ORHAN ÇETİNKAYA METU 2007

VERIFIABILITY AND RECEIPT-FREENESS IN CRYPTOGRAPHIC VOTING SYSTEMS

ORHAN ÇETİNKAYA

DECEMBER 2007

VERIFIABILITY AND RECEIPT-FREENESS IN CRYPTOGRAPHIC VOTING SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ORHAN ÇETİNKAYA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
CRYPTOGRAPHY

DECEMBER 2007

Approval of the Graduate School of Applied Mathematics.

_____
Prof. Dr. Ersan Akyıldız
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

_____
Prof. Dr. Ferruh Özbudak
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Doctor of Philosophy.

_____
Assoc. Prof. Dr. Ali Doğanaksoy
Supervisor

Examining Committee Members

Prof. Dr. Ersan Akyıldız (METU, MATH)_____

Assoc. Prof. Dr. Ali Doğanaksoy (METU, MATH)_____

Prof. Dr. Ferruh Özbudak (METU, MATH)_____

Assoc. Prof. Dr. Melek D. Yücel (METU, EEE) _____

Assist. Prof. Dr. Ali Aydın Selçuk (BİLKENT, CS) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Orhan Çetinkaya

Signature :

ABSTRACT

VERIFIABILITY AND RECEIPT-FREENESS IN CRYPTOGRAPHIC VOTING SYSTEMS

Çetinkaya, Orhan
Ph.D., Department of Cryptography
Supervisor: Assoc. Prof. Dr. Ali Doğanaksoy

December 2007, 141 pages

This thesis examines verifiability and receipt freeness in cryptographic voting protocols in detail and points out the contradiction between these requirements. Firstly, an extensive electronic voting requirement set is clearly defined, and then the voting dilemma is described. This is followed by a suggestion of an applicable solution to overcome the voting dilemma by introducing Predefined Fake Vote (PreFote) scheme.

Based on a comprehensive literature review, a classification of the existing privacy preserving approaches and a taxonomy of the existing cryptographic voting protocols extending the previous studies are provided. Thereby, a complete and secure cryptographic voting protocol satisfying all electronic voting security requirements at the same time seems non-existent. Hence, an alternative privacy preserving approach is highly needed. Pseudo-Voter Identity (PVID) scheme, proposed in the present study, is a practical and low cost one. The PVID scheme is based on RSA blind signature, and it allows recasting without sacrificing uniqueness. Furthermore, this study proposes a dynamic ballot mechanism including an extension with PreFotes.

This study, wherein the PVID scheme and extended dynamic ballots with PreFotes are employed, proposes a practical, complete and secure cryptographic voting protocol over a network for large scale elections, which fulfils all of the electronic voting security requirements: privacy, eligibility, uniqueness, fairness, uncoercibility, receipt-freeness,

individual verifiability and accuracy. Lastly, a method to analyse voting systems based on security requirements is suggested, and a detailed analysis of the proposed protocol, which uses this method, concludes this study.

Keywords: cryptographic voting, dynamic ballot, privacy, receipt-freeness, verifiability

ÖZ

KRİPTOGRAFİK SEÇİM SİSTEMLERİNDE DOĞRULANABİLİRLİK VE OYLARIN İSPATLANAMAMAZLIĞI

Çetinkaya, Orhan
Doktora, Kriptografi Bölümü
Tez Yöneticisi: Doç. Dr. Ali Doğanaksoy

Aralık 2007, 141 sayfa

Bu tezde, kriptografik seçim sistemlerinde doğrulanabilirlik ve oyların ispatlanamamazlığı gereksinimleri detaylı bir şekilde incelenmiş ve bu gereksinimler arasındaki çelişkiye dikkat çekilmiştir. Öncelikle gereksinimler konusunda kapsamlı bir çalışma yapılmış ve geniş bir gereksinim listesi hazırlanmıştır. Bu sırada oylama dilemması açıkça ortaya konmuş ve Önceden Tanımlanmış Sahte Oy (PreFote) yöntemi ile bir çözüm önerilmiştir.

