the needed input for the automaton. Each path in the generated abstract model (call graph) is considered as an input path for the automaton for property checking. Also, we utilized the LTL2BA tool to produce the Büchi automata from LTL properties. The syntax of the Büchi automata was provided also in PROMELA. We built the automaton based on the generated PROMELA code as shown in Algorithm 1, and we checked the properties by feeding the paths as inputs. For example, a fragment of the function call graph corresponding to the authentication process in the source code is shown in Figure 4.4.

An extracted path from the source code of the authentication process is \{signin, getVerified, signin-completion\}. A sample input for the first Büchi automaton is the set of the paths \{\{signin, getVerified, signin-completion, getVerified\}; \{signin, getVerified, signin-completion, getVerified\}; \{signin, getVerified, signin-completion, getVerified, getVerified\}\}. As noted, all input sets end in \textit{init}, which is the accepting state; therefore, the first property is achieved with this sample.
4.1.3 Formal Proof and Analysis of the Properties

The case study of the voting system consists of three main internal processes: the authentication, validation, and confirmation. The processes are formally defined in LTL.

Theorem 1 (Abstraction Theorem [44]) If the specification policy $\phi$ is followed in program $P$ up to the abstraction level, i.e. only function calls are considered, then paths generated from program $P$ are accepted by the automaton $A_{\phi}$.

Theorem 2 (Correctness Theorem of the Translation [14]) The Büchi automaton $A_{\phi}$ accepts precisely the models of $\phi$, i.e. $\mathcal{L}(A_{\phi}) = \mathcal{L}(\phi)$,

where $\mathcal{L}(A)$ refers to the accepted language of the automaton.
The authentication process is defined as the following:

\[ (\text{Signin} (\text{Valid-Username}, \text{Valid-Password}) \rightarrow \diamond \text{Verified} ) \]
\[ (\text{Valid-Username}, \text{Valid-Password}) \lor \text{Failure(Username)}) \], simplified based on the attacker model as

\[ (\text{Signin} (\text{Valid-Username}, \text{Valid-Password}) \rightarrow \diamond \text{Verified} (\text{Valid-Username}, \text{Valid-Password}), \]

where the sign-in request must include a valid username and password based on our definition, which implies valid ABSHER credentials verified by NAFAD.

The validation process is defined as the following:

\[ (\text{Verified} (\text{Valid-Username}, \text{Valid-Password}) \rightarrow \diamond \text{Receive} \]
\[ (\text{Unique-Code}) \lor \text{Failure((Unique-Code))), simplified based on the attacker model as \]

\[ (\text{Verified} (\text{Valid-Username}, \text{Valid-Password}) \rightarrow \diamond \text{Receive} (\text{Unique-Code}). \]

Users who are verified by NAFAD must receive a validation code to gain access to the system; if the verification fails, they are rejected.

The confirmation process is defined as as the following:

\[ (\text{Cast} (\text{vote}) \rightarrow \diamond (\text{Counted} (\text{vote}) \land \text{Confirmed(vote)})) \lor \]
\[ \text{Failure(vote)), simplified based on the attacker model as \]

\[ (\text{Cast} (\text{vote}) \rightarrow \diamond (\text{Counted} (\text{vote}) \land \text{Confirmed(vote))}). \]

Ballots must be counted after users cast their votes; once the votes are counted, the voters receive confirmation messages to indicate that their votes were counted
successfully. Otherwise, they receive a failure message.

Since all the \( LTL \) formulas represent system liveness properties with constructs of the following type:

\[ \Box \rightarrow \Diamond. \]

We show in detail in this report only the analysis of the first \( LTL \) among them, namely, the authentication process. The remaining formulas are handled with a similar methodology of analysis.

For any \( LTL \) formula \( \phi \), such as the one representing the authentication process, we generated \( B\phi \) by translating the \( LTL \) formula to Büchi automata. The automaton is fed with paths extracted from the source code.

