

نموذج ترخيص

أنا الطالب: اسماء أحمد سمان أمتح الجامعة الأردنية و /
أو من تفوضه ترخيصاً غير حصري دون مقال بنشر و / أو استعمال و / أو استغلال و /
أو ترجمة و / أو تصوير و / أو إعادة إنتاج بأي طريقة كانت سواء ورقية و / أو إلكترونية
أو غير ذلك رسالة الماجستير / الدكتوراه المقدمة من قبلي وعنوانها.

Investigating Parallel Implementations of Electronic
Voting Verification and Tallying processes

وذلك لغايات البحث العلمي و / أو التبادل مع المؤسسات التعليمية والجامعات و / أو لأي
غاية أخرى تراها الجامعة الأردنية مناسبة، وأمتح الجامعة الحق بالترخيص للغير بجميع أو
بعض ما رخصته ليا.

اسم الطالب: اسماء أحمد سمان
التوقيع: [Signature]
التاريخ: 2015/5/19

**INVESTIGATING PARALLEL IMPLEMENTATIONS OF
ELECTRONIC VOTING VERIFICATION AND TALLYING
PROCESSES**

By
Israa Ahmad Saadeh

Supervisor
Dr. Gheith Abandah

**This Thesis was Submitted in Partial Fulfillment of the Requirements for the
Master's Degree of Computer Engineering and Network**

**Faculty of Graduate Studies
The University of Jordan**

May, 2015

تعتمد كلية الدراسات العليا
هذه النسخة من الرسالة
التاريخ: 19/05/2015

19/05/2015

Committee Decision

This Thesis (Investigating Parallel Implementations of Electronic Voting Verification and Tallying Processes) was successfully defended and approved on 5/5/2015.

Examination Committee

Signature

Dr. Ghieth Ali Abandah, (Supervisor)
Assoc. Prof., Computer Engineering Department



Dr. Khaled Ahmad Darabkeh (Member)
Assoc. Prof., Computer Engineering Department



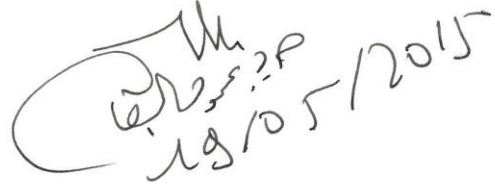
Dr. Basel Ali Mahaftha (Member)
Assoc. Prof., Computer Science Department



Dr. Ali Mohammad Shatnawi (Member)
Assoc. Prof., Computer Engineering Department
(Jordan University of Science & Technology)



تعتمد كلية الدراسات العليا
هذه النسخة من الرسالة
التاريخ: 5/5/2015



INVESTIGATING PARALLEL IMPLEMENTATIONS OF ELECTRONIC VOTING VERIFICATION AND TALLYING PROCESSES

By
Israa Ahmad Saadeh

Supervisor
Dr. Gheith Abandah

**This Thesis was Submitted in Partial Fulfillment of the Requirements for the
Master's Degree of Computer Engineering and Network**

**Faculty of Graduate Studies
The University of Jordan**

May, 2015

Committee Decision

This Thesis (Investigating Parallel Implementations of Electronic Voting Verification and Tallying Processes) was successfully defended and approved on 5/5/2015.

Examination Committee

Signature

Dr. Ghieth Ali Abandah, (Supervisor)
Assoc. Prof. of at Computer Engineering Department

Dr. Khaled Ahmad Darabkeh (Member)
Assoc. Prof. of at Computer Engineering Department

Dr. Basel Ali Mahaftha (Member)
Assoc. Prof. at Computer Science Department

Dr. Ali Mohammad Shatnawi (Member)
Assoc. Prof. of at Computer Engineering Department
(Jordan University of Science & Technology)

Dedication

This thesis is dedicated to all my family members for their endless love, spiritual support, and encouragement.

Acknowledgement

First, I would like to sincerely thank my supervisor Dr. Gheith Abandah for his guidance and support throughout this research. His advice, comments, discussions, and interpretations were very beneficial for my completion of this thesis. I learned from his insight a lot. I believe that I learned from the best and I deeply appreciate it.

Next and foremost, I owe my deepest gratitude towards my husband for his eternal support, understanding of my aspirations, patience, sacrifice, and encouragement in my many crises. His love and support have always been my strength. Without him, I would not be able to complete much of what I have done and become who I am.

My daughter and son deserved my wholehearted thanks as well for giving me happiness during the years of my studies. My parents, thank you for your love and support throughout my life. Thank you both for teaching me the value of hard work and the good things that really matter in life. My parents in law, I gratefully thank your love, affection, and moral support.

Thank you, God, for always being there for me.

List of Contents

Committee Decision	ii
Dedication	iii
Acknowledgement	iv
List of Contents.....	v
List of Tables	vii
List of Figures	viii
List of Abbreviations	ix
Abstract.....	x
1. Introduction.....	1
1.1 Purpose.....	2
1.2 Objective	4
1.3 Research Questions	4
1.4 Thesis Organization	5
2. Literature Review	6
3. Theoretical Background.....	12
3.1 S-Vote System.....	12
3.1.1 Cryptography	12
3.1.2 Homomorphic Cryptography	13
3.1.3 Zero-Knowledge Proofs.....	14
3.2 Parallel Implementation Theoretical Framework.....	18
3.2.1 Parallel Algorithm Design and Implementation	18
3.2.2 Parallel Algorithm Performance Measures.....	20
4. Implementation Details.....	22
4.1 Initialization Routine.....	22
4.2 Voting Process Simulation	25
4.3 Verification and Tallying Processes Implementation	31
4.3.1 Verification Checks	31
4.3.2 Tallying Process.....	37
4.4 Verification and Tallying Processes Parallel Implementations	39
4.4.1 Task Parallelism Scheme	39
4.4.2 Master/Slave Parallelism Scheme.....	41
4.4.3 Data Parallelism Scheme	43

5.	Experimental Evaluation Results and Discussions.....	45
5.1	Experimental Setup	45
5.2	Sequential Implementation Results.....	46
5.3	Alternative Parallel Implementation Results	47
5.4	Data Parallelism Speedup Evaluation	49
5.5	Studying the Effect of Number of Candidates and Options.....	52
5.6	Discussion	54
6.	Conclusion and Future Work.....	55
	References.....	57
	Abstract in Arabic	62

List of Tables

Table 1: Sequential Execution Time Distributed -----	46
Table 2: Performance Time Evaluation of the Parallel Schemes with four cores -----	47
Table 3: Data Parallelism Speedup Evaluation -----	49
Table 4: Data Parallelism Scheme Isoefficiency-----	51
Table 5: Number of Candidates Effect on ZK Proof -----	52

List of Figures

Figure 1: Eligible Voter List Hash Table -----	24
Figure 2: Simulated Voting Process -----	26
Figure 3: Clear Voting Vectors of B Ballots each of C Candidates -----	27
Figure 4: The Process of Encrypting and Adding ZK Proofs for One Voting Vector ---	28
Figure 5: Paillier Zero Knowledge Prover Algorithm -----	29
Figure 6: Adding Digital Signature to Voting Message -----	30
Figure 7: S-Verification and Tallying Processes-----	32
Figure 8: Voter Eligibility and Multiple Voting Check-----	33
Figure 9: Ballot Validity Check (ZK Proof)-----	35
Figure 10: Ballot Authenticity Check (Digital Signature)-----	37
Figure 11: Tallying Process and Final Tally's Decryption -----	38
Figure 12: Task Parallelism Scheme-----	40
Figure 13: Master/Slave Parallelism Scheme -----	42
Figure 14: Data Parallelism Scheme-----	44
Figure 15: Overall Sequential Execution Time -----	46
Figure 16: Serial Execution Time Distribution -----	47
Figure 17: Parallel Implementation Speedups with Four Cores -----	48
Figure 18: Data Parallelism Speedup-----	50
Figure 19: Data Parallelism Efficiency-----	50
Figure 20: Data Parallelism Scheme Isoefficiency Metric -----	51
Figure 21: Number of Candidates Effect on the ZK Proof Check Time -----	53
Figure 22: Number of Options Effect on the ZK Proof -----	53

List of Abbreviations

Acronym	Definition
S-Vote	Secure Electronic Voting System
ZK	Zero Knowledge
PKC	Public Key Cryptosystem
RSA	The Rivest-Shamir-Adleman cryptosystem, a cryptosystem for public-key encryption
SHA-256	Secure Hash Algorithm with 256 bits Message Digest Size

INVESTIGATING PARALLEL IMPLEMENTATIONS OF ELECTRONIC VOTING VERIFICATION AND TALLYING PROCESSES

**By
Israa A. Saadeh**

**Supervisor
Dr. Gheith A. Abandah**

Abstract

Electronic voting systems are being implemented in several countries to provide accuracy and efficiency for the electoral processes with an increased level of security. The Secure National Electronic Voting System (S-Vote) is adopted in this study for its state-of-the-art technologies, privacy, and secure processes. The S-Vote system is a homomorphic e-voting system that uses zero knowledge (ZK) proof protocol to preserve the voter's privacy. Unfortunately, The ZK proofs' is a complex and time consuming protocol which affects the scalability of any homomorphic e-voting system.

This thesis investigates the parallel implementation of the S-Vote verification and tallying processes to reduce the time of vote verification checks especially the ZK proofs verification. Basically, the vote verification process consists of ZK proof, digital signature, and voter eligibility checks.

This thesis work implements parallelism using java multithreaded programs for parallel program execution. It proposes three parallel implementation schemes for the vote verification and tallying processes which are task, master/slave, and data. The task parallelism spawns a separate thread to perform one of the verification process checks (tasks). The master/slave scheme spawns a thread for each voting kiosk package (client) that performs all the checks. The data parallelism scheme spawns a number of threads equal to the number of physical cores of the tallying machine. Each thread performs the whole verification process checks where the voting kiosk packages are dynamically distributed among them.

This thesis uses java development libraries and the library of the Paillier thep java project to simulate the vote casting process and develop the verification and tallying processes based on S-Vote proposed processes. It implements and evaluates the three parallel execution schemes and compares their performance against the serial implementation.

The obtained results show that the data parallelism scheme is the best. It has the highest relative speedup and efficiency with lowest processing cost. It can verify and tally 64,000 ballots in about 44 minutes with 27.5 relative speedup and 86% efficiency while using 32 threads running on the multi-core tallying machine with 32 cores.

The data parallelism scheme reduces ZK proof time. It has a linear speedup with respect to the number of cores and can be used to extend the use of S-Vote system for large electoral processes. For example, using a tallying machine with 128 cores can reduce the verification and tallying processes time for a country as big as Jordan from 25.4 days to 5.7 hours.

1. Introduction

Many democratic societies all over the world suffer from election fraud, suppression, falsification and mock elections (Allansson et al., 2012). They have serious problems throughout their election processes which include, but not limited to, voter lists manipulation, ballots stuffing, voter intimidation and vote buying. On the other hand, the voting centers are often heavily staffed to administer identity check, voting eligibility and ballot dispersal. Some staff members unfaithfully enforce the regulations for the benefit of their favorite candidates. Identity check is intricate business in cultures where women or men cover their faces. Moreover, primitive techniques are often used to disallow multiple voting such as cutting the edge of the ID card or dipping the voter's finger in special ink.

These societies look forward to new fair election systems that can overcome the traditional election systems weaknesses, prevent electoral fraud and improve voter participation and trust. The electronic voting systems can become a popular alternative if they satisfy the following main challenges (Antonyan et al., 2007), (Karro et al., 1999), and (Joaquim et al., 2003):

1. Accuracy: count only the valid votes without being tampered with and exclude any invalid vote from the final tally.
2. Democracy: allow only eligible voters to vote and every voter to vote only once.
3. Privacy: do not reveal any voter's choice or allow any voter to prove how he voted. This is to avoid voter intimidation and vote selling.

4. Verifiability: allow anyone to check that each vote was cast by an eligible voter and all votes are correctly counted. In case of electoral disputes, provides means for rechecking the results.
5. Security: always satisfy reliability, availability and data integrity requirements. Additionally, satisfy the accuracy, democracy and privacy requirements and prevent inside or outside attackers from undermining these requirements.
6. Flexibility: support various election types such as parliaments, municipalities, student boards, plebiscites, referendums, etc. Support any eligible voter to vote irrespective of his native language, special needs or literacy level. Allow him to vote in any voting center that is convenient to him. One more aspect; is the flexibility of changing the hardware devices when new or better devices are available.
7. Cost effectiveness: use economic software and hardware components that are important for large-scale elections.
8. Scalability: efficiently carry out various sizes of elections that achieve flexibility, provide better return on investment and facilitate mass quantities production.

1.1 Purpose

The electronic voting schemes are based on blind signatures, homomorphic encryption or mix-net (Huszti, 2011). The most popular schemes are based on homomorphic electronic voting (Peng and Bao, 2009), (James et al., 2000), (Baudron et al., 2001), and (Hirt et al., 2000).

These schemes count the votes without decrypting them. Such systems have an efficiency bottleneck in vote validity check that is required to preserve the system's privacy property. The vote validity check uses ZK proof to verify that each encrypted vote contains valid data without revealing the vote itself. This proof has too long computation time which limits the application of e-voting, especially in large-scale elections.

Zero-knowledge proof is a method by which one party (the prover) can prove to another party (the verifier) that a given statement is true, without conveying any information apart from the fact that the statement is indeed true. The goal is to prove a statement without leaking extra information

Unlike traditional paper based elections, it is impossible to monitor all electronic operations performed on data from ballot casting to tallying. Accordingly, the validity of votes must be proved by the voters and publicly be verified. The concept of election verifiability that votes have been recorded, tallied and correctly declared is called end-to-end verifiability (Adida, 2006) and (Dagstuhl et al., 2007). This verification is long and complex.

This thesis adopts the S-Vote homomorphic based e-voting system for national and local elections (Abandah et al., 2014). This system relies on homomorphic cryptography, ZK proofs, biometrics, smartcards, open source software, and secure computers for securely and efficiently implementing the system processes over the various stages of the electoral process. It efficiently achieves the e-voting requirements described earlier in term of high accuracy, security, flexibility, privacy, scalability, and cost. However, it is not suitable for large electoral processes due to the time consuming cost of ZK proofs.

It is believed that the S-Vote system can practically be implemented in many countries and would improve trust and participation in the political life if we can overcome the ZK proof overhead. Consequently, we can perform the final verification and tallying processes in not more than eight hours.

1.2 Objective

The objective of the study is to minimize the effect of ZK proof overhead using parallel implementation. So that, we can perform the vote verification and tallying processes in not more than eight hours. Respectively, the S-Vote system will be more trusted for democracies since it satisfies all e-voting requirements in conjunction with practical deployment. For this purpose, we do the following:

- Review S-Vote system's aspects and design.
- Efficiently implement the voting, verification and tallying processes based on S-Vote determinants.
- Employ the multithreading programming techniques to exploit the parallelism and solve the problem of long ZK proofs computation.
- Provide alternative parallel implementation schemes.
- Evaluate the parallel schemes and offer the approach that leads toward acceptable system performance.

1.3 Research Questions

This study implements end-to-end verifiability processes and evaluates the performance of their sequential and parallel executions. Finally, it will try to find out the answers for the following questions:

1. Does the use of parallel execution reduce the votes' verifiability overhead in homomorphic e-voting systems?
2. Which parallel technique is more efficient for deployment?
3. Does the designated parallel solution meet e-vote system requirements?
4. Does the designated parallel solution spread the use of S-Vote system?

1.4 Thesis Organization

The following chapter (chapter 2) explores the complexity of e-voting system, sheds light on the homomorphic e-voting bottleneck and reviews the related work in the field of e-voting systems and parallel programming. Chapter 3 describes the technologies used in the implementation of vote casting, vote verification and tallying processes. This chapter also provides guidelines for developing a parallel program. Chapter 4 details the implementation of voting, verification and tallying processes. It also proposes solutions that can assist in reducing the time consumed in vote verification process. Chapter 5 evaluates the proposed parallel implementation schemes and discusses the performance evaluation results. The conclusion chapter includes the concluding remarks of this thesis and outlines the future work.

2. Literature Review

There have been a number of e-voting systems used in different countries with varying success degrees. Most of the existing e-voting schemes can be classified into two main categories mix-net and homomorphic voting. Mix-net voting employs a mix network to shuffle the encrypted votes before they are decrypted so that the votes cannot be traced back to the voters (Schryen et al., 2009), (Groth, 2003), and (Andrew Neff, 2003). Homomorphic voting schemes exploit homomorphism of certain encryption algorithms. They tallying the votes without decrypting them and only decrypt the sum of the votes (Peng, 2005), (Groth, 2005), and (Peng et al., 2004). Homomorphic voting tallying process costs one single decryption operation for each candidate, so it is much more efficient than tallying in mix-net voting that includes a costly mix network. For this property, Homomorphic voting scheme is preferable for its accuracy, privacy and robustness.

Cramer et al. (1997) proposed a scheme that sends all encrypted votes to a single combiner that computes encrypted tally in a publicly verifiable way using homomorphic cryptosystem. It forwards the tally by running threshold cryptosystem. This model is optimal for the communication between voters and authorities but has tallying computation overhead as the number of candidates increases.