Kapsamlı bir literatür taramasından sonra seçmen ve oyu arasındaki gizliliği korumaya yönelik yaklaşımlar sınıflandırılmış ve mevcut kriptografik oylama protokolleri gruplandırılmıştır. Literatürde bütün güvenlik gereksinimlerini aynı anda sağlayabilen uygulanabilir bir kriptografik oylama protokolü bulunmamaktadır. Bu nedenle, seçmen ve oyu arasındaki gizliliği korumaya yönelik alternatif yaklaşıma ihtiyaç vardır. Bu tezde pratik ve düşük maliyetli bir gizlilik koruma yaklaşımı olarak Sözde-Seçmen Kimliği (PVID) yöntemi önerilmektedir. PVID yöntemi, RSA kör imza kullanan bir gizlilik koruma yaklaşımıdır. Ayrıca bu tezde, geleneksel statik pusula yerine dinamik pusula önerilmiş ve dinamik pusulalar PreFote yöntemi ile geliştirilmiştir.

PVID ve PreFote yöntemleri ile geliştirilmiş dinamik pusulalar kullanılarak; geniş ölçekli seçimler için geniş alan ağlarında kullanılabilen ve bütün elektronik oylama gereksinimlerini sağlayabilen, pratik, güvenli ve uygulanabilir bir kriptografik oylama

protokolü önerilmiştir. Protokol, gizlilik, uygunluk, dürüstlük, tek oy kullanımı, zorlanamamazlık, ispat edilememezlik, bireysel doğrulanabilirlik ve doğruluk gereksinimlerinin hepsini karşılamaktadır. Son olarak oylama sistemlerinin analiz edilebilmesi için bir yöntem tanımlanmış ve önerilen protokol detaylı olarak bu yöntemle analiz edilmiştir.

Anahtar Kelimeler: kriptografik oylama, dinamik pusula, gizlilik, oyların ispatlanamamazlığı, doğrulanabilirlik

To My Wife and My Daughter

ACKNOWLEDGMENTS

It is a great pleasure to have the opportunity to express my thankfulness to all those who gave me the possibility to complete this thesis.

I would like to express my deepest gratitude to my supervisor Assoc. Prof. Dr. Ali Doğanaksoy for his support, guidance and insight he provided throughout this research. He has been very helpful and encouraging during my studies.

I also would like to thank to all of the examining committee members for assessing this thesis and for their advice. Their comments and suggestions have made a significant contribution to increasing the quality of this thesis. My special thanks are for Asst. Prof. Dr. Ali Aydın Selçuk who shared his invaluable comments, advice, and criticism. I would also like to extend my thanks to M. Levent Koç for his support and willingness he has shown to implement prototype of the protocol.

I would like to thank to all of the friends who was involved in the electronic voting study group in IAM and contributed in the early stage of this research. I also would like to thank to all of the members of the Caltech/MIT Voting Technology Project mail list for their valuable discussions.

I thank directors and project managers from my company for their support on my academic studies. Without the opportunities that they provided, I would not be able to come to this point.

My endless thanks are to my friends and my family, near and far, for all their encouragement over the last several years. And finally, I would like to thank my wife Deniz and my daughter Büşra Yıldız for their extraordinary patience and continuous support during the preparation of this thesis. In particular, I am grateful to Deniz for helpful discussions, giving me motivation for the next steps of this work and being with me all the way. Without her, neither this thesis nor my life would be complete.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
ANSI	American National Standards Institute
CCode	Check Code
DES	Data Encryption Standard
DRE	Direct Recording Electronic
FIPS	Federal Information Processing Standards Publications
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
HTML	Hypertext Markup Language
LFSR	Linear Feedback Shift Registers
MD5	Message Digest Algorithm 5
Mix-net	Mix Network
NIST	National Institute of Standards and Technology
RSA	Rivest Shamir Adleman
PKI	Public Key Infrastructure
PreFote	Predefined Fake Vote
RegID	Registration Identity
PRNG	Pseudo Random Number Generator
PVID	Pseudo-Voter Identity
SHA	Secure Hash Algorithm
TCP	Transmission Control Protocol
USB	Universal Serial Bus
XML	Extensible Markup Language
VVPAT	Voter-Verifiable Paper Audit Trails
V&V	Verification and Validation

CHAPTER 1

INTRODUCTION

This chapter contains the introduction and the motivation for the study, providing an overview of the thesis. Section 1.3 explains the main contributions. Section 1.4 lays the outline of the thesis, and this chapter ends with the list of publications which have been produced during the research process of the thesis.