Based on Theorem 1, our properties \( \phi \) complied with the source code up to the abstract model level, and all the paths generated from the source code are accepted by \( B\phi \). Also, based on Theorem 2, the accepted input for automaton \( B\phi \) are the same as the language that models the formula \( \phi \). In the experiment, all the paths ended in the accepting state, and the property was achieved as explained in Section 4.1.2. Details are given in Appendix B.

\section{4.2 Implications of Required Limitations}

The exemplification of high-level voting system design and analysis was conducted on three models of voting schemes, following the concepts of single-member district rule, multi-member districts, and bounded ballot approval voting, respectively. To
our knowledge, bounded ballot approval voting was analyzed for the first time, and we found the implications of the bounds to be remarkable. Subsequently, formal proof was generated, and the results were analyzed for scheme design improvement.

4.2.1 Models of Voting Schemes

Three analyzed voting schemes were implemented in the voting system in Saudi Arabia’s business performance contests:

- Single casting voting scheme
- Limited casting voting scheme
- Unlimited casting voting scheme

**Single Casting Voting Scheme** This model follows the idea of single-member district rule [3], in which voters are allowed to vote for one candidate only. In our model, voters are authenticated through the system, which verifies the voters by sending a verification code to allow access. Then, voters interact with the system and have only one chance to vote for one candidate. After voting, voters receive a ballot receipt that confirms their votes were counted successfully.

**Limited Casting Voting Scheme** This model follows the idea of multi-member district [28], where voters are allowed to vote three times for three different candidates. Voters log in to the system, which verifies the voters by sending them a verification
code to allow access. Voters have three chances to select no more than three different candidates, and they cannot vote twice for the same candidate. The system prevents voters from voting after using all three opportunities. After each time voters vote, they receive a ballot receipt indicating their votes were counted successfully and clarify the number of opportunities left to vote.

**Unlimited Casting Voting Scheme**  This model follows the idea of approval voting [9], in which voters are allowed to vote for any subset of the candidates. The voters log in to the system, which verifies their identities by sending a verification code to allow them access. After each time voters vote, they receive a ballot receipt that proves their vote were counted successfully.

### 4.2.2 Formal Proof and Analysis

Voter ballots can be bounded to a size of one or three choices, or they can be unbounded. Assuming that we have an election with 100 voters and four candidates $A, B, C,$ and $D$, in the case of the mechanism of voting with ballots of unbounded size, 60 voters prefer candidates $A, B, C,$ and $D$ equally while 40 voters prefer only $D$ and select $D$ as their preferred candidate. Therefore, the preferences in an unbounded approval voting are as follows:

<table>
<thead>
<tr>
<th>Ballot</th>
<th>Voters</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,B,C,D</td>
<td>60</td>
</tr>
<tr>
<td>D</td>
<td>40</td>
</tr>
</tbody>
</table>
Using the same scenario but a different mechanism of voting with ballots bounded to a maximum of three choices, 60 voters consider all candidates equally, but because they can only approve three choices, they select alphabetically: A, B, C. Also, they like D equally, but because no more opportunities remain to vote, they drop D. The 40 voters who like only D vote for him as before. Therefore, with a bounded approval voting with a limit of three votes, the actual ballots are as follows:

<table>
<thead>
<tr>
<th>Ballot</th>
<th>Voters</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C</td>
<td>60</td>
</tr>
<tr>
<td>D</td>
<td>40</td>
</tr>
</tbody>
</table>

In the bounded approval voting scheme, D loses, even though the candidate has the most supporters and is the Pareto winner. As such, the version with a bounded ballot size also fails to guarantee an election of the Condorcet criterion winner, the Condorcet criteria loser, and the independence to irrelevant alternatives; since C is an irrelevant loser, the occurrence of C makes D lose.
Chapter 5

Conclusions

Electronic voting processes are diverse as defined by multiple protocols. Some electronic voting processes use kiosk devices, namely, dedicated computers in polling stations while others solely rely on online voting websites, generally classified as uncontrolled environments, which means they are not physically supervised to preclude physical pressure on voters, and they also typically depend on the Internet for communication. Using the Internet in such systems minimizes costs and are expected to increase users’ participation by providing more accessibility to computer savvy participants, allowing them to vote from anywhere. At the same time, the Internet is considered an insecure environment, which leads to many challenges, such as loss of privacy due to hacking or wiretapping, tampering with the process or with results, or stealing or using fake identities.