The homomorphic cryptosystem of Paillier provides an efficient public key cryptography. Its additively homomorphic property can be utilized by secure electronic voting systems (Paillier, 1999) and (Damgård et al., 2001).

Unfortunately, the ZK proofs requisite is the bottleneck of homomorphic e-voting systems for its high time consuming cost.

Many researchers such as (James et al., 2000), (Baudron et al., 2001), (Peng et al., 2004), (Katz et al., 2001), (Kiayias et al., 2002), and (Lee et al., 2002) employ

complex Zero knowledge proofs. Cramer et al. (1994) used ZK proof of partial knowledge that has a linear cost relationship with the number of candidates for every vote.

Many researches were introduced to overcome this bottleneck. Baudron et al. (2001) proposed an election system in which the tally is computed in intermediary levels such as local, regional and national results. The separation between levels is non-cryptographic feature. It is a way to reduce the computational cost by distributing the ZK proofs' calculations. On the other hand, weaknesses consideration is ongoing to system's scalability.

Lee et al. (2000), Katz et al. (2001), and Groth (2005) proposed homomorphic e-voting schemes that adjust the vote format and the corresponding vote validity check mechanism. The large number of checks in small ranges are replaced by a smaller number of checks in larger ranges. Chida et al. (2008) reduces the cost of computation and communication by one fourth to one half. So it is still not efficient enough for large-scale election processes.

An interesting technique called batched bid validity check was designed in (Neff, 2003) and (Peng et al., 2007) to improve the efficiency of bid validity check. It is not a new technique; it is an extension of the traditional batch verification techniques. Meanwhile, this technique has three drawbacks: Firstly, it employs different sealing and parameter settings and cannot guarantee whether it can suit the frequently employed Paillier encryption or its distributed version in homomorphic e-voting schemes. Secondly, it supports one-candidate Yes/No election. Thirdly, it is still not efficient enough for large-scale election applications.

Peng and Bao (2009) proposed two new non-interactive ZK vote validity checks called Protocol 1 and Protocol 2. Their improvements are more advanced than that in

(Peng et al., 2007). They not only do integrate proof of validity of multiple votes like in (Peng et al., 2007) but also the operations within each proof of validity of vote. Both protocols can guarantee much more efficient validity scheme of vote with an overwhelmingly large probability. There is general solution for homomorphic e-voting schemes. They are not limited to special election rules or additive homomorphic encryptions.

Protocol 1 modifies and extends the batched bid validity check in (Peng et al., 2007) and (Neff, 2003). It greatly improves the efficiency of vote validity check computation when only one candidate is selected in a vote. It provides more formal security model than that in (Kikuchi et al., 2000) and (Peng et al., 2007) to illustrate the privacy. However, it supports only one-candidate per vote, needs six rounds of communication and may be too interactive for some applications. In addition, it is not efficient enough for large-scale election applications.

Protocol 2 employs batched ZK proof too, but it is a completely novel. It is more flexible since it does not limit the number of selected candidates in a vote. Moreover, it needs fewer rounds of communication and has efficient computation than Protocol 1.

Peng and Bao (2010) employed honest verifier ZK proof security model such that the privacy depends on a trust assumption that verifiers are honest. They also proposed a scheme to improve the efficiency of homomorphic e-voting system without optimizing the ZK proof itself. This scheme can only handle a small number of voters. The voters' votes must be grouped. The tallying must separately be carried out in every group. After that, all results will be aggregated to get the final electoral results.

Abandah et al. (2014) proposed a new end-to-end homomorphic e-voting system (S-Vote). It relies on Paillier homomorphic cryptography and non-interactive zero-knowledge Protocol 2 described in (Peng and Bao, 2009). Hence, the vote tallying takes

negligible time compared to verifying the vote validity and authenticity. Verifying the vote validity involves checking the ZK proof.

Adida (2006), provided constructions of e-voting system using BGN (Boneh et al., 2005) and Paillier homomorphic cryptosystems. The BGN constructions is only practical for small number of voters and small cipher-text size. Messages computing evaluating are performed in long time. He noted that all constructions are easily parallelized. He assumed that the running time can be reduced directly by using more computers.

Clarkson (2008) introduced Civitas mix-net e-voting system based on (Catalano et al., 2005) cryptographic voting scheme¹. The Civitas security is not free. Tradeoffs exist between the levels of security provided by Civitas tabulation, the time required for tabulation and the tabulation monetary cost. Clarkson (2008) divided the votes into blocks. The blocks were exploited independently to decrease tabulation time by processing blocks in parallel and giving a set of tabulation teller machines for each block. Tabulation time then does not depend on number of voters. Therefore performance can scale independently of the number of voters.

Baudron et al. (2001), Adida (2006) and Clarkson (2008) used distributed system architecture to reduce the vote verification running time. The applications are still sequentially executed but they run on multiple computers (cluster) with high cost of parallelism.

Feng and Balaji (2009) and Loka et al. (2010) said that *sequential programming* is dead. The single-core efficiency affects sequential execution model performance. Kirk and Hwu (2010) show that the stalling of clock frequencies due to heat dissipation and energy consumption issues prevents further improvements in this area.

Parallel computer revolution is introduced as the processors' developers switch to a model where the microprocessor has multiple processing units (cores) (Hwu, 2008). The number of cores per processor chip doubles every 18-24 months based on Moore's law.

Clearly, this change of paradigm has had a huge impact on the software development. Parallel computing can increase the application performance by the execution on multiple cores. For this to happen, the applications must be programmed to exploit parallelism. Dongarra et al. (2003) shows that the responsibility for achieving this falls on the application developers. This new interest toward parallel programming is called *concurrency revolution* (Sutter and Larus, 2005) and is taking prominent role on the stage.

Kasim (2008) discussed two main approaches to parallelize a program: auto-parallelization and parallel programming. In the auto-parallelization approach, the sequential program is automatically parallelized using parallel compiler. Thus, the program needs to recompile with parallel compiler and no manual modifications are required. However, the amount of parallelism reached using this approach is low due to the complexity of the required automatic transformation. In the parallel programming approach, the application is explicitly modified or developed to exploit parallelism. Generally, this approach obtains a higher performance than auto parallelization one but with the cost of more programming efforts.

Pusukuri et al. (2011) presented that the performance of a parallel application depends on the number of threads used to run on a multi-core system. He provided guidelines for finding the appropriate number of threads for getting best performance.

Diaz et al. (2012) provided tips of motivation showing the relationships between the problem and the various approaches to divide it into parts. These parts are intended to be executed simultaneously via threads to solve the problem.

3. Theoretical Background

This chapter provides the theoretical basis used in S-Vote system giving more attention to the verification and tallying processes implementations. As this research is mainly within the field of parallel programming, we recall here the best practices of parallel implementation design and evaluation.

3.1 S-Vote System

Abandah et al. (2014) adopts state-of-the-art technologies to meet the e-voting requirements in their proposed S-Vote system. This thesis uses cryptography, homomorphic cryptography, and zero-knowledge proofs for implementing the voting, verification, and tallying processes described in the S-Vote.

3.1.1 Cryptography

Industry standard public key cryptography (PKC) is used for achieving the S-Vote system authenticity and confidentiality (Schneier, 2007). Public key cryptography relies on key pairs. A key pair for system X consists of a private key K_x^- and a public key K_x^+ . The private key is only known by system X where the public key is available to other systems that need to communicate with system X .

Authentication: System X signs a message m by encrypting it using its private key ($K_x^-(m)$) and sends it. The receiver of the encrypted message validates/authenticates the source of the message when it successfully retrieves the original message using the system public key (Schneier, 2007).

$$m = K_x^+(K_x^-(m)) \quad (1)$$

Confidentiality: A message m is encrypted using system X public key ($K_x^+(m)$). Only system X ; who is the holder of the private key, can retrieve the original message using its private key.

$$m = K_x^-(K_x^+(m)) \quad (2)$$

Using RSA industrial cryptography provides sufficient security level for S-Vote system (Silverman, 2002).

Data Integrity: The public key cryptography is also used to check the integrity of data in addition to authentication which is defined as digital signature. A signed message is a message along with its signed digest ($m + K_x^-(H(m))$). The cryptographic hash algorithm is used to compute the message digest $H(m)$. The authenticity and integrity are validated when the digest computed from the received message m , matches the recovered digest received.

$$H(m) = K_x^+ \left(K_x^-(H(m)) \right) \quad (3)$$

The RSA key pair cryptography and SHA-256 hash algorithm are used in the S-Vote system to provide the authenticity and data integrity of the encrypted ballot records casted via the voting kiosk.

3.1.2 Homomorphic Cryptography

S-Vote uses Paillier key pair cryptosystem for encrypting the voting vectors and decrypting the encrypted tallies. It is adopted for its useful homomorphic addition feature in preserving the privacy of votes (Paillier, 1999). Principally, the Paillier homomorphic allows finding the sum of the clear votes by multiplying their encrypted votes.

$$K_v^+(m_1 + m_2) = K_v^+(m_1) \times K_v^+(m_2) \quad (4)$$

$$m_1 + m_2 = K_v^-(K_v^+(m_1) \times K_v^+(m_2)) \quad (5)$$

For flexibility, S-Vote allows each voter to select up to O options of the C Candidates. The vote of each voter V_i is encoded as a voting vector

$(m_{i,1}; m_{i,2}; \dots m_{i,j} \dots; m_{i,C})$ where $m_{i,j} = 0$ or 1 for $j = \{1; 2; \dots \dots; C\}$. When the voter chooses the candidate j , then $m_{i,j} = 1$ otherwise it is 0 . The voting vector is encrypted to $(C_{i,1}; C_{i,2}; \dots C_{i,j} \dots; C_{i,C})$ where the homomorphic property allows finding the encrypted tally of candidate j from N number of voters through

$$K_v^+(\sum_{i=1}^N m_{i,j}) = \prod_{i=1}^N K_v^+(m_{i,j}) \quad (6)$$

As a result, we can count the votes casted for candidate j by just decrypting the encrypted tally (Paillier, 1999). For this reason, the Paillier cryptography is fit for this system as it can get the final results while preserving the individual vectors privacy.

3.1.3 Zero-Knowledge Proofs

As S-Vote requires keeping the voter privacy, the zero knowledge proofs are a necessity to ensure that encrypted voting vectors carry valid votes. For instance, voter V_i can cheat by submitting for his favorite candidate j the vote $C_{i,j} = K_V^+(100)$ instead of $C_{i,j} = K_V^+(1)$. For this reason, the system requires that each voter must submit his ZK proof (Lipmaa et al., 2003).

Vote verification checks are typically the bottleneck of the homomorphic e-voting systems for its lengthy and complex calculations (Groth, 2005). The advantage of ZK proofs is allowing one party called prover to convince another party called verifier that he knows some secret or knowledge about specific object without revealing what is the object itself.

Generally, both of the prover and verifier possess the object x , the prover wishes to convince the verifier that x is in the set S . The prover needs to prove this without giving away any information about the object and guaranteeing that no prover strategy may fool the verifier to accept an object not in S , except with negligible probability.

The first property is called *completeness* and the second one is *soundness*. It is a probabilistic protocol and can be amplified to a proof with soundness error 2^{-k} by repeating it k times (Sarath and Ainapurkar, 2014).

The non-interactive zero knowledge proofs do not require an interaction between the prover and verifier. De Santis et al. (2001) change the model in a way that reduces the number of rounds in ZK proofs where just a single message is sent from the prover to the verifier. They showed the possibility of disposing the interaction between the prover and the verifier if they share a common random public reference string. This is enough to perform zero-knowledge proof check without requiring interaction.

Honest-verifier ZK is the simplest type of ZK proofs. The ZK protocol is called honest if both the verifier and the prover fully follow the protocol. The prover shall certainly know the secret and the verifier shall follow the behaviour specified in the protocol to accept the prover claim (Goldreich et al., 1998).

The ZK proofs play a central role in building secure public key cryptosystem. Its complexity-theoretic assumptions secure the system against the cipher text attacks (De Santis et al., 2001).

The S-Vote system adopts an efficient honest verifier ZK protocol which is the non-interactive version of protocol 2 described in (Peng and Bao, 2009). This is made non-interactive using Fiat-Shamir heuristic (Fiat and Shamir, 1987). In this protocol, each $voter_i$ proofs the following two criteria:

$$\bigwedge_{j=1}^C (K_v^-(c_{i,j}) = 0 \vee K_v^-(c_{i,j}) = 1) \quad (7)$$

$$KN \left[\left(\left(\prod_{j=1}^C c_{i,j} \right) / G^O \right)^{1/N} \right] \quad (8)$$

Criterion 1 is a proof that every vote in the voting vector is either 1 (for) or 0 (against).

Criterion 2 is a proof of knowledge of the N^{th} root and demonstrates that there are exactly O ones in the voting vector where G and N are part of the cryptosystem public key.

The following describes the ZK proof protocol 2 adopted by S-Vote from Peng et al., (2009). It is a brief overview for proving that the encrypted value $c_{i,j}$ is within the set S of $\{0, 1\}$. For full description, kindly refer to (Peng et al., 2009).

Vote Casting:

1. Suppose there are n voters and each voter has to choose O parties from the C candidates.
2. Each voter V_i has his voting vector $(m_{i,1}; m_{i,2}; \dots m_{i,j} \dots; m_{i,C})$ where $m_{i,j} = 0$ or 1 for $j = \{1; 2; \dots \dots; C\}$. A rule is followed: $m_{i,j} = 1$ iff the voter V_i chooses the j^{th} candidate.
3. The voting vector is encrypted to $(C_{i,1}; C_{i,2}; \dots C_{i,j} \dots; C_{i,C})$ using homomorphic cryptography where $N = pq$ is the RSA Modulus (Boneh and Franklin, 2001).

ZK Proof:

The prover generates the proving values and challenge for proving that the encrypted vote $C_{i,j}$ is in the set of $\{0, 1\}$ for $j = \{1; 2; \dots \dots; C\}$ that shall be sent to the verifier. A security parameter L is used and is chosen to be 40 according to Fiat-Shamir heuristic (Peng et al., 2009). Since Fiat-Shamir is run for $L = 20$ to 40 executions, the probability for an adversary to fool the verifier for all executions of L is very small and does not exceed 2^{-L} .

1. The prover randomly selects the following proving values for $j = \{1; 2; \dots \dots; C\}$ where:

$$t_{j,0} \in \{0,1, \dots, 2^L - 1\} \quad (9)$$

$$t_{j,1} \in \{0,1, \dots, 2^L - 1\} \quad (10)$$

$$\text{challenge } v \in \{0,1, \dots, 2^L - 1\} \quad (11)$$

$$v_{j,1-m_{i,j}} \in \{0,1, \dots, 2^L - 1\} \quad (12)$$

$$r \in Z_N^* \quad (13)$$

2. The prover generates another proving value for $j = \{1; 2; \dots; C\}$ where:

$$v_{j,m_{i,j}} = v - v_{j,1-m_{i,j}} \text{ mod } 2^L$$

3. The prover generates the commitments that shall be sent to the verifier

$$a = r^N \prod_{j=1}^C (c_{i,j} g^{m_{i,j}-1})^{t_{j,1-m_{i,j}} v_{j,1-m_{i,j}}} \text{ mod } N^2 \quad (14)$$

$$u = r \prod_{j=1}^C S_{i,j}^{t_{j,m_{i,j}} v_{j,m_{i,j}}} \text{ mod } N^2 \quad (15)$$

Public Verification:

The verifier calculates the commitments, checks the response, and returns true when they are matched in probability of $1 - 2^{-40}$ (Peng and Bao, 2009).

$$u^N = a \prod_{j=1}^C C_{i,j}^{t_{j,0} v_{j,0}} \left(\frac{C_{i,j}}{g}\right)^{t_{j,1} v_{j,1}} \text{ mod } N^2 \quad (16)$$

$$v = v_{j,0} + v_{j,1} \text{ mod } 2^L \text{ for } j = \{1; 2; \dots; C\} \quad (17)$$

The Thep, 2011 Paillier java project developed the ZK proofs according to Paillier cryptosystem (N and G). The project has updated the algorithm such that S can be any integer value and not limited by $\{0, 1\}$.

3.2 Parallel Implementation Theoretical Framework

There are several aspects that must be considered when developing a parallel program. Mainly, designing the parallel algorithm for a given application problem, implementing the proposed design using parallel programming languages, and evaluating the developed program.

3.2.1 Parallel Algorithm Design and Implementation

Designing a parallel algorithm follows three main steps in spite of the environment or system are used: decomposition, scheduling, and mapping (Sutter and Larus 2005).

Decomposition divides the application computations and data into parts which can be concurrently processed on parallel processors. Defining the partitions in an appropriate way is one of the intellectual tasks of developing a parallel program. There are many possibilities for partitioning the same problem. Fortunately, there are some typical kinds of decomposition such as task, data, recursive, and pipelined (Sottile et al., 2010) (Andrews, 2000) (Mattson, 2005). The programmers most commonly focus on the computation associated with the problem and the data on which this computation operates to select the appropriate decomposition.

In data decomposition, programmers decompose the data associated with a problem by dividing it into small pieces with approximately equal sizes. They associate the computation with the data portion (Barney, 2012). As a result, each processor performs the same task on different pieces of partitioned data. Data decomposition is also known as domain decomposition.