1.1 Introduction

Voting is regarded as one of the most effective methods for individuals to express their opinions on a given topic. Electronic voting (e-voting) refers to the use of computers or computerised voting equipment to cast ballots in an election. Due to the rapid growth of computer technologies and advances in cryptographic techniques, electronic voting is now an applicable alternative for small scale non-critical elections. However; in many cases, voting needs to be performed in a large scale such as in governmental elections; thus, security requirements become even more critical. Electronic voting is a challenging topic in advanced cryptography. The challenge arises primarily from the need to achieve security and democracy requirements such as privacy, accuracy, receipt-freeness and verifiability. Therefore, electronic voting has been intensively studied in the last decades.

The majority of people may accept and use electronic voting, but many people have doubts about the privacy, security and accuracy of the election. They cannot easily trust the voting system unless the security of the system is greatly enhanced. Many controversies have been raised and many inconsistencies have been reported to be experienced with the real world electronic elections. The electronic voting experience in Ohio in 2004 is a well-known example; the incident caused much debate surrounding the evidence about vote miscount and modification [98].

The cryptographic voting protocols use advanced cryptography to make electronic voting secure and applicable. Voting protocols proposed so far could be classified into

three by their privacy preserving approaches: protocols using mix-nets, protocols using homomorphic encryption and protocols using blind signature. Blind signature based protocols are commonly implemented due to their practicality and applicability. However, there is no complete and secure cryptographic voting protocol which satisfies all electronic voting security requirements (especially receipt-freeness, uncoercibility and individual verifiability) at the same time. Although individual verifiability is not fully satisfied in paper based voting, it should be fulfilled in electronic voting protocols due to the nature of computer systems and electronic equipment. Individual verifiability is paramount to establish public trust in electronic voting.

There is a trade-off between receipt-freeness and individual verifiability. If a voting system provides any receipt, which enables the voter to verify his vote in the final tally, then that receipt can also be used for vote buying or selling. Individual verifiability also contradicts with privacy and uncoercibility because they are in close relation with receipt-freeness. In fact, checking a receipt is more convenient for a coercer than buying or stealing access keys and casting all votes himself. If receipt-freeness is not fulfilled, then uncoercibility and privacy cannot be assured.

Most of the research studies in the literature do not consider the voting dilemma; they generally focus on either individual verifiability or receipt-freeness. Most of them sacrifice receipt-freeness at the cost of accuracy and individual verifiability. A few protocols, which claim that they satisfy receipt-freeness, provide only universal verifiability or even worse no verifiability. In the literature, there is no protocol which satisfies receipt-freeness, uncoercibility and individual verifiability simultaneously, even under certain conditions or with assumptions.

In this thesis, we examine verifiability and receipt-freeness characteristics of cryptographic voting systems and propose applicable cryptographic building blocks and schemes in order to overcome the trade-off between these characteristics. The main contributions of this thesis are as follows:

- an extensive, clearly defined, electronic voting requirement set after a comprehensive review;

- a clear statement of the trade-off between individual verifiability and receipt-freeness and an applicable solution introduced by the Predefined Fake Vote (PreFote) scheme;

- a classification of the existing privacy preserving approaches and a taxonomy of the existing cryptographic voting protocols, drawing attention to a gap (i.e. there is no complete and secure cryptographic voting protocol which satisfies all of the electronic voting security requirements at the same time);

- proposal of Pseudo-Voter Identity (PVID) scheme, which is a practical and low cost privacy preserving approach;

- a dynamic ballot mechanism and presentation of how to extend it with PreFote scheme;

- a practical, complete and secure cryptographic voting protocol over a network for large scale elections, namely DynaVote protocol via the employment of PVID scheme and extended dynamic ballots with PreFote scheme. It fulfils all of the electronic voting security requirements: privacy, eligibility, uniqueness, fairness, uncoercibility, receipt-freeness, individual verifiability and accuracy;

- a prototype implementation of PVID scheme and DynaVote protocol;

- a method to analyse voting systems based on electronic voting security requirements and an analysis of DynaVote in detail with this method.

1.2 Motivation

Chaum introduced the first cryptographic voting protocol in 1981 [18] and many voting protocols have been proposed from both theoretical and practical perspectives since then. Many cryptographers, as well as Chaum [22], [21], Rivest [57], [89] and Benaloh [10] are still studying on cryptographic voting. However, no complete and practical solution for large scale elections over a network has been found. In all proposed voting protocols and implementations, different sets of requirements are defined and almost all academic studies focus on a subset of the requirements. However for an applicable electronic voting solution, all e-voting requirements should be met and a practical solution for large scale elections should be developed.