The first motivation point behind our research was the digital transformation program, which is one of the important components of Saudi Vision 2030. The target of
this program is to create a digital society, economy, and homeland based on establishing digital platforms to enrich interaction and effective community participation, to improve the experience of using the online platforms, and to raise the technological knowledge of the citizens and residents of the kingdom.

The second motivation point was the inspiring history of electronic governance of Estonia, the first country in the world that implemented an online voting system throughout the nation. Their successful implementation and the enthusiasm of their citizens to adopt this method suggest that such systems have much potential to be popular mechanisms for elections.

The significant recent promotion of online voting systems, mainly in the context of the trendy blockchain technology, raised additional interest for this application of the Internet.

The last motivation point was the failure in the Iowa Democratic caucus, a critical event when the application designed and utilized for the electoral process collapsed while the backup center failed to work to the point where the local officials became unable to record the results effectively. Long security and quality checks were completed on these results to examine the authenticity with the actual figures, which further delayed the publication of results.

These four motivation points drove our focus to the principal implementation of an online voting system. We highlighted the following: authentication by using an identity authentication management (IAM) system, verification of the security
requirements of the internal system processes by using an automatic system, and implications of required limitations of a bounded ballot approval voting.

As a background of this research, we introduced related studies regarding voting about common solutions, methods, and ideas with previous techniques and technology, on which we also built our work. We reviewed the recent trends on voting with blockchain technology and introduced the blockchain mechanism and concepts, and we showed the difference between proof-of-work and proof-of-stake mechanisms.

Subsequently, we showed a set of well-known cryptographic election protocols, such as the Benaloh election protocol and the Merritt’s election protocol along with their tallying processes. Also, we introduced one of the verifiability mechanisms of online voting systems based on software analysis, including the analysis of implementations of election technologies and epistemic policies for voting system. Besides, we showed tools and databases available for voter authentication at national level in Saudi Arabia, such as the ABSHER database and the NAFAD access portal to ABSHER.

We introduced and showed the voting-related concepts employed in our research. In the context of our work, we defined the contribution-related concepts, such as the voting systems, eligible voters, communication sessions, and authenticated voters. Also, we defined the concepts related to proving system properties, such as property logic representation, security requirements, and the attacker model. In terms of voting mechanisms, we defined the single casting mechanism, the limited casting mechanism,
and the unlimited casting mechanism.

The addressed problem is how to design from scratch a voting system that is verifiable rather than to prove verifiability after the design is ready. Verifiability of properties are applied not only to software implementation but also to the principles of the involved voting mechanisms. In addition, electoral system properties are considered performance indicators of voting systems. When presented with the same underlying voter preferences, different voting system constraints produce different results.

We integrated the verifiability of the voting system at a formal level from the design stage, and then we continuously refined and propagated the verifiability requirements to implementation levels as tested by the extension to code analysis as the system was developed.

Consequently, we used formal verification and logic representation to check security requirements with respect to the behavior and functionality of the main internal processes in a voting system. The processes supported by the voting system are authentication, voting, and verification of the election. We showed how to model the processes individually. We formulated the processes by using LTL. Then, we converted all the LTL formulas to nondeterministic Büchi automaton models. We observed that security requirements can be modeled with liveness properties that map easily into such acceptance conditions.

As another step of the proposed approach to voting system design, we analyzed
the selected election system criteria. In our case study, the approach was tested with respect to some restrictions applied in the suggested voting system. The studied versions were single, limited, and unlimited casting methods. The first version permits voters to vote for one unique candidate, corresponding to the common single plurality vote mechanism. The second version, which permits voters to vote for three unique candidates at a time, is a new method that can be described as approval voting with a bounded ballot size. The third version permits voters to vote for all candidates, corresponding to the common approval voting mechanism.