Contrary to data decomposition, task decomposition initially focuses on the computation to be performed and then dealing with the data. Accordingly, each

processor executes different process on the same or different data but communication usually takes place. The processed data may require passing from one computation unit to the next one as part of a workflow to avoid replication of data (Rauber and Runger, 2010). This scheme is also known as function or control decomposition.

The selected decompositions are coded in a parallel programming language and are typically assigned to threads to be mapped to physical computation units for execution (Rauber and Runger, 2010). Such languages often provide special parallel programming constructs and statements that allow sharing variables and parallel code sections (threads) to be declared. Threads contain regular high-level code sequences that will be assigned later to individual computing unit to be run in parallel. Finally, the compiler is responsible for producing the final executable code.

Scheduling is the assignment of problem partitions to processes or threads. It fixes the order in which the partitions are executed. This can be done statically at the compile time or dynamically at the runtime. Mapping is the assignment of threads onto physical computing units (cores) and is usually done by the runtime environment, but sometimes can be influenced by the programmer (Lewis and Berg, 1999) (Rauber and Runger, 2010).

Threads can run independently, but may also depend on each other resulting in data or control dependencies of threads. The concurrency in parallel programs introduces several classes of potential software bugs of which the race condition are the most commonly known problems. These dependencies are scheduling constraints and may require a specific execution order of the parallel tasks. In this case, synchronization and communication must be put in place. For example; any thread that needs data produced by another one, should only be started after the first thread has actually produced this data (Jacobsen et al., 2010) (Chapman et al., 2007).

3.2.2 Parallel Algorithm Performance Measures

The parallel computing execution time for an application is the time elapsed between the start of the application on the first processor and the end of its execution on all used processors. It should be smaller than the sequential execution time on one processor (Barney, 2012). Generally, smaller parallel execution times are obtained when the workload is equally distributed among the cores, which is called load balancing. In addition, smaller overhead of information exchange, synchronization, and idle times reduces the parallel execution time. Consequently, finding an appropriate scheduling and mapping strategy leads to a good load balance and small overhead but this is often difficult to achieve due to the multiple interactions.

Cost measures like speedup and efficiency are quantitative evaluations of the parallel application performance. The relative speedup factor S_R measures the increase in speed using multiprocessing and is defined by the ratio of the sequential execution time t_s to the parallel time t_p .

$$S_R = \frac{t_s}{t_p} \quad (18)$$

The relative efficiency E_R is a measure of how efficient is the parallel implementation in using the given parallel resources and is defined by the ratio of relative speedup S_R to the number of processors/cores P .

$$E_R = \frac{S_R}{P} \quad (19)$$

The relative cost of parallel computation is proportional to the number of processors used and the parallel execution time and is also defined by the ratio of

sequential execution time t_s to the relative efficiency of the parallel implementation E_R (Wilkinson and Allen 2004).

$$Cost_R = P \times t_p = \frac{t_s}{E_R} \quad (20)$$

The isoefficiency scalability metric of the parallel system establishes a relationship between the workload (W) to be accomplished and the number of cores or processors such that E_R remains constant as P increases (Grama et al., 1993). This metric dictates the rate of W growth required to keep the efficiency fixed as P is increased.

As a result, there is a continual interplay between parallel algorithms, languages, architecture, and performance evaluation.

4. Implementation Details

This research attempts to reduce the votes' verification process time of S-Vote model. First, the voting, verification and tallying processes are implemented in a manner that meets the S-Vote technical and procedural levels of assurance. This implementation applies the system's regulations and determinants that keep the notion of system's trust and transparency valuable properties. It employs the technologies necessary for its secure implementation. Then java multithreading capabilities are used to reduce the verification time. At the end, all running procedures and alternatives are tested and evaluated.

As a consequence, we develop the `projvoting java` project using java 1.7 development kit for building the processes and alternative running procedures. The developed java project uses standard java libraries and the software components from homomorphic thep encryption project (thep, 2011). These components implement Paillier cryptosystem in java along with its homomorphic operations, key generation, and zero knowledge proof where the `BigInteger` is the underlying class.

4.1 Initialization Routine

At first, the initialization routine prepares the eligible voters list. The `Projvoting.NIDTable()` java class generates the eligible voter list that is globally announced for all system's components. For illustration, we create an eligible voters list consists of two million national ID (NID). The list is created once, arranged in hash table data structure, and read by system components whenever needed.

The idea of hashing is to distribute the list entries (values) across an array of buckets (slots). The hash table uses a hash function to compute an index within the array of buckets from which the value shall be correctly stored or found. The hash function

uses a key and run the hash algorithm to compute the index that suggests where the value can be stored or found. The hash collisions will occur when different keys are hashed to the same bucket and must be accommodated in some way (Donald, 1998) (Cormen et al., 2001).

We use the hash table data structure as the average cost for each lookup is independent of the number of elements stored in the table (Demaine and Lind, 2003). In many situations, hash tables are more efficient than the tree search or other table lookup algorithms.

For simplicity, `NIDTable()` defines a continuous range of two millions NID entries. We create a hash table of 200,000 slots where each slot is an array of ten elements. The NID list is stored in the hash table. Each NID is a key for the hash function to compute the *index*. A prime number *prime* is chosen from the NID list such that the NIDs will be equally distributed within the hash table. If a hash collision occurs, the value is stored in the next free index within the slot's array. For example, consider that we have A, B, and C values. The indices are $H(A) = 3$, $H(B) = N$, and $H(C) = 3$. The values will be stored into the hash table such that: $3[0] = A$ (free collision), $N[0] = B$ (free collision), and $3[1] = C$ (collision). Figure 1 illustrates the above procedure.

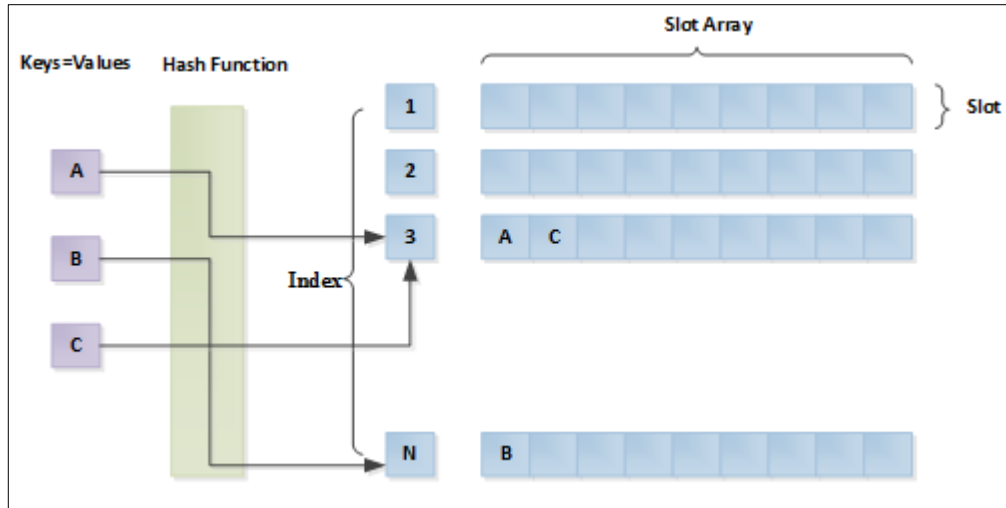


Figure 1: Eligible Voter List Hash Table

The final initialization step is for setting the system's parameters. They are globally declared across the project's classes within the `projvoting.GlobalConstants()` class. These constants are used by voting, verification, and tallying processes:

1. Number of candidates C
2. Number of options O
3. Number of kiosk packages K
4. Number of thread T
5. Number of ballots B
6. Public voting encryption key K_v^+

To generate public voting encryption key K_v^+ , the `projvoting.paillierp.PrivateKey` class generates the encryption private key K_v^- , then it generates the corresponding public key K_v^+ using `getPublicKey()` method from `projvoting.paillierp.PublicKey` class and declares it as a global constant while keeping the private one for final tallying decryption.

4.2 Voting Process Simulation

The voting process is responsible for vote casting and kiosk packages' preparation. Figure 2 shows the developed procedure used to simulate the voting process. The `Projvoting.MainVoting()` class is responsible for generating the kiosk packages. It performs the steps of Fig. 2 flowchart for each ballot within the kiosk which are:

1. **Creating voters voting vector:** The `kioskEntry.ClearKioskTable()` method randomly selects a voter from the eligible voter list, then casts his vote by randomly selects O options out of C candidates (O of ones). Finally, it produces the clear kiosk voting vectors. Figure 3 describes the clear kiosk voting vectors.

$$V_i = (V_{i,1}; V_{i,2}; \dots; V_{i,c}), \quad \text{where } V_{i,c} \quad (21)$$

$\rightarrow \text{Voter}_i \text{ Vote for Candidate}_c$

2. **Generating encrypted voting vectors and ZK proof (Message):** Figure 4 shows these steps which are started with `EncryptingKiosk.GetPublicKey(Vi,j)` method. It generates for each vote $V_{i,j}$ an encrypted vote $C_{i,j}$ and ZK proof $P_{i,j}$. This is done by performing the following:

- a) Calling the `projvoting.Paillier.EncryptedInteger(Vi,j, Kv+)` Paillier encryption class to encrypt the clear vote $V_{i,j}$ using the vote encryption public key K_v^+ and produce encrypted vote $C_{i,j}$.

$$C_i = K_v^+(V_i) = (c_{i,1}; c_{i,2}; \dots; c_{i,c}), \quad (22)$$

$\text{where } c_{i,c} \rightarrow \text{Voter}_i \text{ Encrypted Vote for Candidate}_c$



Figure 2: Simulated Voting Process

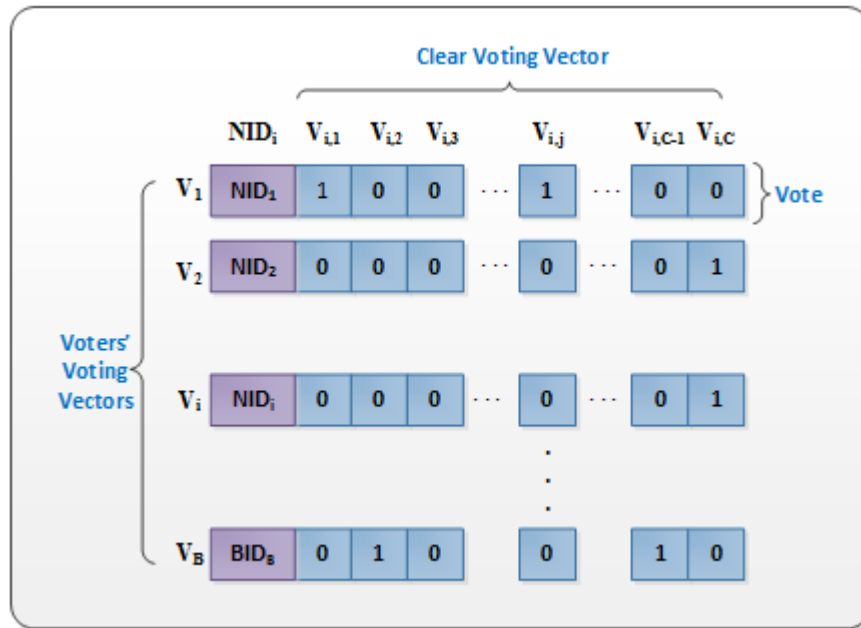


Figure 3: Clear Voting Vectors of B Ballots each of C Candidates

b) Calling `projvoting.Paillier.ZKSetMembershipProver(K+v, Set, msgIndex, Ci,j)` Paillier ZK prover class to generate a proof $P_{i,j}$ for each encrypted vote $C_{i,j}$. This proof claims that the encrypted vote is within the vote set of $\{0, 1\}$ in our case.

Thep, (2011) Paillier prover class first generates the commitments $u_{i,j}$ for proving that the encrypted vote $C_{i,j}$ is in the set of $\{0, 1\}$. The commitments should be sent to the verifier. It then uses the idea behind the Fiat-Shamir paradigm to generate the challenge $e_{i,j}$ from the commitments $u_{i,j}$ for a non-interactive proof. After that, it computes the prover response to the challenge $e_{i,j}$ from the random number $r_{i,j}$. It then uses `getVs()` and `getEs()` to send those $Vs_{i,j}$ and $Es_{i,j}$ values to the verifier that needed for the last part of the proof. As a result, the proof $P_{i,j}$ is a big integer vector of $[u_{i,j}, e_{i,j}, Vs_{i,j}, Es_{i,j}]$. Figure 5 describes the above algorithm. For further detail and correctness proofs, kindly see (Peng and Bao, 2009).

Finally, the `EncryptingKiosk.GetKeyPublicBallotStr(O_i)` method generates for each voting vector an encrypted value $C_{i,o}$ for the number of options O_i within the vector V_i . Then, it generates ZK proof $P_{i,o}$ that the voting vector has O options. Paillier ZK prover class is used where the set is $\{O\}$ in this case.

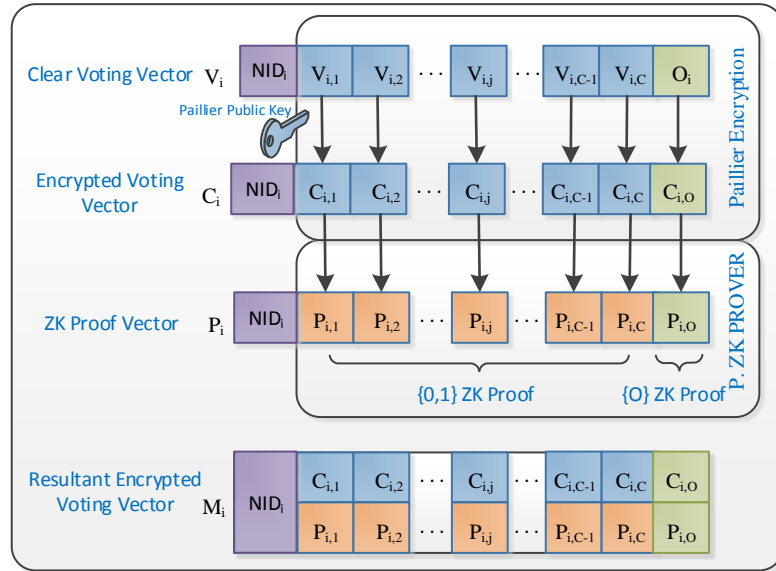


Figure 4: The Process of Encrypting and Adding ZK Proofs for One Voting Vector

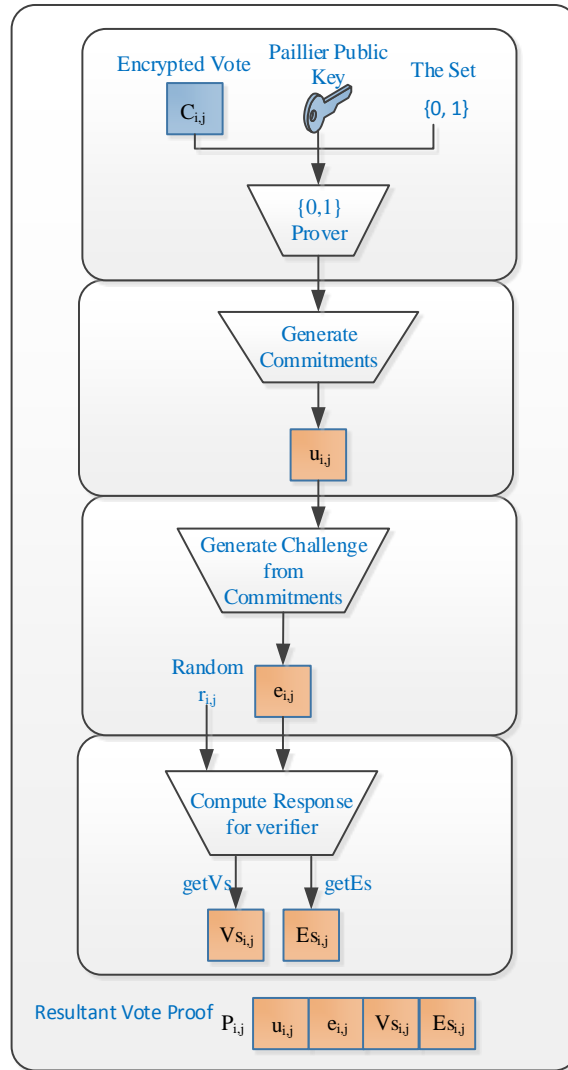


Figure 5: Paillier Zero Knowledge Prover Algorithm

3. Digital Signature: Figure 6 shows that each Message M_i is digitally signed using SHA-256 hash function and RSA cryptosystem. The resulting signature R_i is the vote Receipt. Referring to the S-Vote system architecture, the voter's smartcard signs the ballot. For simulation purposes it is performed by the process of kiosk package preparation. The digital signature is required to ensure the authenticity and integrity of the encrypted voting vector and its proof M_i . The following summarizes this stage :

- a) The `RSAGenerateKeys.GenerateKeys(NIDi)` method generates RSA cryptosystem key pair for each voter and is used in digital signature. Voter_i RSA

private key K_i^- is kept private while the public key K_i^+ is available for verification process.