1.2.1 Voting Types

Based on the voting equipment and voting location, there are five types of electronic voting. Table 1.1 summarises all of these including paper based voting.

DRE voting: Direct Recording Electronic (DRE) machine is physically hardened electronic equipment with running special purpose voting software. It lacks a tamper-proof audit-trail. Satisfying accuracy and verifiability is almost impossible at DRE voting since any fraud during the voting process is unrecoverable and undetectable. This is similar to the current paper-based voting systems. The votes are cast inside a voting booth at a polling site; however, cast votes are recorded in electronic ballot boxes.

Poll-site voting: In poll-site voting, the votes are cast by using public computers at a polling site. Voting booths are not used, but a public polling-site is provided. The computers at the site are connected over a closed and controlled network. Cast votes are recorded by a counting authority server instead of electronic ballot boxes. Voters can be authenticated and authorised at the site before allowed to access to the voting machines, or they can have some voting credentials prior to the voting period.

Poll-site kiosk voting: In poll-site kiosk voting, the votes are cast inside a voting booth at a polling site as in DRE voting. Typically, voting booths at the site contain electronic voting terminals, and they are connected with a closed and controlled network. Cast votes are recorded by a counting authority server instead of electronic ballot boxes. Voters are authenticated and authorised at the site before allowed to access to the voting booths. Votes are cast using the terminal inside the voting booths.

Poll-site Internet voting: In this type, the votes are cast by using public computers at a polling site over Internet. Voting booths are not used, but a public polling-site is provided. The computers at the site are online over an uncontrolled network. Cast votes are recorded by a counting authority server instead of electronic ballot boxes. Voters can be authenticated and authorised at the site before allowed to access to the voting machines, or they can have some voting credentials prior to the voting period.

Remote Internet voting: Voters cast their votes over Internet. For authentication, the credentials of voters are verified prior to the voting period through the use of a password or some type of authentication token.

Table 1.1: Voting types.

	Stand-alone Voting	***Networked Voting***	
		Controlled Network	**Uncontrolled Network**
Paper Voting	Paper Based Voting	N/A	N/A
Electronic Voting	DRE Voting	Poll-site kiosk voting	Poll-site Internet voting
		Poll-site voting	Remote Internet voting

Most of the cryptographic voting protocols are proposed to be used with voting booths; and most of the academic studies focus on DRE voting. Besides, there are some voting protocols which can be employed over a network, especially an uncontrolled network such as the Internet [28], [2], [61], [75], [102], [85]. However, [2], [61] do not satisfy individual verifiability, nor do the others receipt-freeness and uncoercibility. Therefore, no complete cryptographic voting protocol has been proposed which is suitable for voting over a network.

1.2.2 Election Integrity

An election process wherein integrity is truly manifested would allow all voters to perform the voting and instil confidence that counting authorities will count all votes accurately. Election integrity would include creating a unified national voting system where all citizens have the same rights and equal opportunities to vote. It would fortify voting machine security, and require a verifiable record of every vote transaction.

A voting system that cannot assure that every vote is accurately counted is fundamentally flawed. Election integrity cannot be assured without openness and transparency. The voter cannot know that the vote eventually reported is the same as the vote cast, nor can candidates or others have confidence in the accuracy of the election without observing the voting and vote counting processes. Elections must be administered to minimise the possibility of error and fraud, and maximise the likelihood of detecting them if they occur.

1.2.3 Electronic Voting over a Network

Design of secure voting protocols over a network is not an easy task. It is extremely difficult to achieve the e-voting requirements while employing the protocol over a network. Especially in uncontrolled environments such as Internet, satisfying the criteria privacy, receipt-freeness, uncoercibility and verifiability are major problems.

Without a carefully designed system, the threats of coercion and vote buying are potentially far more problematic in Internet voting schemes than in ordinary, physical voting schemes. Vote buying or selling are not directly ascribed to Internet-based voting which appears to be the new trend voting in the near future. But Internet-based voting does have the potential to exacerbate them because it is prone to the reach and data collection abilities of an adversary. Allowing recasting prevents coercion, vote buying and selling of votes in uncontrolled environments.