In order to validate our proposed method and produce a proof of concept, we designed and implemented a full election system. The design integrates verifiability in the early stages of the system development. First, we analyzed the defined election mechanisms. We built the case study system architecture and described it in details as shown in Section 3.4. In brief, the system architecture consists of a user browser, a VCVS server, a VCVS database, a logs database, the identity provider (NAFAD) system, the ABSHER database, and Internet and Intranet connections. An attacker model was proposed by extracting the multiple assumptions needed for the designed system to resist malicious activities initiated by attackers.

In terms of analysis and results, we detailed the necessary steps for integration of the verifiability during the design and development stage. We showed the steps of the designs of the verifiability and the LTL models of the voting process properties. We specified the internal processes by using LTL.
The first formula sample LTL statement specifies that whenever a sign-in action happens for valid users, then sometime later, a verification action takes place and qualifies them. Furthermore, according to the second statement, whenever users are verified, then sometime later, they receive a unique code as a token for casting a vote. The third statement indicates that a confirmation message must be generated, and the vote is counted whenever cast.

After that, tools such as the LTL2BA is able to create a Büchi automaton from such LTL properties. The automaton generated in this way is specified in PROMELA. The call graph of the source code is built and focuses only on function calls, and then execution paths of the implementation are also provided as the traversal from root to the end nodes.

For example, a verification piece of code corresponding to the automata generated for the first LTL property for the property checking is shown in Algorithm 1.

An extracted path from the source code of the authentication process is \{
\text{signin, getVerified, signin-completion}\}. A sample input for the first Büchi automaton is the set of the paths \{\text{signin, getVerified, signin-completion, getVerified}%; \text{signin, getVerified, signin-completion, getVerified}%; \text{signin, getVerified, signin-completion, getVerified, getVerified, getVerified} \}. As noted, all input sets end in \text{init}, which is the accepting state; therefore, the first property is achieved with this sample.

We studied the implications of required limitations by the exemplification of high-level voting system design and analysis was conducted on three models of voting
schemes, following the concepts of single-member district rule, multi-member districts, and bounded ballot approval voting, respectively.

In the experiment analysis, we proved that one of the new proposed voting schemes, a version of approval voting in which the size of the ballot is bounded, a set of the properties of approval voting are lost, in particular, Pareto efficiency, Condorcet criteria winner support, a Condorcet criteria loser, and independence to irrelevant alternatives.

In summary, online voting processes are shown to be specified, checked, and improved with formal verification methods. We proposed a method to verify security requirements of critical multi-agent system processes by using logic representations and automatic reasoning at the design level. The sample multi-agent system considered in our research was an election system with agents representing their users and aiming to ensure security. Relevant processes were authentication, voting, re-voting, and election verification. The security requirements commonly addressed in such a voting system are no users can vote unless their credentials are authenticated, no invalid votes are counted, no votes are counted twice, and all valid votes are eventually counted. We show a model of such security requirements by using system liveness properties and exemplify their verification on a real system that we implemented for this purpose.

Previous work in the same area of research [44] addressed the verifiability of the property of an electronic voting system via code scanning to check the predefined
evidence flow. The proposed technique was applied to the vVote system, which was utilized in the state elections of Victoria, Australia.

The novel contributions are that we covered the authentication part by using an IAM system called NAFAD, which identifies users’ identities by verifying them through the ABSHER system. We showed that the security requirements of the internal multi-agents system processes can be tested based on liveness system properties when they are modeled with LTL representations. We showed that the corresponding properties of the security requirements are verified with an automatic system. Moreover, we proposed the integration of the verifiability at the design level by defining all the critical system’s processes, checking and verifying them as part of the system development.

Also, we proved that one of the new voting schemes in the proposed election website, a version of approval voting where the size of the ballot is bounded, a set of the properties of Approval Voting are lost, in particular Pareto efficiency, Condorcet criteria winner support, Condorcet criteria loser, and independence to irrelevant alternatives. To our knowledge, bounded ballot approval voting was analyzed for the first time by us, and we found the implications of the bounds to be remarkable. Subsequently, formal proof was generated, and the results were analyzed for scheme design improvement.