- b) The `RsaSign.GetMessageDigestPrvKey(Encrypted Balloti + ZK Proofsi)` method runs the SHA-256 hash function to calculate the message digest from each of the encrypted voting vector and the associated ZK proof .
- c) The `RsaSign.GetDigitalSignRsaEncPrvKey(Digesti, NID)` method is responsible for signing the message digest D_i using voter_i RSA private key K_i^- . This is defined by S-Vote system as vote receipt R_i .

$$R_i = K_i^-(H(C_i + P_i)) = K_i^-(H(M_i)) \quad (23)$$

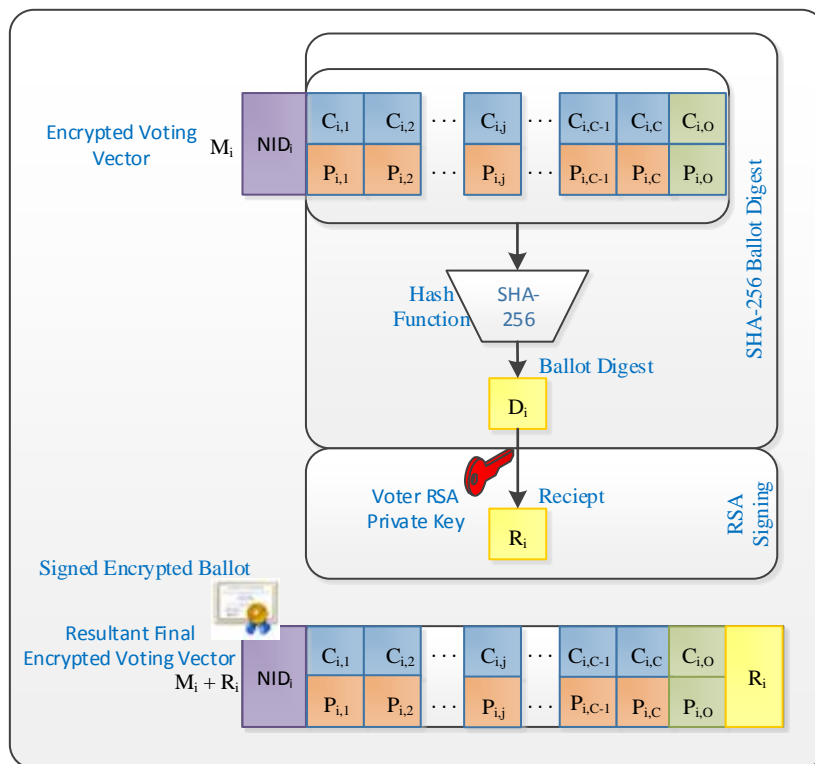


Figure 6: Adding Digital Signature to Voting Message

After the termination of the voting stage, the kiosk packages are ready for verification and tallying processes. Each kiosk package includes the voter IDs, the encrypted voting vectors, and the voting receipts.

4.3 Verification and Tallying Processes Implementation

Figure 7 shows the implemented vote verification and tallying processes based on the S-Vote techniques. S-Vote uses several security controls to meet the system's security requirements. These controls are effectively integrated within the system functional and operational requirements. These are implemented into the form of voting vector validity check (zero knowledge proof), voting vector authenticity and data integrity check (digital signature), voter eligibility check, and multiple voting check.

The verification process starts with performing the above checks for each voter ballot record. Upon finishing the verification stage, the eligible voting vectors move to tallying process using homomorphic property. Finally, the resultant encrypted tally is decrypted using Paillier decryption algorithm.

The `Projvoting.MainVerification()` class is responsible for the sequential execution of the verification and tallying processes by calling the following methods for performing the steps shown in Fig. 7.

4.3.1. Verification Checks

The three main steps in this stage are:

1. **Voter Eligibility and Multiple Voting Checks:** Figure 8 shows the voter eligibility and multiple voting checks that are applied on each ballot record. The `Verification.CheckValidNID(NIDi)` method checks that each voter NID is in the

eligible voter list. It looks for this NID in the eligible voter NIDs hash table. The hash table data structure is used for its good performance.

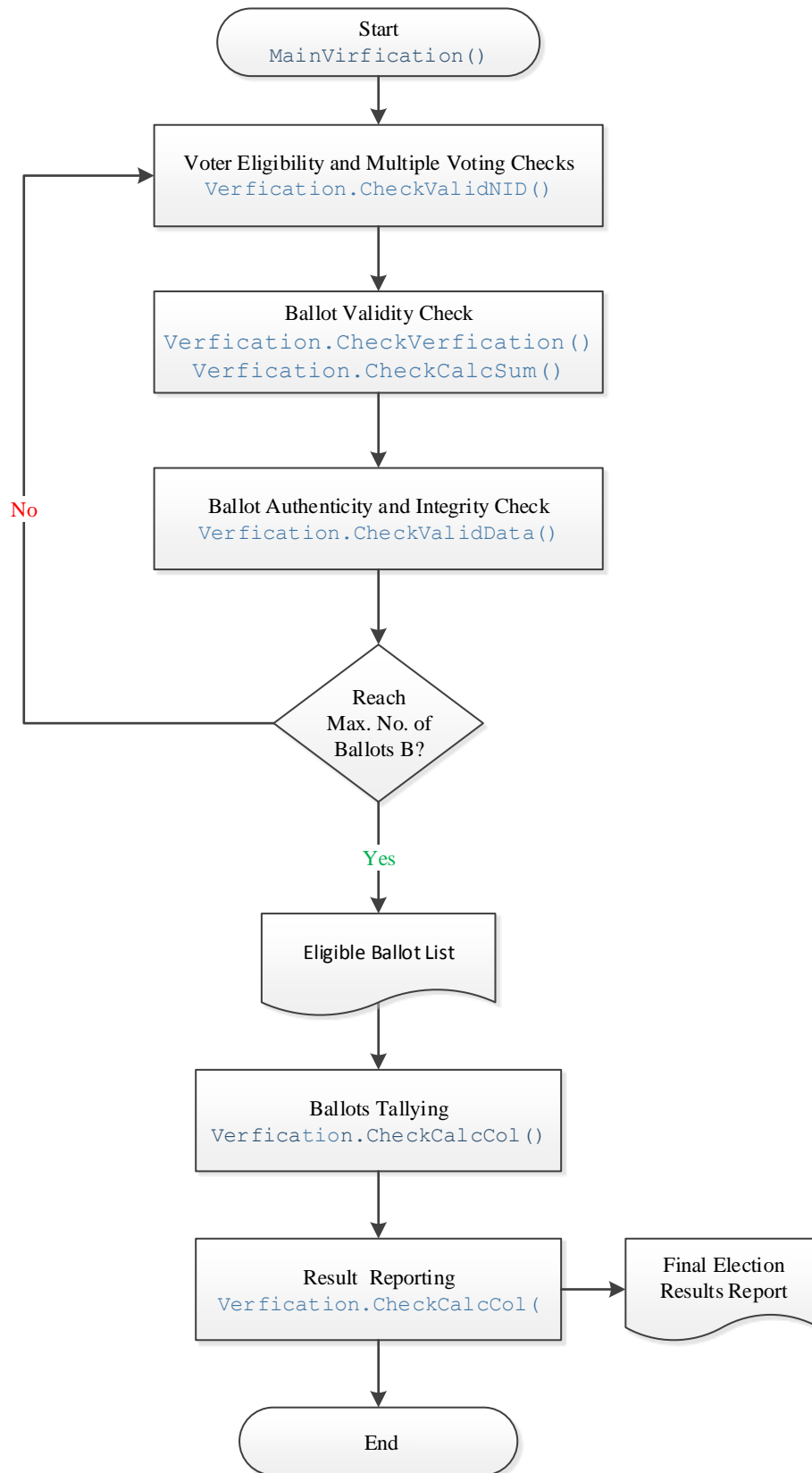


Figure 7: S-Verification and Tallying Processes

The three possible cases for this check are:

- I. Voter ID is found within the voter list: It is marked as a voted voter and the check passes (see NID_{3i} example in Fig. 8).
- II. Voter ID is not found within the Voter List: Eligible voter ID error occurs. The associated voting vector is not forwarded to the tallying process. The check fails for this voter ID. It is reported for voter eligibility check failure (see NID₆₀ example in Fig. 8).
- III. Vector ID is found within the voter list but with multiple voting error: the duplicated voting vectors of this Voter ID are excluded from the tallying process. The check fails. It is reported for voter eligibility check failure (see NID_{4N} example in Fig. 8).

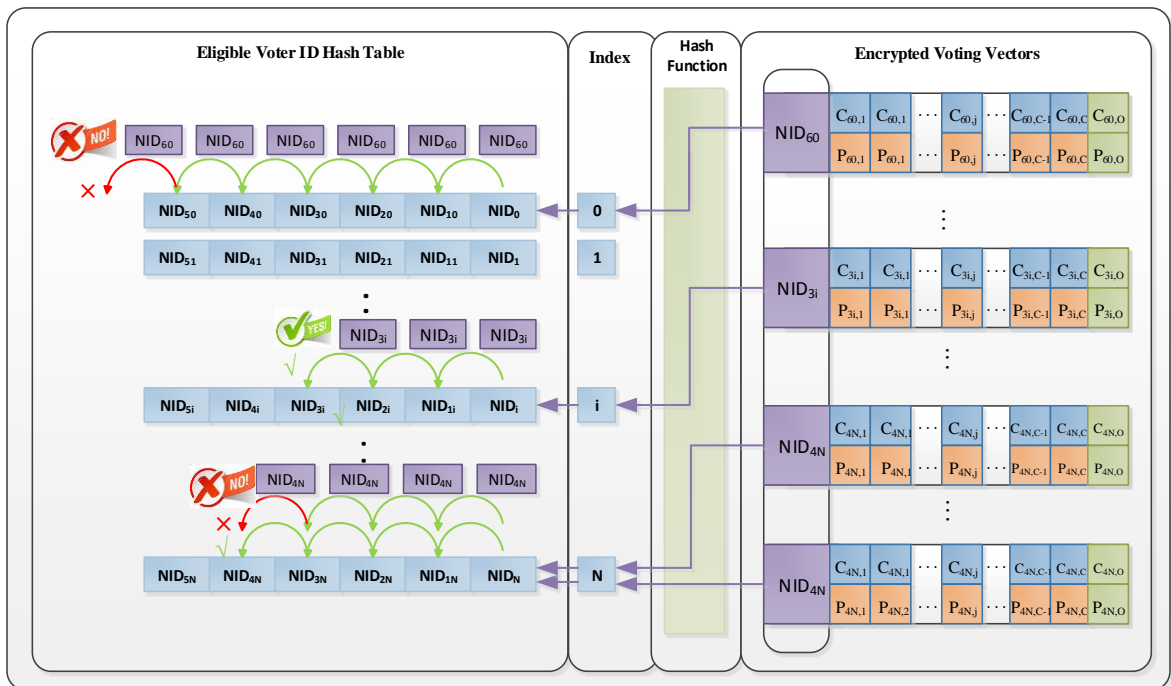


Figure 8: Voter Eligibility and Multiple Voting Check

2. Ballot validity (zero knowledge proof) check: Figure 9 shows ZK proof check that is applied to check the validity of each voting vector. The Verification.CheckVerification() method checks that each encrypted vote is

within the set $\{0, 1\}$. The `Verification.CheckCalcSum()` class checks that the number of options in the encrypted voting vector is O . The details are as follows:

- a) The `projvoting.Paillier.ZKSetMembershipVerifier(Kv, Ci,j, ui,j, theSet)` Paillier zero knowledge verifier class calculates the proof values $P_{i,j}$ for each encrypted vote $C_{i,j}$. The verifier class uses the commitments $u_{i,j}$ generated by the prover to check that $C_{i,j}$ is in the set $\{0, 1\}$.
- b) The `projvoting.Paillier.checkResponseNonInteractive(eVals, vVals, e)` method checks the response from the prover $V_{S_{i,j}}, U_{S_{i,j}}$, and the challenge $e_{i,j}$ using the Fiat-Shamir heuristic. The response check returns true if it is OK and accept the prover claim that $C_{i,j}$ in the set $\{0, 1\}$, otherwise it returns false.

Same class and method are used to check the prover claim that the voting vector has O options. $C_{i,0}$, $\{0\}$ set, and $P_{i,0}$ proof vector are used instead. There are four possible outcomes and actions for these checks:

- I. The ZK verifier accepts the prover claim that all $C_{i,j}$ are in $\{0, 1\}$: The true response is reported for each $C_{i,j}$, and the ZK proof check passes.
- II. The ZK verifier rejects the prover claim that a $C_{i,j}$ is in $\{0, 1\}$: Invalid voting vector error occurs. False response is reported for $C_{i,j}$. The associated voting vector is excluded from the tallying process. The ZK proof check fails. It is reported that the voter's *ballot_i* is excluded for vote validity check failure.
- III. The ZK verifier accepts the prover claim that $C_{i,0}$ has O options: The true response is reported for the encrypted vector_i and the ZK proof check passes.
- IV. The ZK verifier rejects the prover claim that $C_{i,0}$ has O options: Invalid voting vector error occurs. False response is reported for the voting *vector_i*. It is excluded from the tallying process. The ZK proof check fails. It is reported that the voter's *ballot_i* is excluded for vote validity check failure.

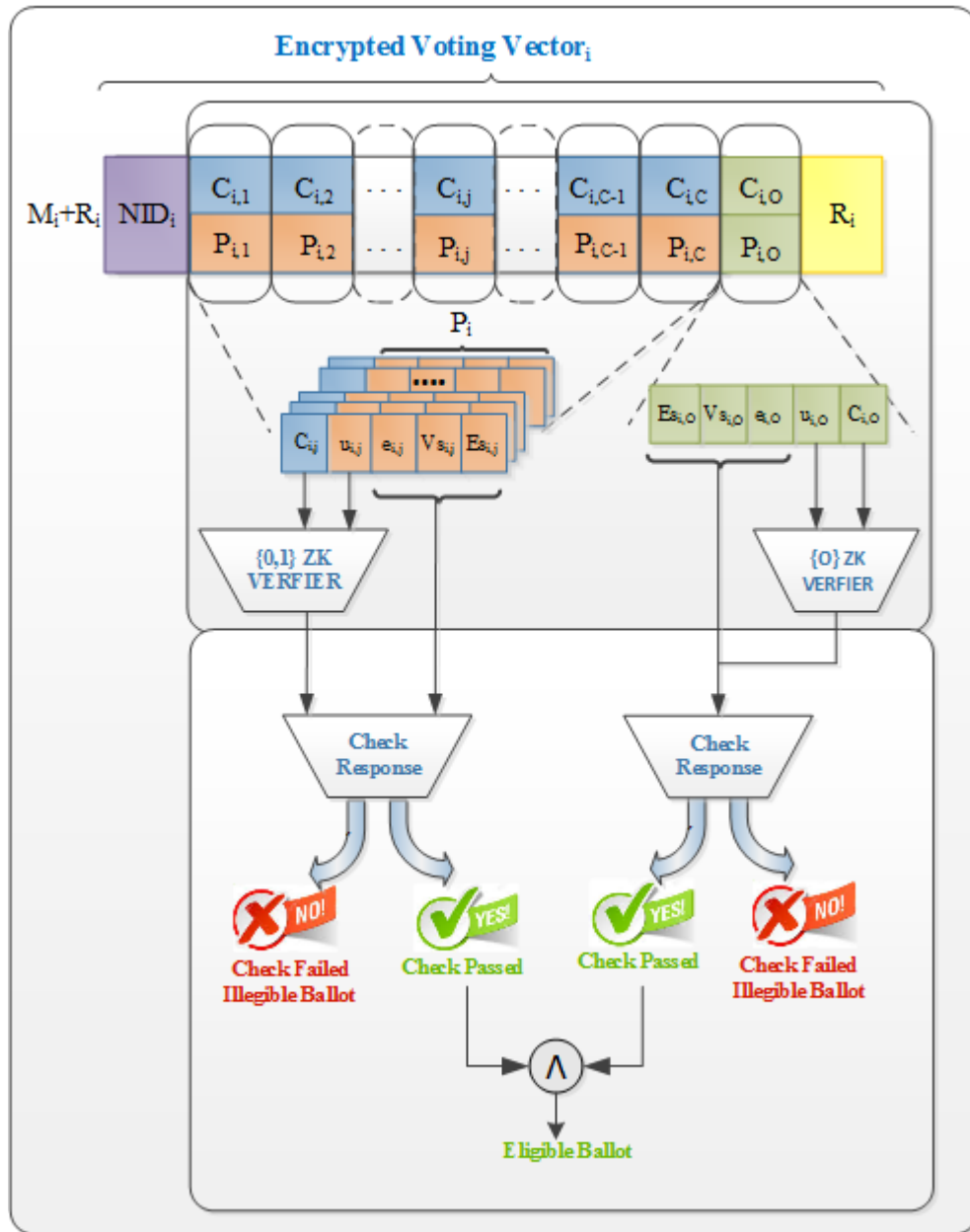


Figure 9: Ballot Validity Check (ZK Proof)

3. Ballot authenticity and integrity (digital signature) check: Figure 10 shows digital signature check that is applied to check the integrity of each voting vector. The Verification.CheckValidData() method is responsible for performing this check.