1.2.4 Large Scale Elections

Privacy also known as voter anonymity is the main classification factor of the voting protocols. Up until now, cryptographic voting protocols have used either homomorphic encryption or anonymous communication channels mostly based on mix-nets to achieve voter privacy. Mix-net implementations need expensive operations and complex calculations. Moreover, mix-nets are not easy to set up and add substantial complexity to the protocol. Voting protocols based on homomorphic encryption, on the other hand, have communication complexity. Homomorphic voting protocols are inefficient if there are many candidates or choices. Thus, the existing voting protocols are not suitable for large scale elections since they require complex computational operations, specific hardware or some physical assumptions.

1.2.5 Voting Dilemma

In voting systems, there is a functional conflict between verifiability and secrecy. On the one hand, the voter wants to verify that the entire voting process has taken place appropriately. In particular, he wants to be assured that his individual vote was counted correctly. However, if the voter obtains adequate information from the voting process, then he can convince a vote buyer of how he voted, which in turn increases the likelihood that vote selling becomes a threat. Somehow, we want the voter to obtain enough

information to be personally convinced that his vote was indeed recorded as he cast, but not to unduly reveal information by which he could convince someone else.

1.2.6 Existing Voting Protocols

Voting technology is an active research area that has already produced several proposals that promise to be much better than any system currently in use. However, the literature lacks complete, practical and applicable voting protocols. Thus, commercial voting implementations are far from being cryptographically secure, and they are focused on DRE voting. Since the market employs DRE voting, most of the funded projects are limited to DRE-voting or poll-site kiosk voting. On top of its low level security, DRE voting is really expensive due to the cost of DRE machines, special hardware equipment, printers, system maintenance, and upgrade operations.

Another fallacy in this system is that the use of voter-verifiable paper audit trails (VVPAT) is thought to be enough for verifiability. However, a paper trail is pointless unless all printed records are counted, which will not make any sense. Unfortunately, if available funds are spent on fatally-flawed "high-tech" voting equipment, then it will be a long time before there is more funding to adopt a truly superior voting technology.

1.3 Main Contributions

This thesis contributes to the theory and practice of cryptographic voting. Each contribution attempts to make electronic voting more effective, practical and applicable in real life.

1.3.1 Classification of Privacy Preserving Approaches

A comprehensive literature review has yielded a classification of the existing privacy preserving approaches and a taxonomy of the existing cryptographic voting protocols extending the previous studies. In the literature, there is no complete and secure cryptographic voting protocol which satisfies all electronic voting security requirements (especially receipt-freeness, uncoercibility and individual verifiability) at the same time. The classification is given in Chapter 4.

1.3.2 PVID Scheme for Voter Secrecy

Our literature review made it clear that, in order to carry out practical secure elections, an alternative privacy preserving approach instead of mix-nets or homomorphic encryption is highly needed. An efficient and practical solution enhancing privacy will make secure electronic voting applicable in real-life.

Thus, we propose a practical and low cost privacy preserving approach, Pseudo Voter Identity (PVID) scheme. PVID scheme is based on blind signature and RSA. It provides anonymous pseudo identities which are unlinkable to the voter's real identity. PVID scheme affords privacy without requiring any complex operations, which are the drawbacks of the mix-nets and homomorphic encryption. PVID scheme only employs blind signature and the cost of blind signature is reasonable. Due to the fact that PVID is not the voter's real identity and counting authorities can keep PVIDs, PVID scheme allows vote recasting without sacrificing uniqueness. PVID Scheme is presented in Chapter 5.

1.3.3 DynaVote Electronic Voting Protocol

This thesis proposes a complete and secure cryptographic voting protocol over a network for large scale elections, which is voter-verifiable, receipt-free and coercion-resistant. The proposed protocol, namely DynaVote, contributes to the literature mainly by presenting a practical and secure cryptographic voting protocol which fulfils all of the electronic voting security requirements: privacy, eligibility, uniqueness, fairness, uncoercibility, receipt-freeness, individual verifiability and accuracy. DynaVote has no physical assumption such as untappable channels, voting booths, mix-nets, special hardware. What is more, it is a complete protocol for large scale elections which can be performed over an existing network such as the Internet. DynaVote is verifiable in each stage, and voters can object to the voting system on the grounds that it is corrupt without revealing his real identity. The DynaVote protocol uses dynamic ballots extended with predefined fake votes and employs an unlinkable pseudo identity mechanism. It utilises bulletin boards and cryptographic hash functions. DynaVote protocol is presented in Chapter 7.