In term of limitations, the proposed verifiability designing and proving method consists of three phases: modeling, which refers to designing both the internal system
processes and desirable security requirements; running, which consists of using a model checking technique to verify the validity of the properties of the model and eventually to find any counterexample if one exists; and analysis, which uses the results of the running phase to either accept the design or to change it and return to phase one.

We performed the second and third phases partially manually using the Doxygen code analyzer. We found a tool called Coverity to build the automaton from the PROMELA file but could not employ the tool due to the access limitation available only under the insufficient academic license program. We had to create the automaton in part manually by writing a piece of code for the property checking process according to the produced PROMELA code, and then we checked the properties by feeding the paths as inputs, which is supported by Theorem 1.

In future work, a fully automated model checker can be utilized to conduct property checking. In the process of generating the abstract model using a source code analyzer, function calls can be extended to include variables and data. Also, further efforts can address other security implications of the system design, such as buffer overflow, SQL injections, and bugs. Moreover, logs can be formalized, encrypted, checked, and verified using formal verification methods, such as timestamped log entries, consistent logging of important actions, and controlled access to logs database.
Appendix A

Experiments Details and Code

The proposed approach to verifiability integration in design and implementation was validated using an experiment with a case study in the development of a voting system for Saudi Arabia’s business performance contests, namely the Verifiable Casting Voting System (VCVS). Basically the VCVS system has five modules which contain multiple classes as following:

1. The Voting.system.dev.bean package is composed of:

   (a) Candidate.java class whose objects represents the candidates. The candidate has an idCandidate as a primary key, first and last name, candidate creation date, and numberOfVotes which represents the total received votes. Also, idCandidateGroup represents the group of all the candidates.

   (b) CandidateGroup.java class is responsible to regroup all the added candidates. It has statusCandidateGroup to track the status of the voting phase,
and dateCreationCandidateGroup to represent the date of candidate group creation.

(c) User.java class is responsible to represent a system user. The user is characterized by an userId which is a primary key, first and last name, email, mobile phone, login of his credentials, password which would be stored as hashes in the DB. Also, creationDate which refers to the date of creation in the system. The membertypeUser refers to the user type whether the user is a normal user or a system admin, and numVotesAllowed refers to the number of votes that the voter can make at any specific voting phase.

(d) Vote.java class is made to track the votes that made to a specific candidate. It has voteId as an ID of the class, userIdVote refers to the voter who has made the vote, canIdVote refers to the candidate who has received the vote, and dateVote refers to the date of when the vote has been made. Also, statusPhaseVote shows that the vote has been made on the first, second or last voting phase.

(e) VSConnect.java class is responsible for making the connection between the application and MySQL database. The Vue class is created to select only the columns that needed form Candidate.java class, user.java class and vote.java class. Subsequently, VueCandidate.java class, VueUser.java class, and VueVote.java class. In VueVote.java class, firstName and lastName refer to the first and last names of the voters who have made the
votes. DateVote represents the creation date of the vote, and statusPhaseVote shows the voting phase whether if it first, second or last phase. Also, firstNameCandidate and lastNameCandidate refer to the name of the Candidate who has received that specific vote.

2. The Voting.System.dev.common package is composed of:

   (a) CryptWithMD5 class which responsible to encrypt the voters’ passwords in the database.

3. The Voting.System.dev.controller package is composed of Servlet controllers as follow:

   (a) The CandidateController servlet which is made for the admin to modify, add, update or delete the candidates.

   (b) LoginController.java is responsible for the voters’ login, whenever a user called http://.../login, the LoginController.java code will be executed. The GET and the POST methods can be distinguished. The GET method means when a voter call the servlet without passing parameters, then the system login interface returned to the user.

   The POST method means when the user called the servlet and pass parameters such as username and password, then the user can access the system.
(c) LogoutController.java, this **servlet** is called when the voters is logging out.

(d) UserVotesController.java **servlet** is executed when the admin want to show all the votes that have been made by the voters at any voting phase.