The detailed procedure is described below:

- a) The `RsaSign.GetMessageDigestPrivKey(Received Encrypted Balloti + ZK Proofsi)` method runs SHA-256 hash function to calculate the message digest from each received Message M_i encrypted voting vector and its associated ZK proofs .

$$\text{Calculated Digest} = H(C_i + P_i) \quad (24)$$

- b) The `RsaSign.GetDecSignEncrUsePublicKey(Received R_i , NID)` method is responsible for decrypting the receipt R_i received using voter $_i$ RSA public key K_i^+ .

$$\text{Recieved Digest} = K_i^+(R_i) \quad (25)$$

- c) The `Verification.CheckValidData()` caller method compares the received digest with the calculated one and considers this check is passed if they match.

$$H(C_i + P_i) = K_i^+(R_i) \quad (26)$$

There are two possible outcomes and related actions for this check:

- I. The encrypted ballot and its ZK proof M_i are authentic: The digital signature check passes for the encrypted voting vector C_i .
- II. The encrypted ballot and its ZK proof M_i are not authentic: The $ballot_i$ is excluded from the tallying process. Digital signature check fails. It is reported that the voter's $ballot_i$ is excluded for ballot authenticity check failure.

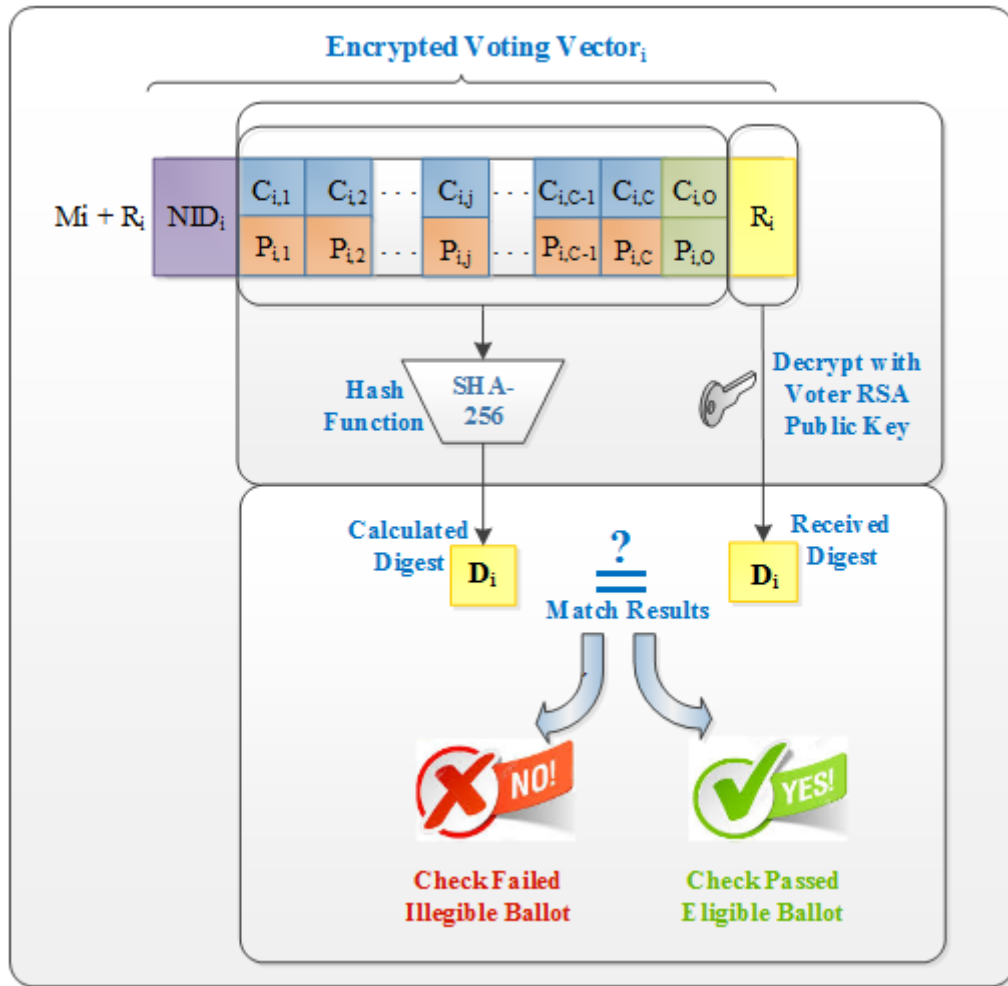


Figure 10: Ballot Authenticity Check (Digital Signature)

4.3.2. Tallying Process

The final tallying process goes as follows:

1. Ballot tallying: All encrypted ballots that passed the four checks are considered eligible voting vectors and are included in the final tallying process. Figure 11 shows that tallying process. It finds the final encrypted tally for each candidate C_j according to the Paillier homomorphic property.
 - a) The `Verification.CheckCalcColumn(C_j)` method finds the encrypted tally T_j for each candidate by calculating the product of all eligible encrypted votes casted for this candidate.

$$T_j = \prod_{i=1}^n C_{i,2} \quad (27)$$

2. Result Decryption: The `projvoting.paillierp.DecryptedInteger(Tj)` class decrypts the final tally T_j for each candidate using the voting private key K_v^- . By this step, the final election results are found, reported in final result report, and the election process finishes.

$$R_j = K_v^-(T_j) = \sum_{i=1}^n V_{i,j} \quad (28)$$

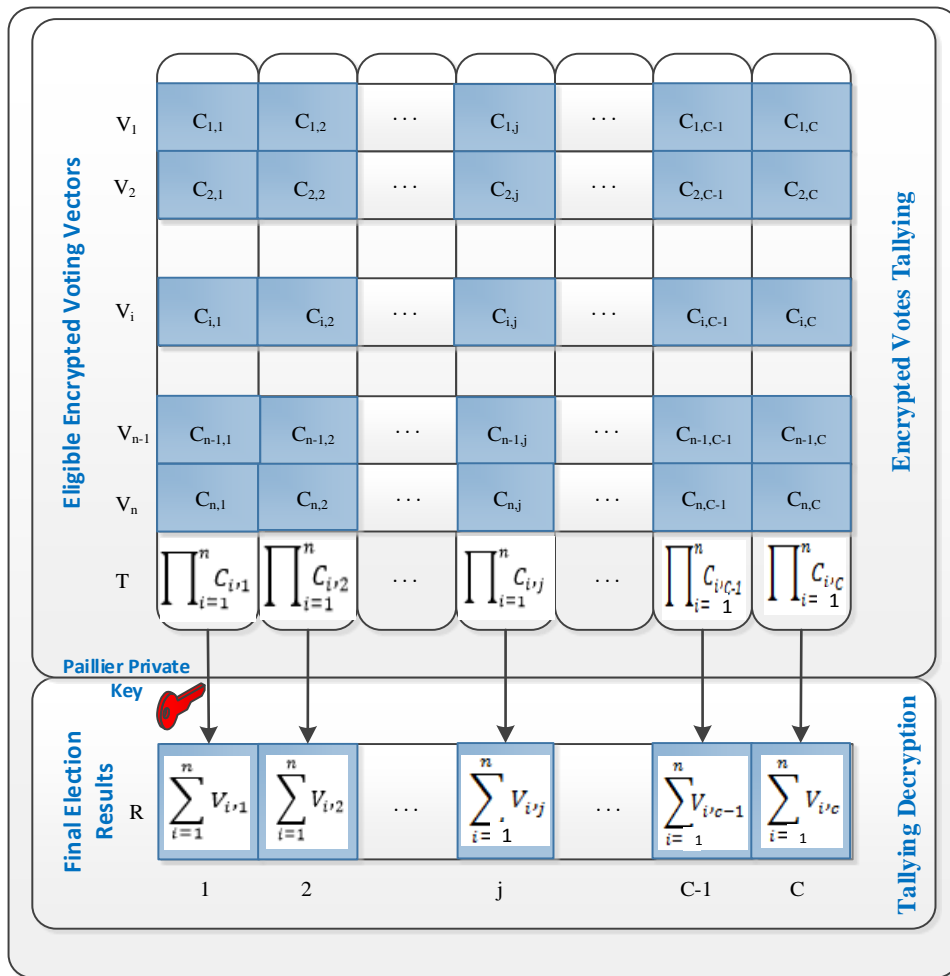


Figure 11: Tallying Process and Final Tally's Decryption

4.4 Verification and Tallying Processes Parallel Implementations

Java multithreading programming is used to speed up the verification and tallying processes. As there are no dependencies among the verification checks, they can be executed in parallel. Additionally, each kiosk package can be performed in parallel and all results can be aggregated together. Consequently, three parallel schemes are implemented and evaluated: task, master/slave, and data schemes. The following subsections describe these schemes

4.4.1. Task Parallelism Scheme

The ballot verification process has multiple checks (tasks): voter eligibility check, multiple voting check, ballot authenticity check, and vote validity check. When you think in parallel execution, you start with dividing the problem into tasks. Accordingly, the vote verification checks are divided into tasks. Each task is responsible for one check and runs through a separate thread with the intention that all threads work on the entire data set, but each thread does a specific task.

This scheme is known as task decomposition. The verification function is divided into four separated sub functions and the kiosk packages (ballots) are given to all threads for processing. The final tallying process starts after the finish of the last running thread.

Figure 12 clarifies this scheme, where `projvoting.MainRunThread2()` class spawns the following threads:

- a) The `projvoting.RunThreadCheckValidNID(Voter IDs)` thread to perform the task of voter validity and multiple voting (hash table lookup) checks.
- b) The `projvoting.RunThreadCheckValidData(Receipts)` thread to perform the task of ballot authenticity (digital signature) check.

- c) The `projvoting.RunThreadCheckVerification2(Encrypted Votes, {0,1} Proofs)` thread to perform the task of vote validity ($\{0,1\}$ ZK proof) check.
- d) The `projvoting.RunThreadCheckCalcSum(Encrypted Votes Sum, {0} Proof)` thread to perform the task of ballot validity ($\{0\}$ ZK proof) check.

The `projvoting.MainRunThread2Calc(Eligible Encrypted Votes)` class starts at the finish of the last verification thread run. This class calls `projvoting.RunThreadCheckCalcCol(Eligible Encrypted Votes)` to perform the final tallying process. The election process ends at the completion of this task.

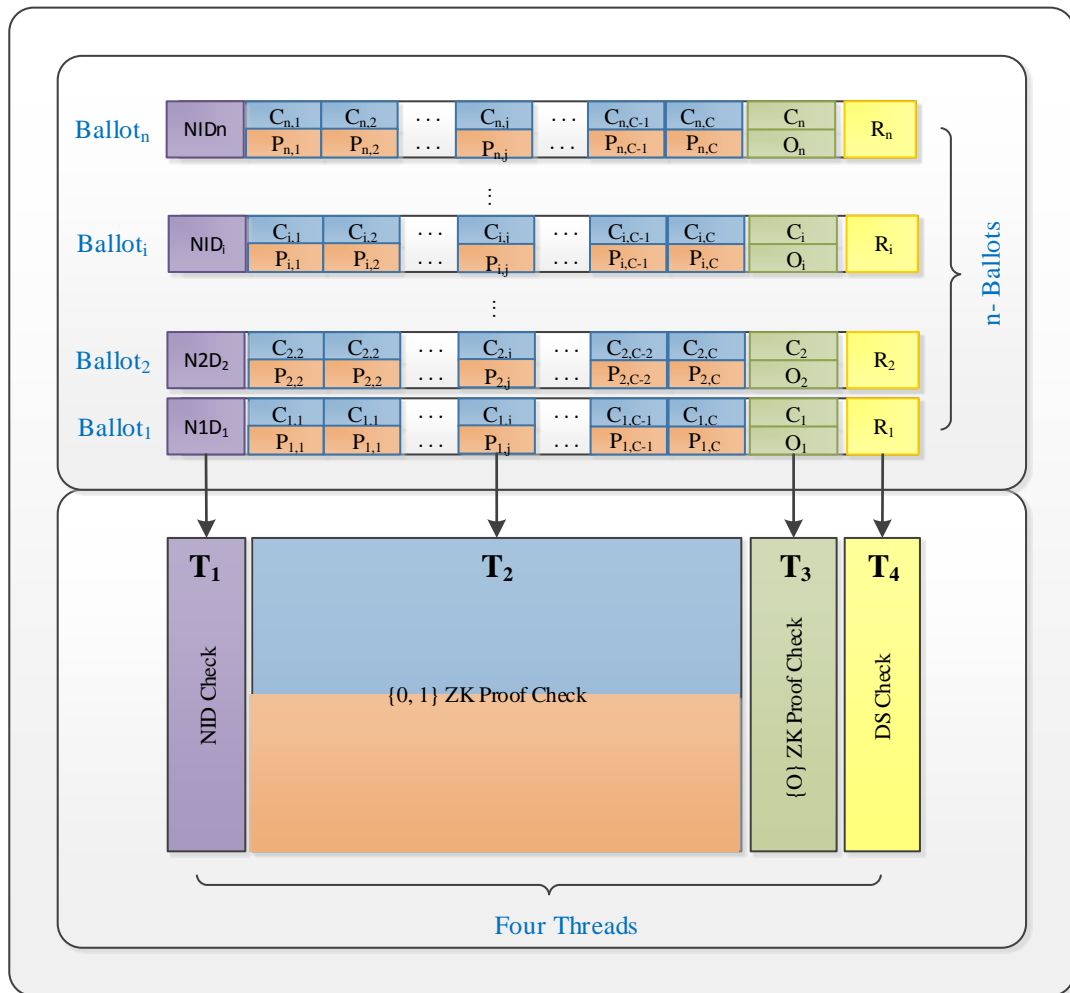


Figure 12: Task Parallelism Scheme

4.4.2. Master/Slave Parallelism Scheme

In this Master/Slave parallelism scheme. The `projvoting.MainRunThread()` server class spawns one client thread to handle every kiosk package request. This is some kind of data parallelism in which data is partitioned to a large the number of threads and each thread handles one kiosk package. Each of the `projvoting.RunThreadCheckVerfication(Kiosk Package)` client thread works on the kiosk package independently, and exists when it is done.

Figure 13 clarifies this scheme where each thread performs all vote verification checks on a single kiosk package. `projvoting.MainRunThreadCalc()` class starts at the finish of the last client thread. This class calls `projvoting.RunThreadCheckCalcCol(Eligible Encrypted Votes)` to perform the final tallying process. The election process ends at the completion of this task.

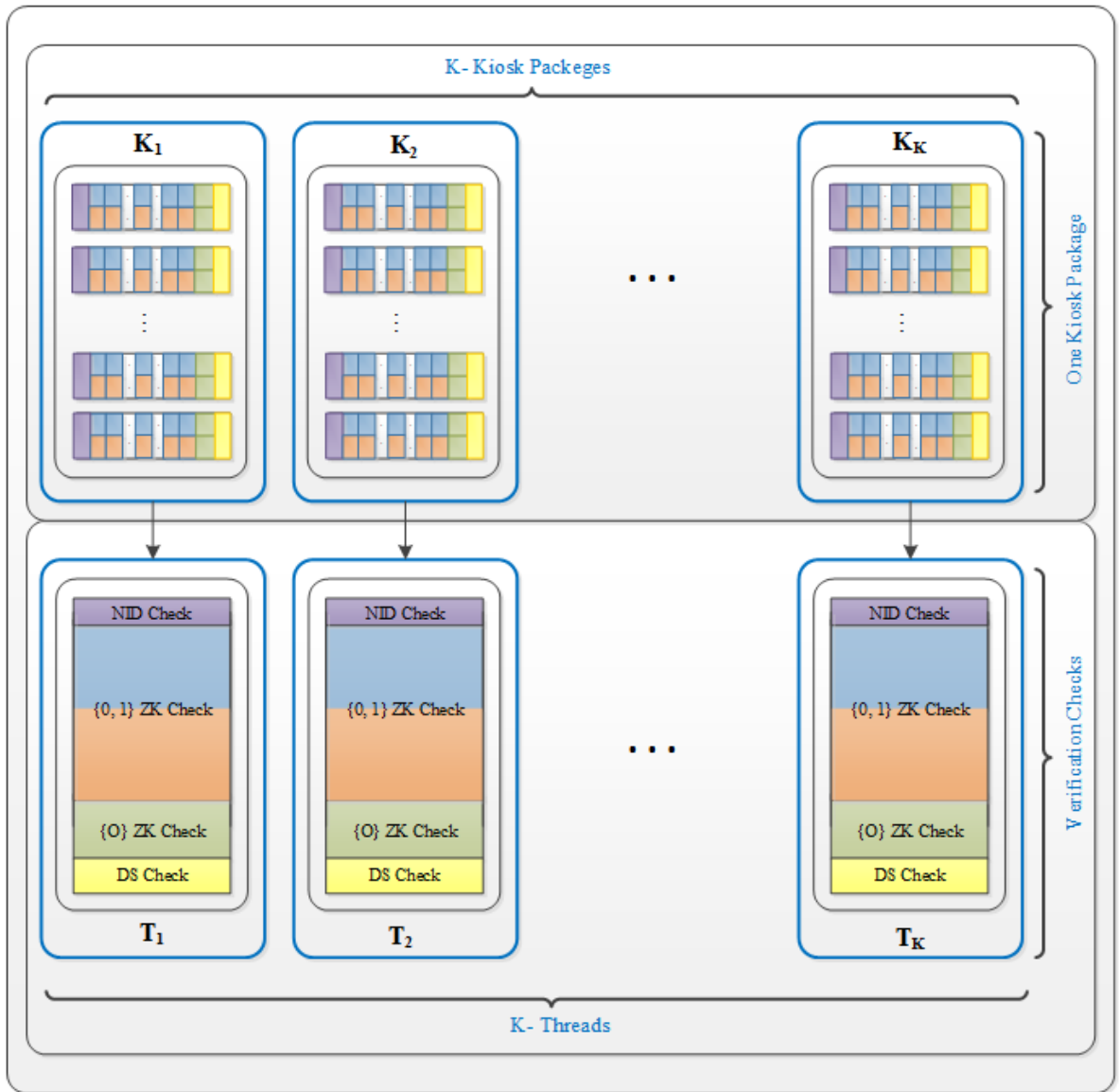


Figure 13: Master/Slave Parallelism Scheme

4.4.3. Data Parallelism Scheme

When having many kiosk packages (data) to process, we can divide this large set of data among multiple threads. This concept is known as data parallelism in which each thread does the same work but on its subset of data. Supercomputers have excelled at for years. In presence of this, the numbers of simultaneously running threads will be equivalent to the numbers of physical cores to get higher efficiency. Each thread performs the verification and tallying processes as the way as the sequential implementation does but on a sub set of data. The kiosk packages (data) will be dynamically distributed among the different threads in a round robin manner during the run time.