1.3.4 Predefined Fake Votes (PreFotes)

In this thesis we examine verifiability and receipt freeness in cryptographic voting protocols and identify the contradiction between these requirements. Receipt-freeness and individual verifiability conflict with each other, causing the voting dilemma. It is due to the fact that if a voting system provides any receipt which enables voter to verify his vote in the final tally, then that receipt can also be used for vote buying or selling. Hence, we introduce Predefined Fake Vote (PreFote) scheme as an applicable solution in order to overcome the voting dilemma. The PreFote scheme can be directly employed to the protocols that perform poll-site voting or kiosk voting.

A PreFote list includes intentionally prepared fake votes which consist of a unique code and a candidate associated from the candidates list. For each candidate, a constant threshold number of PreFotes is generated and listed in the PreFote list. This feature provides direct individual verifiability without sacrificing receipt-freeness and accuracy. The PreFote scheme is elaborated in Section 3.8.

1.3.5 Dynamic Ballots

In this thesis, we propose a dynamic ballot mechanism instead of the predefined usual ballot in order to increase accuracy, verifiability and fairness of voting protocols. Dynamic ballot mechanism is not a user interface implementation; it is a part of the protocol itself and employed not only in the user interface layer but also in the protocol layer. DynaVote protocol employs dynamic ballots that are extended with PreFotes; therefore the proposed protocol is called as “DynaVote”. Dynamic ballots are presented in Chapter 6.

1.3.6 Comprehensive Definitions of E-voting Requirements

The security requirements make electronic voting a challenging issue in advanced cryptography. In this thesis an extensive electronic voting requirement proposed is set with clear definitions after a widespread review of characteristics associated with secure election systems in the literature. This review study shows that in all proposed voting protocols and implementations, different sets of requirements are laid and almost all academic studies focus on a subset of the requirements. As well as this, the definitions of some requirements are inadequate and unclear. Voting requirements are explained in

Chapter 3. As well as suggesting a method to analyse the voting systems based on electronic voting security requirements, we analyse DynaVote in detail with this method.

1.3.7 Verification and Validation in Electronic Voting

Verifiability in electronic voting protocols has been discussed recently. Unfortunately the definitions of verifiability are inadequate and unclear. An innovative study which states the insufficiency of the way verifiability is explained in the literature is presented and clear definitions are provided in Section 3.5.

1.4 Thesis Outline

After this introduction chapter, the thesis continues with Chapter 2 which introduces cryptographic primitives. In Chapter 3, an overview of electronic voting is given and the voting dilemma is described. Then, approaches to preserve privacy in electronic voting protocols are explained in Chapter 4. In Chapter 5, "PVID Scheme" is introduced and discussed. Dynamic ballots are presented in Chapter 6. A new cryptographic voting protocol "DynaVote" is proposed and explained in Chapter 7, and it is analysed in Chapter 8 based on the fulfilment of the e-voting security requirements. Finally, conclusions are drawn and future work is suggested in Chapter 9. Appendix A contains some supplementary cryptographic primitives and Appendix B gives Prototype implementation details.

1.5 Published Research

The following papers have been published and presented throughout the research process of this thesis.

Refereed Journal/Conference Papers:

- O. Cetinkaya and D. Cetinkaya, "*Verification and Validation Issues in Electronic Voting", The Electronic Journal of e-Government (EJEG), Volume 5 Issue 2*, pp 117-126, Academic Conferences Limited, United Kingdom, 2007.

- O. Cetinkaya, *"Analysis of Security Requirements for Cryptographic Voting Protocols (Extended Abstract)", 4th Symposium on Requirements Engineering for Information Security (SREIS'08),* Barcelona, Spain, 4-7 March 2008. ©IEEE

- O. Cetinkaya and A. Doganaksoy, "*A Practical Verifiable E-Voting Protocol for Large Scale Elections over a Network", In Proceedings of the 2nd International Conference on Availability, Reliability and Security (ARES'07)*, Vienna, Austria, pp. 432-442, 10-13 April 2007. ©IEEE

- O. Cetinkaya and A. Doganaksoy, "*Pseudo-Voter Identity (PVID) Scheme for E-Voting Protocols*", *In Proceedings of the International Workshop on Advances in Information Security (WAIS'07) in conjunction with ARES'07*, Vienna, Austria, pp. 1190-1196, 10-13 April 2007. ©IEEE

- O. Cetinkaya and D. Cetinkaya, "*Validation and Verification Issues in E-Voting*", *In Proceedings of the 7th European Conference on E-Government (ECEG'07)*, The Hague, Netherlands, pp. 63-70, 21-22 June 2007.