(e) VotingControllerAdmin.java **servlet** is made for the admin to change the voting phase status. We have 0 as a status of the CandidateGroup when the candidates have been added but the voting phase has not started yet. 1 means that the first voting phase is started. 2 means that the first voting phase is finished. 3 means that the second voting phase is started. 4 means that the second voting phase is finished. 5 means that the third and last voting phase is started. 6 means that the third and last voting phase is finished.

(f) VotingControllerUser.java **servlet** is executed when the user is authenticated and accessed the system to choose a candidate and cast his/her vote. All the candidates would be showed up on the system interface. In order to proceed, the voter may choose the preferred candidate. Then, cast his/her vote simply by clicking on a vote button.

4. The Voting.System.dev.dao package contains all the methods needed in the controllers (**servlets**), as listed below: CandidateDAO.java class contains some methods such as find method to search about a specific candidate by the ID number. The findAll method to select all the candidates. Create, update and
delete methods to create, update or delete a specific candidate respectively.

CandidateGroupeDAO.java class contains findALL method as needed in CandidateGroup class. Also, create and update methods to create and update the status of this group of candidates.

UserDAO.java class contains find method to return the specific user who is authenticated. findALL method to return all users who are authenticated. Also, findAllPhase to show all voters on all phases. Update method to change the number of votes allowed for each user of the system when the voting phase change.

VoteDAO.java class contains the findAll to return all the votes that were made by the voter. It also defines the Create method to create a vote whenever a user make a vote on a specific candidate, and the Delete method to delete a specific Vote.

5. Voting.System.dev.impl package contains classes inherited from the DAO class to implement the methods that we need. So, it pass the connection to the DAO class to access the methods that we specified there and that we called from the controller. It contains CandidateGroupImp.java class, CandidateImp.java class, UserImp.java class, and VoteImp.java class.

**Absher System** is a PHP program simulating the actual ABSHER system with its exact interface for the purpose of testing. It uses the Laravel framework. This
framework is based on CMV system which means controller, module, and view. The (App/http/controller/admin or /user) files define all functions related to admin or user pages. Module (/user.php) file connected to user table in the database, and it responsible to bring the data from the database. The (resources/views/admin or user) files define the front end pages which would be seen by user or admin. The (route/web.php) file defines routes that are for the user or admin web interface. The Absher system is working as follows:

1. Homepage: Route::get('/','HomepageController@index'), calls index function from (App/http/controller/User/HomepageController.php) file to return home page view from (resources/views/users/templates/home.blade.php) file.

2. Register Form: Route::get('/register', 'AuthController@showRegistrationForm'), when the user clicks new user, this route calls showRegistrationForm function from (app/Http/Controllers/Users/AuthController.php) file to return the view of the registration from (resources/views/users/templates/auth/register.blade.php) file to show the empty registration form to the user.

3. Register Form: Route::post('/register', 'AuthController@register') , this route calls register function from (app/Http/Controllers/Users/AuthController.php) file to allow user to post the data and validate what the user entered. After that, calls the user module to save the entered data in the user table in the database, then redirects user to the login page (url('/login).
4. Login Page: Route::get('/login', 'AuthController@showLoginForm'), when the user click login, this route calls showLoginForm function from (app/Http/Controllers/Users/AuthController.php) file to return the view of the login form from (resources/views/users/templates/auth/login.blade.php) file to show the empty login form to the user.

5. Login Page: Route::post('/login', 'AuthController@login'); this route calls login function from (app/Http/Controllers/Users/AuthController.php) file to allow user to post his/her username and password and validates what the user entered, if the data entered has been correctly validated in the database, calls success.login function which responsible to generate random code and send it to the user, then redirect the user to authenticate page (url('/authenticate')). Otherwise, return to the same view and show incorrect password or username message.

6. Authentication Form: Route::get('/authenticate', 'AuthController@showAuthenticationForm'); this route calls showAuthenticationForm function from (app/Http/Controllers/Users/AuthController.php) file to return the view of the authentication form from (resources/views/users/templates/auth/authentication.blade.php) file to show the empty authentication form to the user.