Figure 14 clarifies this scheme. The `projvoting.MainRunThreadVerification()` class spawns `projvoting.RunThreadVerification()` threads as many as the `GlobalConstant.NumThread` set. Each running thread asks `projvoting.GetKiosPackage()` class for an available kiosk package to process. Accordingly, this class updates the global kiosk counter and gives the requester thread an available package. The kiosk assignment step is synchronously executed so that only one thread can be served at a time to keep data consistency.

Java `lock()` and `unlock()` method are used to control the access to this shared resource by the multiple threads. Commonly, `lock()` grants on thread at a time an exclusive access to kiosk assignment procedure, and it is released for another thread by `unlock()`. Finally, by the end of the last kiosk package processing, the electoral process ends.

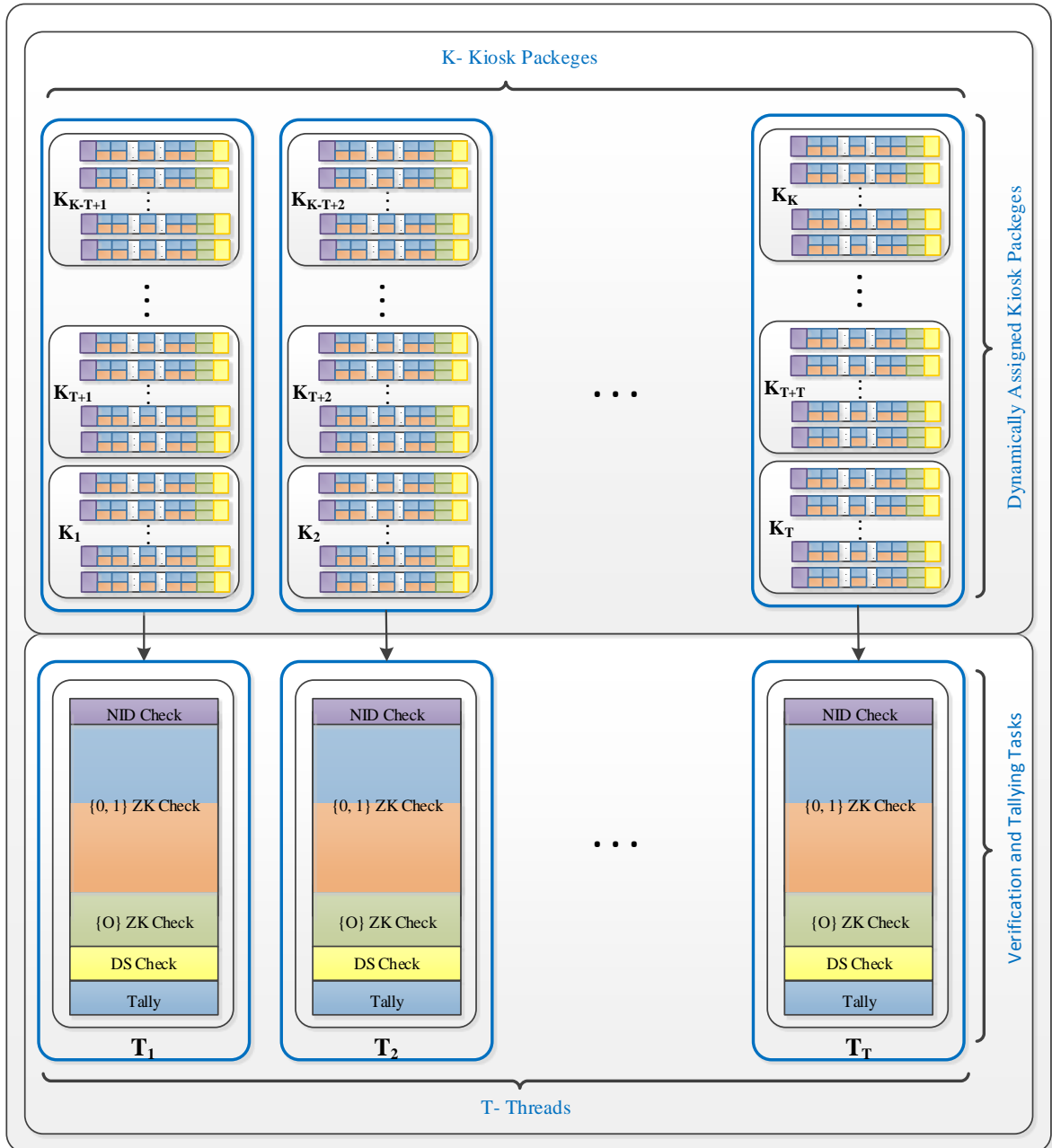


Figure 14: Data Parallelism Scheme

5. Experimental Evaluation Results and Discussions

This chapter evaluates the three parallelism schemes implemented in the last chapter. It starts with the experimental setup then continued with the conducted experiments.

5.1 Experimental Setup

All experiments were done on a host with four Intel Xeon processors E7-8837, each with eight cores, 2.67 GHz clock rate, and has an access to 256 GB memory. All implemented cryptography, classes, threads, and methods are coded in the *projvoting* java project which is based on JRE 1.7 standard java libraries and Paillier cryptography thep java project (thep, 2011). This project is compiled using GNU Compiler for Java (GCJ) version 4.8 and runs on an Ubuntu 12.04 virtual machine. The host is windows server 2012 R2 datacenter edition -x64 bit. The virtual machine monitor is Microsoft Hyper-v 2012 R2. This host serves only the Ubuntu virtual machine in which the experiments are run. This virtual machine has access to 160 GB memory. The number of assigned cores varies from 4 to 32 according to the experiment. The number of ballots in a kiosk package is 500 ballots. The number of candidates is sixteen unless it is configured otherwise. The number of options is four unless it is configured otherwise. The ballot's storage size is 52 KB. The size of each encrypted vote $C_{i,j}$ along with its ZK proofs values $\{e, uvals, evals, \& vlals\}$ are 3KB, the encrypted number of options and its ZK proof values are 3 KB, and the digital signature size is 1 KB. Accordingly, the storage requirements for each ballot is given by:

$$Ballot\ Size\ (KB) = 3C + 3 + 1 = 3C + 4 \quad (29)$$

Additionally, `Runtime.getMemory()` java instrumentations are used to estimate the memory usage for verifying a ballot. 273 KB is occupied to verify a ballot and 308 MB for creating the NID hash table with 2,000,000 entry.

5.2 Sequential Implementation Results

The sequential implementation of the verification and tallying processes is evaluated in terms of sequential execution time as a function of the number of ballots.

Table 1 summarizes the performance measures of executing the verification and tallying processes serially. Additionally, it summarizes the distribution of the execution time among the processes tasks. Figure 15 shows that there is a direct linear relationship between the execution time and the number of ballots as any homomorphic e-voting systems

Table 1: Sequential Execution Time Distributed

No. of Ballots	Total	Eligibility	Validity	Authenticity	Tallying
	$t_s(\text{hr})$	$t_s(\text{hr})$	$t_s(\text{hr})$	$t_s(\text{hr})$	$t_s(\text{hr})$
5,000	1.6	6×10^{-4}	1.51	0.02	0.05
10,000	3.1	8×10^{-4}	2.96	0.02	0.10
15,000	4.6	8×10^{-4}	4.49	0.01	0.14
20,000	6.3	11×10^{-4}	6.06	0.01	0.20

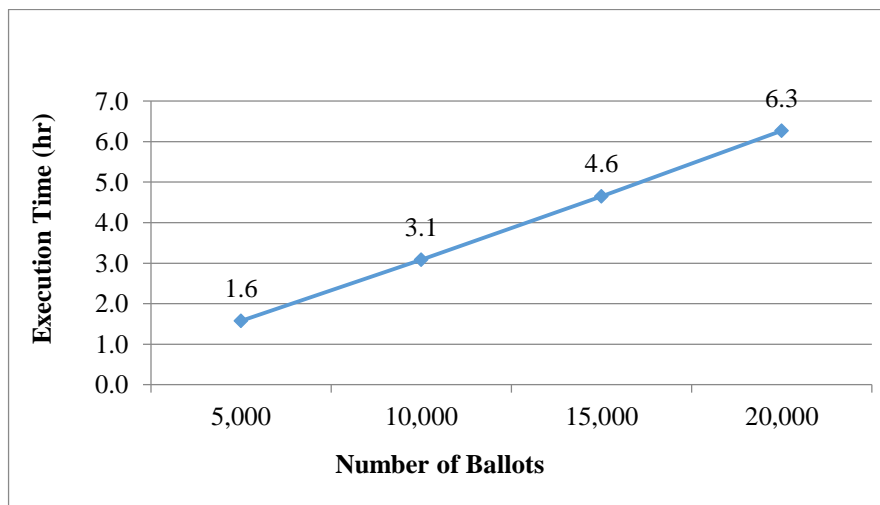


Figure 15: Overall Sequential Execution Time

Additionally, Figure 16 shows that tallying takes negligible time compared to the verification. Furthermore, it shows that about 95% of time is spent in verifying the vote validity which involves checking the ZK proofs. This emphasizes the long ZK proof computations.

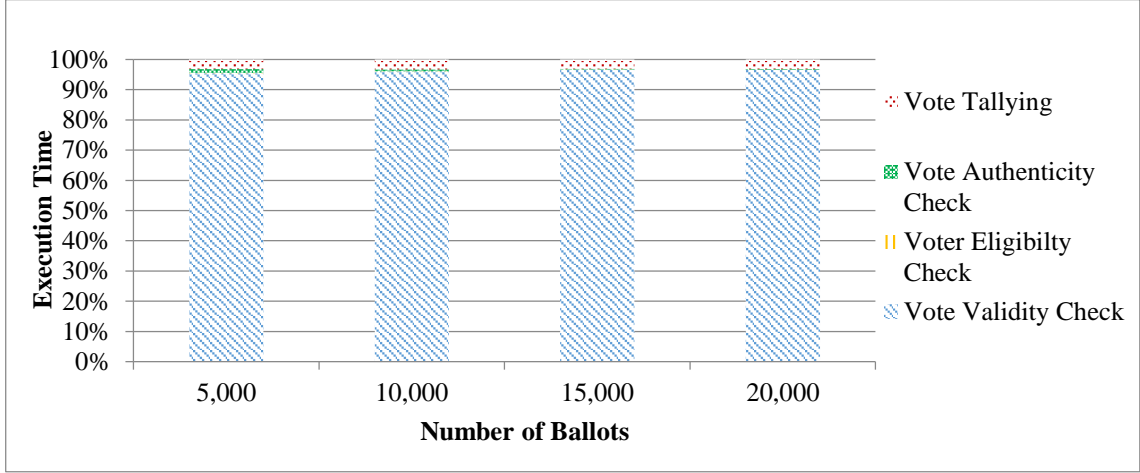


Figure 16: Serial Execution Time Distribution

5.3 Alternative Parallel Implementation Results

The previous sequential implementation evaluation suggests that the verification time linearly increases with more voters. Therefore, parallel implementation is required to get the final tally in an acceptable time by exploiting the potentials of parallel processing computers.

We present here the result of evaluating the three parallel schemes: Task, Data, and master/slave schemes. We evaluate the cost of each parallel implementation scheme with respect to the sequential one. Table 2 summarizes the running cost of the three schemes on a virtual machine with 4 cores and the number of ballots $B = 20,000$.

Table 2: Performance Time Evaluation of the Parallel Schemes with four cores

Scheme	t_p (hr)	Speedup	Efficiency	Cost
Sequential	6.27			6.27
Task	5.89	1.06	27%	23.55
Data	1.74	3.61	90%	6.95
Client-Server	1.71	3.66	91%	6.85

Figure 17 show that the task parallelism does not give good speedup in addition to its expensive cost and poor efficiency. This scheme not able to divide the problem into equivalent tasks, the load is unbalanced because the ZK proof check still run sequentially and takes very long time compared to the others. It gives only a slight speedup. In this scheme, all checks finish much earlier than the ZK proof check.

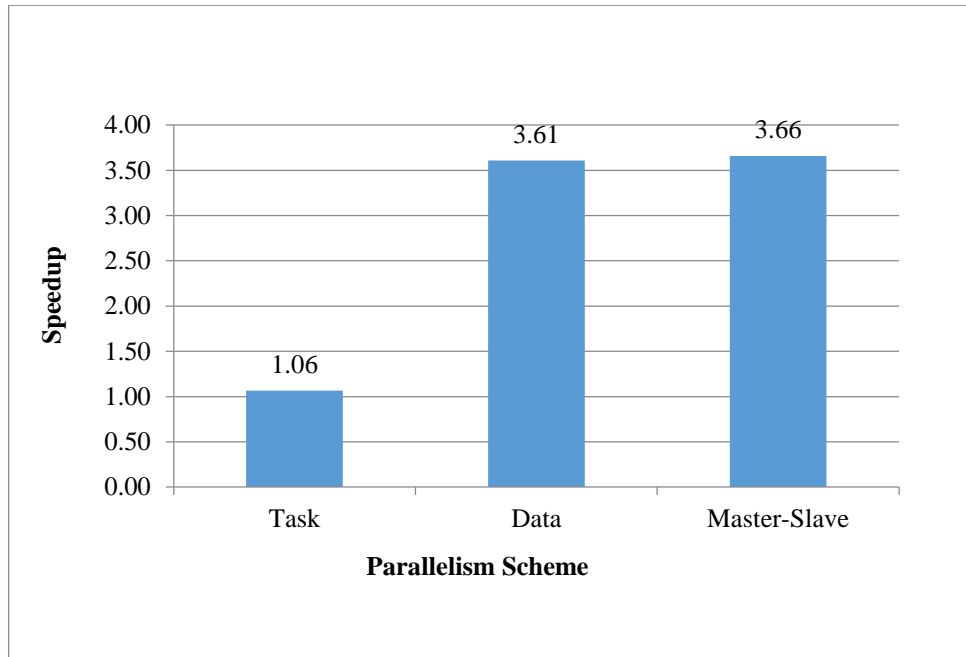


Figure 17: Parallel Implementation Speedups with Four Cores

On the other hand, the data and master/server schemes give good speedups, high efficiencies, and low running costs. Although the master/slave scheme has the best performance for this set of data in addition to its simplicity and directness, it is not a scalable solution. The cost of thread creation increases as the number of threads increases and the OS will straggle with large number of threads. The excessive number of threads leads to performance degradation due to their competition on the limited number of cores, available memory and other resources. Running this program on a large set of data will crash the system when it runs out of virtual memory.

Therefore, writing the program with a limit on the number of threads would be best for handling the problem in order to control the number of threads. As a result, the proposed data parallelism scheme is the most preferable one. Beside its acceptable performance, it limits the number of spawned threads, proportionally maps the thread to physical cores. It dynamically balanced system as the workload is distributed among the spawned threads in round robin manner.

5.4 Data Parallelism Speedup Evaluation

The data parallelism implementation of the verification and tallying processes is evaluated according to the number of running cores so the number of threads respectively. Table 3 summarizes the running cost when the number of threads is increased. The experiment is performed on a number of ballots $B = 64,000$.

Table 3: Data Parallelism Speedup Evaluation

Cores	$t_p(\text{hr})$	Speedup	Efficiency	Cost
Serial	19.50			19.50
4	4.99	3.9	98%	19.95
8	2.66	7.3	91%	21.32
16	1.32	14.8	92%	21.13
32	0.73	27.5	86%	22.68

Table 3 and Figure 18 shows that we can get sub-linear speedup improvement as we increase the number of cores. The number of threads are increased to match the number of available cores. Thus, the assigned workload becomes smaller as the number of threads increased and reduces the overall computation time. Based on this, we can determine the hardware specifications requirements for serving a number of ballots within 8 hours.