- O. Cetinkaya and D. Cetinkaya, "*Anonymity in E-Voting Protocols", In Proceedings of the 3rd International Conference on Global E-Security (ICGeS'07),* London, United Kingdom, pp. 137-143, 18-20 April 2007.

- O. Cetinkaya and D. Cetinkaya, "*Towards Secure E-Elections in Turkey: Requirements and Principles*", *In Proceedings of the 2nd International Workshop on Dependability and Security in e-Government (DeSeGov'07) in conjunction with ARES'07,* Vienna, Austria, pp. 903-907, 10-13 April 2007. ©IEEE

National Conference Papers:

- O. Cetinkaya and A. Doganaksoy, "*A Practical Privacy Preserving E-Voting Protocol Using Dynamic Ballots", National Cryptology Symposium II*, Ankara, Turkey, 15-17 December 2006.

- O. Cetinkaya and A. Doganaksoy, "*Electronic Voting Protocols Based on Blind Signatures", National Cryptology Symposium I*, Ankara, Turkey, 18-20 November 2005.

- D. Cetinkaya and O. Cetinkaya, "*E-Seçim Uygulamaları için Gereksinimler ve Tasarım İlkeleri", XI. 'Türkiye'de Internet' Konferansı (inet-tr'06),* Ankara, Turkey, 21-23 December 2006.

CHAPTER 2

CRYPTOGRAPHIC PRIMITIVES

Some cryptographic primitives mentioned throughout the thesis and used in the proposed work are briefly explained in this chapter.

2.1 Blind RSA Signature

A blind signature is a form of digital signature in which the content of a message is concealed (i.e. *blinded*) before it is signed. The concept of blind signature was introduced by Chaum [19] as a method to digitally authenticate a message without knowing the content of the message.

A distinguishing feature of blind signatures is their unlinkability: The signer cannot drive any association between the signing process and the signature, which is later made public. The resulting blind signature can be publicly verified against the original, unblinded message by means of a regular digital signature. In other words, blind signatures are the equivalent of signing carbon paper lined envelopes. Putting a signature on the outside of such envelopes leaves a carbon copy of the signature on a slip of paper within the envelope. When the envelope is opened, the slip will show the carbon image of the signature.

The blind RSA signature scheme is briefly as follows. Suppose Alice has a message *m* that she wishes to have signed by Bob. Alice does not want Bob to learn anything about *m*. Let *(e, n)* and *(d, n)* be Bob's public and private keys, respectively.

- Alice generates a random number r such that *gcd(r,n)=1*, and calculates $x = (r^e m) \bmod n$ and then sends *x* to Bob. The value *x* is *blinded* by the random value *r*; hence, Bob cannot derive any useful information from it.

- Bob signs the blinded message *x* and obtains a blinded signature $t = x^d \bmod n$. Then he sends the blinded signature *t* to Alice.

- Alice reads the blinded signature *t* and obtains the true signature *s* of *m*. The blinded signature *t* can be calculated as:

  $t = x^d \bmod n = (r^e m)^d \bmod n = r^{ed} m^d \bmod n = r m^d \bmod n$

  The true signature *s* of *m* can be computed as:

  $s = r^{-1} t \bmod n = m^d \bmod n.$

  This is indeed the signature of *m*.

2.2 Pseudo Random Number Generator

Pseudo random number generator (PRNG) is a deterministic algorithm to generate a sequence of numbers with little or no discernible pattern in the numbers. The sequence is not truly random since it is determined solely by a relatively small set of initial values. Although sequences that are closer to truly random ones can be generated using hardware random number generators, most pseudo random generator algorithms produce sequences which are uniformly distributed.