7. Authentication Form: Route::post('/authenticate', 'AuthController@authenticate'); once the user enter the code and
click submit, this route call authenticate function from (app/Http/Controllers/Users/AuthController.php) file to allow user to post the received code and validate through database records, then return the user to homepage. If the user entered a wrong code, return to authentication page and show invalid code message.

8. Resend Authentication Code: Route::post('/resend-authentication-code', 'AuthController@sendAuthenticationCode'); if the user clicks resend icon, this route calls sendAuthenticationCode function from (App/Http/Controllers/Users/AuthController.php) file to return generate-random-code private function to generate the random code to be entered in the authentication form.

9. Logout Page: Route::get('/logout', 'AuthController@logout'); when the user logs out, this route calls logout function from (App/Http/Controllers/Users/AuthController.php) file to redirect the user out of the system (url('/')).

**Nafad Portal** is integrated with the Absher system, serving as an interface for it. It has many functions located at (nafad/app/http/Controllers/AuthController.php) as follows:

1. Public function showLoginForm: to show the login page to the user.

2. Public function login: to allow the user to enter the credentials which should be
an Absher credentials. By entering the username, they going to be saved into array, and a request sent to the following link; ('http://absher.local/api/nafad/login') to check if the credentials is known in the database or not. If yes, then return to the function with user’s information and redirect the user to the authenticate page. If the username or password did not match an entry in the Absher database, then Nafad redirects the user to back and shows an error message.

3. public function sendAuthenticationCode: called when the user press resend code icon.

4. Public function showAuthenticationForm: this function ensures that the user is in the session or the user has authenticated login-status, or the session has a code. If yes, redirect the user to the home page to enter the credentials again, then redirect him to the authentication page to enter the code.

5. Public function send-mail: this function allows the user to enter the generated code and the email address, then send the code to the email.

6. public function success-login: this function is responsible to generate the random code. And record the user who is in the authentication phase with the code in a session. Then send him email containing the code.

7. private function generate-random-code: this is a loop function to generate random code starting from 0 to 9.
8. public function authenticate: this function works after user enter the code and insures if the user really enter code. if the user have wrong code, we save it the false login session because Nafad does not have a database, if the number of false logins is more than 3, a request send to Absher (http://absher.local/api/nafad/block-user) in order to call [block-user] to block him. block-user is a function, its route is located in Absher in Nafad Controller. Moreover, if the code saved in session equals the code entered by user, then remove the code from session and let login login-status be true. Then, it redirects the user to the homepage. if the code is not correct, bring the old false login and save it in the session and increment the counter by 1.

9. public function logout: this function deletes the session and returns the user back to the home page.

Since Nafad is integrated with the Absher system, so it has a controller located in Absher (absher/app/Http/Controllers/Users/NafadController.php) and it has two functions as the following:

1. Public function login: this function handles all the requests coming for Absher and validates that the request has username and password. Then, it checks in the user table located in Absher database if these credentials exist or not. If yes, then return true. Otherwise, return false and show incorrect message.

2. Public function block-user: this function is responsible to block the user by knowing his ID, if the number of failed attempts exceeds 3 times.
The Database of this system is created by using MYSQL. Basically, we have two databases, one for the Absher system and the other one for VCVS. The Absher database contains five tables, as follows: **citizens** table, **failed-jobs**, **migrations**, **password-resets**, and **users**. The **citizens** table contains: identification number, first name, last name, username, photo, fingerprint, address, birthday, and phone number. **Users** table contains: ID, citizens-id which links **citizens** table with **users** table, first name, last name, username, email, email-verified-at to clarify the date, password, identification number, phone number, photo, fingerprint, login-code, status, remember-token, created-at, updated-at, false-login, and is-blocked.
Appendix B

Call Graphs

Figure B.1: First (LTL) Formula Related Functions
Figure B.2: Second (LTL) Formula Related Functions
Figure B.3: Third (LTL) Formula Related Functions
References


[30] Political Reporter In Des Moines For Dailymail.com Nikki Schwab, Senior U.s. Iowa Democratic caucus results in crisis as party says it has found 'INCONSISTENCIES', Feb 2020.