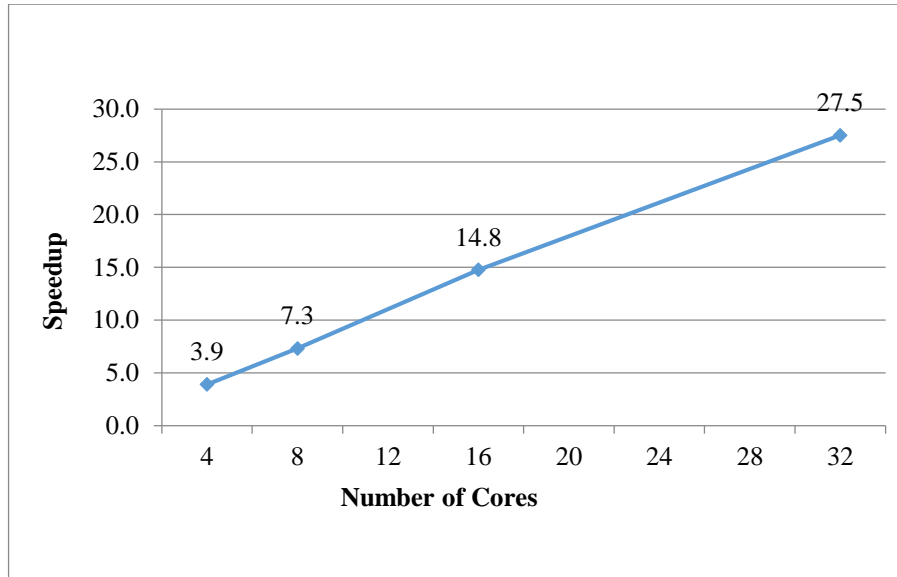


Figure 18: Data Parallelism Speedup

On the other hand, when the number of threads is increased, the synchronization delay becomes larger and the contention on the memory and storage increases. In average, each thread will ask for a kiosk package K/T times. The thread waits up to T time units till it has an exclusive access to kiosk assignment procedure. Thus, each thread has K overhead time. For this reason, Figure 19 shows a slight degradation in parallelization efficiency as the number of threads increases due to the synchronization of kiosk assignment.

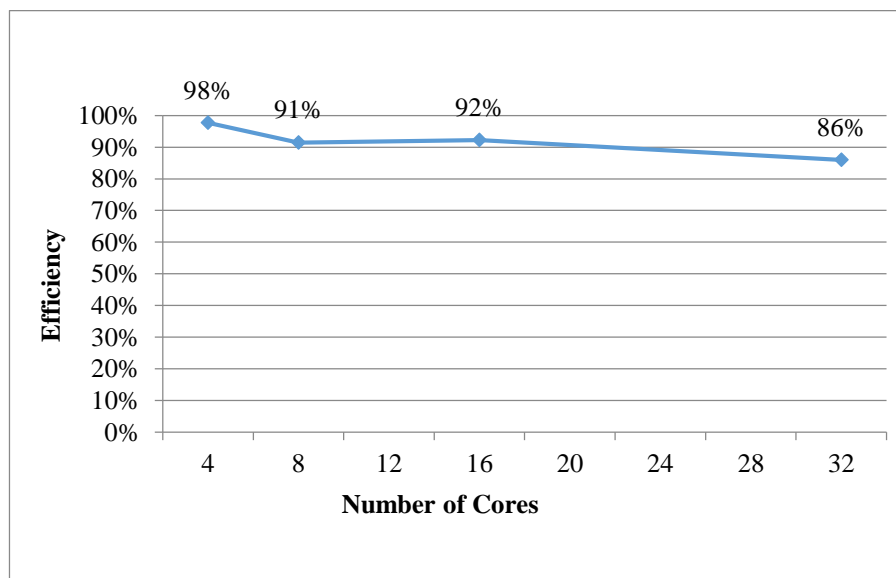


Figure 19: Data Parallelism Efficiency

Moreover, a set of experiments are conducted to check the scalability of data parallelism scheme based on the isoefficiency metric. Figure 20 plots speedup against number of cores for different values of B up to 32 cores.

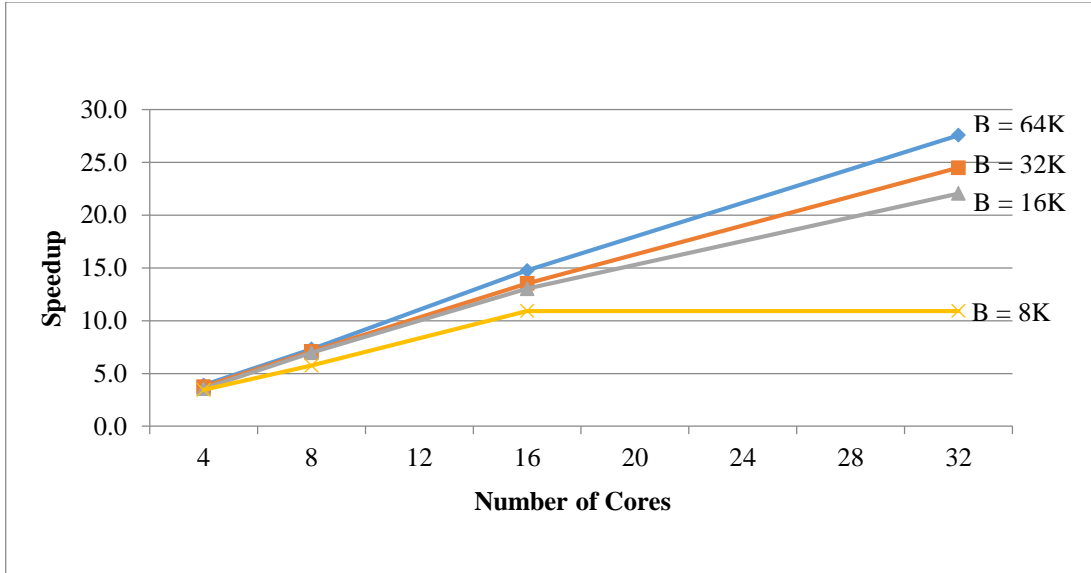


Figure 20: Data Parallelism Scheme Isoefficiency Metric

Table 4: Data Parallelism Scheme Isoefficiency

Core →	4	8	16	32
Ballots ↓				
8K	86%	72%	68%	34%
16K	89%	87%	81%	69%
32K	94%	89%	85%	77%
64K	98%	91%	92%	86%

Figure 20 and Table 4 illustrate two things. First, for a given problem instance, the speedup does not keep the linear increase as the number of cores increases beyond the assigned workload. The speedup curve tends to saturate as in the instance of processing 8,000 ballots on 32 cores. In other words, the efficiency drops with increasing the number of cores. Second, a larger number of B yields higher efficiency for the same number cores.

Given that increasing the number of cores reduces efficiency and increasing the size of the computation increases efficiency, it should be possible to keep the efficiency

constant by increasing both the size of the problem and the number of cores simultaneously to consider that the parallel system is scalable. For instance, in Table 4, the efficiency of verifying and tallying 8,000 ballots on tallying machine with four cores is 86%. If the number of cores is increased to eight, the number of ballots is scaled up to verify and tally 16,000 ballots the efficiency remains in the average of 86%.

Accordingly, the S-Vote with data parallelism is scalable since the efficiency of data parallel execution maintained at a constant value by simultaneously increasing the number of cores and the number of ballots being verified.

5.5 Studying the Effect of Number of Candidates and Options

As another point of view, we study the factors that may affect the zero knowledge proof itself. The next two experiments are performed to study the effects of numbers of the candidates and options on the ZK proof check.

The experiment is running on 32 cores using the data parallelism scheme, number of ballots $B = 64,000$, and number of options $O = 2$

Table 5: Number of Candidates Effect on ZK Proof

No. of Candidate	Time _p (min)
4	17.7
8	27.3
12	37.0
16	44.5

Table 4 and Figure 20 show that the ZK proof has a direct linear relationship with the number of candidates. The ZK proof is a function of the number of votes per ballot. This true as the ZK proof is conducted for each encrypted vote within the ballot and its execution time proportioned to the increase number of votes within the ballots.

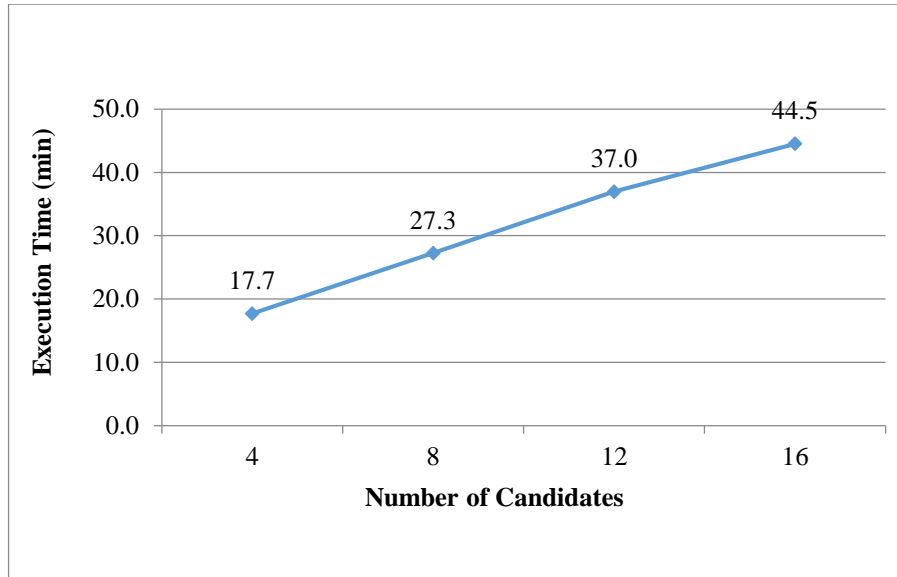


Figure 21: Number of Candidates Effect on the ZK Proof Check Time

On the other hand, the number of options per ballot does not affect the ZK proof as only one proof is required to ensure that there is O option in the ballots for each ballot regardless what is the value of O is. The experiment is running on 32 cores and number of ballots $B = 64,000$.

Figure 21 shows that the number of options does not affect the ZK proof time since it runs once per ballot. Only the {O} set need to be updated to correctly perform this check.

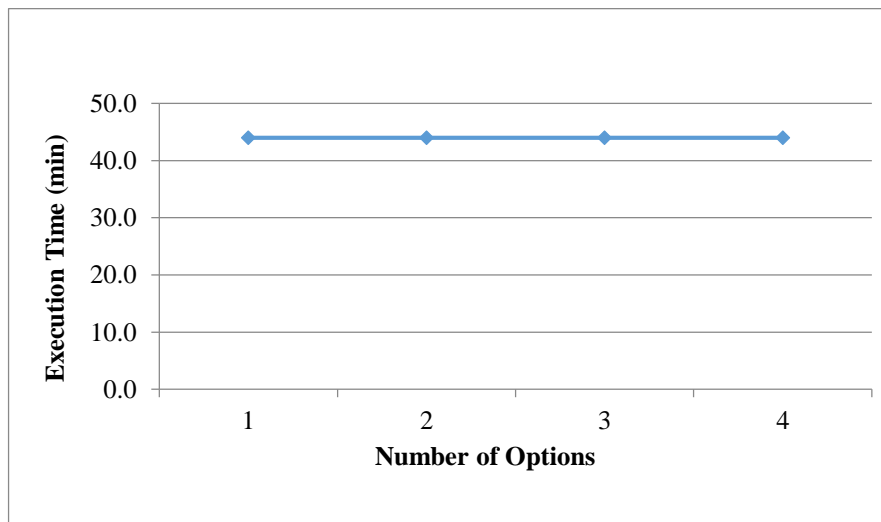


Figure 22: Number of Options Effect on the ZK Proof

5.6 Discussion

The cost of ballot verification is clearly dependent on number of ballots. Data parallelism scheme offers the capability of reducing its cost for an acceptable time. The parallel processing degree can be determined according to the election process size. For example; Table 3 results illustrate that the kiosk package can be processed in about 11 minutes. Using data parallelism scheme, a 128 cores machine can verify and tally 2 million ballots with 16 candidates each in a country like Jordan in 5.7 hour $((2,000,000/500)*11)/(128*60) = 5.7$ hours. Larger elections require larger servers. Faster and larger servers would decrease this time even further.

6. Conclusion and Future Work

In this thesis, we adopted the S-Vote homomorphic e-voting system for its e-voting requirements satisfaction. S-Vote uses public key cryptography, hashing techniques, homomorphic cryptography, and zero knowledge proofs for achieving the e-voting requirements of privacy, authentication, and validation of data integrity. The implemented vote casting, verification, and tallying processes efficiently permit testing the corresponding technologies and processes used in S-Vote.

The implemented processes use the Paillier homomorphic cryptography and ZK proofs java project for encrypting the casted votes, keeping their privacy, and decrypting their final encrypted tallies. The homomorphic cryptosystem allows tallying the ballots without decrypting them, thus preserving the voters' privacy. A hash table data structure is used for creating the eligible voters list for its efficient search feature. Standard SHA-256 hash function and RSA public key cryptography java libraries are used for authenticating/signing the encrypted ballots and for generating the voting receipts. These receipts prevent multiple voting and allow voters to verify that their votes have reached the final tally.

The sequential implementation shows that the verification process has long execution time and has a linear relationship with the number of voters. This thesis uses parallel implementation to reduce the ballots verification time. It presents and evaluates three parallel schemes for ballot verification and tallying: task, master/slave, and data. It uses java-multithreading techniques to exploit parallelism of the three problem decomposition schemes. Function decomposition gives inefficient task parallelism scheme as it unbalances the workload among the verification checks (tasks). Thread per request decomposition is used in the master/slave scheme. New thread is spawned to serve each kiosk package. This scheme may crash the system as the number of ballots

increases for the threads competition on the limited hardware resources. Domain decomposition is used in the data parallelism scheme in which the number of threads is relative to the number of physical cores and the kiosk packages are dynamically distributed across the spawned threads.

The three parallelism schemes are evaluated against the sequential one. It shows that the data parallelism is the winner scheme in speeding up the verification process. The analysis shows that data parallelism speedup has direct correlation with the number of physical cores. An increase in the number of cores; number of spawned threads, results in a decrease in the time required for performing vote verification checks. It can verify and tally 64,000 ballots in about 44 minutes and with 86% efficiency when using 32 threads running on the multi-core tallying machine of 32 cores. So that, we can determine the hardware size required to verify and tally a number of ballots within an eight hours. For example, we can verify and tally two millions ballots for a country as big as Jordan in about 5.7 hours using 128 cores. The analysis also shows that the ZK proof is a function of number of candidates per ballots and is not affected by the number of options per ballot. As a result, Data parallelism scheme enhances e-voting systems' acceptance, reduces the costs of electoral process compared with the paper-based ones, and complies with e-voting systems requirements.

My directions for future work, is to extend our work to a full implementation of S-Vote system. Beside of the parallel implementation for vote validity and authenticity checks, we would like to cover all aspects of the S-Vote proposed components including the distributed key generation, threshold cryptography, kiosk design, and smartcard implementations.

References

- Abandah, G. Darabkh, K. Ammari, T. and Qunsul, O. (2014), Secure National Electronic Voting System. **Journal of Information Science and Engineering**, 30, 1339-1364.
- Adida, B. (2006), Advances in Cryptographic Voting Systems. **Doctoral Dissertation, Massachusetts Institute of Technology, Cambridge.**
- Allansson, M. Baumann, J. Taub, S. Themner, L. and Wallensteen, P. (2012), The first year of the Arab Spring. **SIPRI Yearbook**, 45-56.
- Andrew Neff, C. (2003), Verifiable mixing (shuffling) of elgamal pairs, retrieved from **<http://theory.lcs.mit.edu/rivest/voting/papers/Neff-2004-04-21-ElGamalShuffles.pdf>**
- Andrews G. (2000), **Foundations of Multithreaded, Parallel, and Distributed Programming.** Addison Wesley, Reading, MA.
- Antonyan, T. Davtyan, S. Kentros, S. Kiayias, A. Michel, L. Nicolaou, N. Russell, A. and Shvartsman, A. (2007), State-wide elections, optical scan voting systems, and the pursuit of integrity. **IEEE Trans. Information Forensics and Security**, (4), 597–610.
- Barney, B. (2012). **Introduction to parallel computing.** Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/parallel_comp/.
- Boneh, D. and Franklin, M. (2001), Efficient generation of shared RSA keys. **Association for Computing Machinery ACM digital library**, (4), 702-722.
- Baudron, O. Fouque, P. Pointcheval, D. Stern, J. and Poupard, G. (2001), Practical multi candidate election system. **Principles of distributed computing (PODC 01)**, New York, USA 1st August, 2001, 274–283.
- Benaloh, J. (1987), Verifiable Secret-Ballot Elections, Doctoral Dissertation. **Yale University, New Haven.**
- Boneh, D. Goh, J. and Nissim, K. (2005), Evaluating 2-DNF formulas on ciphertexts. **Springer Berlin Heidelberg**, 3378, 325–342.
- Catalano, D. Juels, A. and Jakobsson, M. (2005), Coercion-resistant electronic elections. **Computer and Communications Security**, New York, USA 7 November, 2005, 61–70.
- Chapman, B. Jost, G. and van der Pas, R. (2007), Using OpenMP: Portable Shared Memory Parallel Programming. **Journal of Computer Science & Technology.**
- Chida, K. and Yamamoto, G. (2008), Batch processing for proofs of partial knowledge and its applications. **IEICE Trans. Fundamentals E91CA**, (1), 150–159.
- Clarkson, M. Chong, S. and Myers. A. (2008), Civitas: Toward a secure voting system. Security and Privacy, **Security and Privacy, SP 2008. IEEE**, Oakland, CA 18-22 May, 2008, 354 – 368.

- Cramer, R. Damgård, I. and Schoenmakers, B. (1994), Proofs of partial knowledge and simplified design of witness hiding protocols. **Springer Berlin Heidelberg**, » **Lecture Notes in Computer Science**, 839, 174-187.
- Cramer, R. Gennaro, R. and Schoenmakers B. (1997), A secure and optimally Efficient Multi-Authority Election Scheme. **Springer-Verlag**, Eurocrypt 97, 113 118.
- Cormen, T. Leiserson, C. Rivest, R. Stein, C. (2001), **Introduction to Algorithms**. (2nd ed.). MIT Press and McGraw-Hill.
- Damgaard, I. and Jurik, M. (2001), A Generalization, A Simplification and Some Applications of Paillier's Probabilistic Public-Key System. **Public Key Cryptography PKC**, 119-136.
- Demaine, E. Lind, J. (2003), **Advanced Data Structures**. MIT Computer Science and Artificial Intelligence Laboratory. Spring.
- De Santis, A. Di Crescenzo, G. Ostrovsky, R. Persiano, G. and Sahai, A. (2001), Robust Non-interactive Zero-Knowledge. **Springer Berlin Heidelberg**, 2139, 566-598.
- Diaz, J. Munoz-Caro, C. and Nino, A. (2012), A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era. **IEEE Trans. Parallel and Distributed Systems**, (23), 1369-1386.
- Donald, K. (1998), The Art of Computer Programming. (2nd ed.). **Addison-Wesley**.
- Dongarra, J. Foster, I. Fox, G. Gropp, W. Kennedy, K. Torczon L. and White, A. (2003), The Sourcebook of Parallel Computing. **Morgan Kaufmann Publishers, San Francisco**, 491–541.
- Feng, W. and Balaji, P. (2009), Tools and environments for Multicore and Many-core Architectures. **IEEE Computer Society**, 26-27.
- Fiat A. and Shamir, A. (1987), How to prove yourself: practical solutions to identification and signature problems. **Springer Berlin Heidelberg**, **Advances in Cryptology**, 263,186-194.
- Foster, I. (1995), **Designing and Building Parallel Programs**. Addison Wesley, Reading.
- Fouque, P. Poupard, G. and Stern, J. (2001), Sharing decryption in the context of voting or lotteries, **4th International Conference, FC 2000**, Anguilla, British West Indies, 20–24 February, 2000, 90-104.
- Goldreich, O. Sahai, A. and Vadhan, S. (1998), Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge, **ACM Theory of computing**, New York, USA 23 May, 1998, 399-408.
- Grama, A. Gupta, A. and Kumar, V. (1993), Isoefficiency: Measuring the scalability of parallel algorithms and architectures, **IEEE Parallel & Distrib. Technol**, (1), 12-21.
- Groth, J. (2003), A verifiable secret shuffle of homomorphic encryptions, Desmedt, Y.G. (ed.) **PKC LNCS, Springer, Heidelberg**, 145–160.

- Groth, J. (2005), Non-interactive zero-knowledge arguments for voting, Ioannidis. **Springer**, Heidelberg, Third International Conference, ACNS, New York, USA, 7-10 June, 2005, 467–482.
- Hennessy, J. Patterson, D. and Larus, J. (1999), **Computer organization and design: the hardware/software interface**. (2nd, 3rd print. ed.). San Francisco: Kaufmann. ISBN 1-55860-428-6.
- Hirt, M. and Sako, K. (2000), Efficient Receipt-Free Voting Based on Homomorphic Encryption. **International Conference on the Theory and Application of Cryptographic Techniques**, Bruges, Belgium, 14–18 May, 2000, 539-556.
- Husztai, A. (2011), A homomorphic encryption-based secure electronic voting scheme. **Publ. Math. Debrecen** 79, 479-496.
- Hwu, W. Keutzer, K. and Mattson, T. (2008), The concurrency challenge. **IEEE Design and Test of Computers**, (4), 312-320.
- Jacobsen, D. Thibault J. and Senocak, I. (2010), An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Cluster, **48th AIAA Aerospace Sciences Meeting and Exhibit**.
- James, M. Adler, Dai W. Richard L. Green, and Neff A. (2000) “Computational Details of the VoteHere Homomorphic Election System”. **Technical Report, VoteHere Inc**, Retrieved from <http://www.votehere.net/technicaldocs/hom.pdf>.
- Joaquim, R., Zúquete, A., and Ferreira, P. (2003), REVS—A robust electronic voting system. **IADIS Int’l Conf. e-Society**.
- Johnson, K. (2005), An open-secret voting system. **Computer**, 38, (3), 98-100.
- Karro, J. and Wang, J. (1999), Towards a practical, secure, and very large scale online election. **Annual Computer Security Applications Conf**, Phoenix, AZ, 06-10 Dec, 1999, 161–169.
- Kasim, H. March, V. Zhang R. and See S. (2008), Survey on Parallel Programming Model. **Proc. of the IFIP Int. Conf. on Network and Parallel Computing**, 5245, 266-275.
- Katz, J. Myers, S. and Ostrovsky, R. (2001), Cryptographic Counters and Applications to Electronic Voting. **Springer, Heidelberg, Advances in Cryptology** (01), 78-92.
- Kiayias, A. and Yung, M. (2002), Self-tallying Elections and Perfect Ballot Secrecy, Public Key Cryptography. **5th International Workshop-PKC**, Paris, France, 12–14 February, 2002, 141-158.
- Kikuchi, H., Hotta, S., Abe, K., and Nakanishi, S. (2000), Distributed auction servers resolving winner and winning bid without revealing privacy of bids. **Parallel and Distributed Systems: Workshops, Seventh International Conference**, Iwate, 4-7 Jul, 2000, 307–312. IEEE.

- Kirk, D. and Hwu, D. (2010), **Programming Massively Parallel Processors: A Hands-on Approach**. Morgan Kaufmann, San Francisco.
- Krste A. and et al. (2006), "The Landscape of Parallel Computing Research: A View from Berkeley. **University of California, Berkeley**, Technical Report No. (UCB/EECS-2006-183).
- Lee, B. and Kim, K. (2000), Receipt-free electronic voting through collaboration of voter and honest verifier. **JW-ISC 2000**, 101–108.
- Lee, B. and Kim, K. (2002), Receipt-free Electronic Voting Scheme with a Tamper- Resistant Randomizer. Information Security and Cryptology. ICISC 2002 in Lecture Notes in Computer Science, 2002. **Springer-Verlag**, 389-406.
- Lewis, B. and Berg, D. (1999), **Multithreaded Programming with Java Technology**. (1st ed.), Prentice Hall.
- Lipmaa, H. Asokan, N. and Niemi, V. (2003), Secure Vickrey auctions without threshold Trust. **Springer Berlin Heidelberg, Financial Cryptography**, 2357, 87-101.
- Loka,R, Feng, W. and Balaji P. (2010), Serial Computing is not dead,. **IEE Computer**, 8.
- Mattson, T. Sanders B. and Massingill, B. (2005), **Patterns for Parallel Programming**. Addison-Wesley Professional.
- Nishide, T. and Sakurai, K. (2011), Distributed Paillier cryptosystem without trusted Dealer. **Springer Berlin Heidelberg, Information Security Applications**, 6513, 44-60.
- Paillier, P. (1999), Public key cryptosystem based on composite degree residuosity classes. **Springer, Heidelberg**, 1592, 223-238.
- Patterson, A. and Hennessy, J. (1996), **Computer Architecture: A Quantitative Approach**, (2nd ed). Morgan Kaufmann.
- Peng, K. (2005), Efficient Proof of Vote Validity Without Honest Verifier Assumption in Homomorphic E-Voting. **KIPS (Journal of Infrmation Processing System)**.
- Peng, K. and Bao. B, (2009), Efficient vote validity check in homomorphic electronic voting. **Information Security and Cryptology, ICISC in Lecture Notes in Computer Science LNCS, Springer-Verlag**, 202-217
- Peng, K. and Bao, F. (2010), Efficient Proof of Validity of Votes in Homomorphic E-Voting. **NSS (International Conference on Network and System Security)**, 17-23.
- Peng, K. and Dawson, E. (2007), Efficient bid validity check in elgamal-based sealed-bid Eauction. **ISPEC 2007**. LNCS.
- Peng, k. Boyd, C. and Dawson, E. (2006), Batch verification of validity of bids in homomorphic e-auction. **Computer Communications**, vol. 29, (15): 2798-2805.

- Peng, K., Boyd, C. Dawson, E. and Lee, B.(2004), Multiplicative homomorphic e-voting. **Springer, Heidelberg, INDOCRYPT**. LNCS, 61–72.
- Peng, K., Boyd, C., and Dawson, E. (2007), Batch verification of validity of bids in homomorphic e-auction, **Springer Heidelberg**, 209–224.
- Pusukuri, K. Gupta, R. and Bhuyan, L. (2011), Thread reinforcer: Dynamically determining number of threads via OS level monitoring Workload Characterization (IISWC). **IEEE International Symposium**, 116 –125.
- Rauber T. and Runger, G. (2010), **Parallel programming for multicore and cluster systems**. Springer Conference, Berlin.
- Sarath, D. and Ainapurkar, M. (2014), **An improved parallel interactive Feige-Fiat-Shamir identification scheme with almost zero soundness error and complete zero-knowledge**. IEEE, Networks & Soft Computing (ICNSC) International Conference, Guntur, 252 – 257.
- Schneier, B. (2007), **Applied Cryptography: Protocols, Algorithms, and Source Code in C**. John Wiley & Sons.
- Schryen, G., and Rich, E. (2009), Security in large-scale internet elections: A retrospective analysis of elections in Estonia. The Netherlands, and Switzerland, IEEE Trans. **Information Forensics and Security**, 729–744.
- Silverman, R. (2002), Has the RSA algorithm been compromised as a result of Bernsteins paper?. **RSA Laboratories**, vol. 8.
- Sottile, M. Mattson, T. and Rasmussen C. (2010), **Introduction to Concurrency in Programming Languages**. CRC Press.
- Stern, J. (ed.) EUROCRYPT. LNCS, 223–238.. Participants of the Dagstuhl Conference (2007), Dagstuhl Conference in Frontiers of E-Voting.Dagstuhl ccord. **Retrieved from <http://www.dagstuhlaccord.org/>, 2007.**
- Sutter, H. and Larus, J. (2005), Software and the Concurrency Revolution, **ACM Queue**, vol. 3, (7): 54-62.
- Sutter, H. and Larus, J. (2005), **Software and the concurrency revolution**. ACM Queue, 3(7): 54–62.
- Thep (2011), The Homomorphic Encryption Project - Computation on Encrypted Data for the Masses..**<http://code.google.com/p/thehp/>**
- Wilkinson B. and Allen, M. (2004), **Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers**. (2nd ed.), Prentice Hall.
- Yasinsac A. and Bishop, M. (2008), of paper trails and voter receipts. **41st Annual Hawaii Int'l Conf. on System Sciences**, Waikoloa, HI, 7-10 Jan, 2008, 487-487.

التحقيق في إيجاد طرق فاعلة لعمليتي الفرز والتأكد من صحة الأصوات الإلكترونية باستخدام التقنيات والبرمجيات المتوازية

إعداد

إسراء أحمد حسن سعادة

المشرف

الدكتور غيث عبدة

ملخص

تتناول هذه الرسالة موضوع استخدام أنظمة التصويت الإلكتروني لخدمة الانتخابات بأنواعها عوضاً عن استخدام الأنظمة الورقية. توفر الأنظمة الإلكترونية الدقة والكفاءة في العمليات الانتخابية فضلاً عن تحقيقها للمتطلبات الأمنية بحيث لا يتمكن أي شخص أو جهة من العبث بعمليات أو معلومات أو نتائج هذه الأنظمة.

تبنت هذه الرسالة نظام التصويت الإلكتروني الوطني الآمن (S-Vote) لتلبيته للمتطلبات الواجب توفرها في أنظمة التصويت الإلكترونية، واستخدامه التقنيات الحديثة التي تجعل التصويت الإلكتروني حلاً ممكناً وآمناً، واتباع عملياته لإجراءات آمنة، وتطبيقه بالحفاظ على الخصوصية التي تضمن عدم إمكانية كشف أي صوت لأي ناخب وذلك لتلافي شراء الأصوات.

يعتمد نظام الـ S-Vote على تقنية تشفير تمكنه من فرز الأصوات دون الحاجة إلى فك تشفير أي منها للحفاظ على الخصوصية. إن استخدامه لهذه التقنية يستوجب اعتماده على فحص $zero\ knowledge\ (ZK)\ proof$ الذي يقوم بدوره بالتأكد من صحة أصوات الناخبين دون الحاجة إلى الكشف عن أي منها وضمان وصول الأصوات الصحيحة فقط لعملية العد النهائية. بالرغم من فعالية فحص ZK بالحفاظ على الدقة والخصوصية إلا أن تعقيد وطول العمليات الحسابية التي يمر بها يحد من استخدام أنظمة التصويت الإلكترونية لعمليات انتخابية كبيرة كالانتخابات القومية أو النيابية واقتصارها على انتخابات صغيرة مثل الانتخابات البلدية والطلابية.

سعت هذه الرسالة إلى استخدام تقنيات الحاسوب المتوازية لإجراء كل من عملية التحقق من صحة أصوات الناخبين وعملية فرز الأصوات الصحيحة وذلك للتقليل من الوقت المستغرق في إجراء فحص التأكد من صحة الأصوات ($ZK\ proof$). تقوم عملية التحقق من الأصوات على إجراء ثلاثة فحوصات: فحص التأكد من هوية الناخب وأنه ممن يحق لهم التصويت وأنه قام بالتصويت لمرة واحدة فقط، فحص التأكد من صحة تصويت الناخب ($ZK\ proof$)، وفحص التأكد من صحة التوقيع الإلكتروني للناخب.

قدمت الرسالة ثلاث طرق مختلفة لتنفيذ عمليتي فحص وفرز الأصوات باستخدام التنفيذ المتعدد للخيوط الخاصة بلغة البرمجة جافا ($Java\ Multithreading$) والاستفادة من وحدات المعالجة المركزية المتعددة النوى ($Multicore\ Computers$) لتنفيذ الخيوط أو سلاسل

التعليمات بشكل متزامن. تعرف الطريقة الأولى بـ Task Parallelism Scheme حيث يتم تقسيم عملية الفحص إلى وظائف منفصلة تقوم كل منها بإجراء إحدى الفحوصات وتنفيذ بإنشاء مسار أو سلسلة تعليمات (thread) خاصة بها. الطريقة الثانية تعرف بـ Master/Slave Parallel Scheme حيث يتم إنشاء مسار خاص بفحص وفرز أصوات كل مركز اقتراع على حدا. الطريقة الأخيرة تعرف بـ Data Parallelism Scheme وتقوم بتوليد عدد من سلاسل التعليمات مساوٍ لعدد نوى وحدات المعالجة المركزية بحيث تكون كل منها مسؤولة عن فحص وفرز مجموعة من الأصوات التي تقاسمتها فيما بينها بشكل ديناميكي ومتوازن.

تم تمثيل عملية الاقتراع بإنشاء حزم من الأصوات وإخضاعها لعمليتي فحص وفرز الأصوات وفقاً لإجراءات ومحددات نظام S-Vote، من ثم تم تطبيق كل من طرق التنفيذ المتوازي وتقييمها مقارنة بالتنفيذ المتسلسل لعمليتي الفحص والفرز. أنتت النتائج بتفوق Data Parallelism Scheme عن نظيراتها بتحقيقها لنسب تحسين وكفاءة عالية وبأقل التكاليف. تمكنت هذه الطريقة من فحص وفرز 64,000 صوتاً يحوي كل منها على 16 مرشحاً خلال 44 دقيقة وبنسبة كفاءة بلغت 86% عند استخدامها لحاسوب يحتوي على 32 نواة.

تمكنت Data Parallelism Scheme من تقليل الوقت المستغرق بتنفيذ فحص ZK proof وكانت نسبة التسريع والتحسين بعلاقة طردية مع عدد نوايا جهاز الحاسوب الخاص بفحص الأصوات وفرزها. يمكن باستخدام هذه الطريقة توسيع نطاق استخدام نظام S-Vote واستعماله في مختلف أنواع الانتخابات إلى جانب تقليل كلفة الامتلاك والتشغيل. على سبيل المثال، يمكن باستخدام خادم يحوي 128 نواة تقليص وقت انجاز عمليتي الفحص والفرز لدولة بحجم الأردن من 25,4 يوماً إلى 5,7 ساعة.