Getting truly random data is typically based on nondeterministic physical phenomena. In the deterministic environment of computer systems, people often use deterministically generated pseudorandom data. The truly random data are used only as a seed for deterministic pseudorandom number generators and after seeding, an arbitrary amount of pseudorandom data is always available. The PRNG is in fact a deterministic finite state machine, which implies that it is at any point of time in a certain internal state. This PRNG state is kept confidential since the PRNG output must be unpredictable.

Many classes of PRNGs exist, but the goal of a PRNG in cryptography is the production of pseudo random data that are computationally indistinguishable from statistically ideal random data. A PRNG is cryptographically secure, on condition that it is computationally infeasible to predict the next output even if all the previous outputs and the complete algorithm are given.

Basic types of PRNGs utilise linear feedback shift registers (LFSR) [71], NP hard problems of number and complexity theory and typical cryptographic functions/primitives. Mechanisms necessary for recovering from the state compromise are used only in the last category.

Well known PRNGs based on NP hard problems of number and complexity theory are RSA PRNG, Micali-Schnorr PRNG and Blum Blum Shub (BBS) PRNG. The first two PRNGs are based on the well-known factorisation problem (as cryptosystem RSA). The last one is based on the quadratic residuosity problem. The generators based on the discrete logarithm problem with small exponents are much faster than the others [45], [32]. Some standardised generators are ANSI X9.17/X9.31 PRNG [5], [63] and FIPS 186-1/2 PRNG [37]. ANSI X9.17 PRNG is based on using 3DES (ANSI X9.31 allows also AES), and the generators described in FIPS 186-1/2 are based on single DES or SHA-1.

2.3 Cryptographic Hash Functions

A cryptographic hash function is a hash function *h* with certain additional security properties, which takes an arbitrary size input *x* and outputs a fixed length output *h(x)*. Although a cryptographic hash function is deterministic and efficiently computable, it should behave as much as possible like a random function. Hash functions are assumed to be public; therefore if *x* is given, anyone can compute *h(x)*.

Digital signatures and data integrity are the most common cryptographic uses of hash functions. With digital signatures, a long message is usually hashed (using a publicly available hash function), and only the hash-value is signed. The party receiving the message then hashes the received message and verifies that received signature is correct for this hash-value. This saves both time and space compared to signing the message directly.

In order to meet the requirements of a signature scheme the following three properties are required of a cryptographic hash function *h*:

- *Pre-image resistance* means that given *h(x)*, it is computationally infeasible to extract any bits of *x*.

- *Second pre-image resistance* means that given *x*, it is computationally infeasible to find *y* such that *h(x) = h(y)*.

- *Collision resistance* means that it is computationally infeasible to find any *x* and *y* such that *h(x) = h(y)*.

MD5, SHA-1, SHA-256 are well known hash algorithms. The MD5 algorithm produces a 128-bit message digest used to validate data integrity. The SHA-1 algorithm produces a 160-bit message digest and is therefore considered a stronger algorithm than MD5. SHA-1 is utilised in a broad range of popular security applications and protocols. The SHA-256 hashing algorithm extends the size of the digest to 256 bits for heightened security. Wang *et al.* showed the collisions for MD5 [101]. Therefore, SHA-256 is the preferred cryptographic hash function in practice. All SHA hash functions (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512) are approved by NIST [79].

2.4 Bulletin Boards

A bulletin board is a public broadcast channel with universally accessible memory where a party may write information via secure communication in the designated areas. The information can be read by any party. Bulletin boards are commonly used in electronic voting protocols. All communications with the bulletin boards are public and therefore can be monitored. Generally, data already written into a bulletin board cannot be altered or deleted in any way, but it can be read or appended.

CHAPTER 3

ELECTRONIC VOTING AND SECURITY

Based on a comprehensive literature review, this section proposes an extensive set of electronic voting requirements. These requirements are categorised as e-voting security requirements, e-voting system requirements and e-voting properties. E-voting security requirements are mandatory for any cryptographic voting protocol. A secure and complete protocol should meet these requirements. Otherwise it will not be an adequate solution to the electoral needs. E-voting system requirements are needed for any electronic voting system which has a software implementation of any voting protocol. E-voting properties are additional requirements that any voting protocol or system may have.

The verifiability requirements are discussed in detail as there is little discussion of verification and validation in e-voting. The discussion points out the inadequate and unclear definitions. Proper definitions for verifiability and validity are suggested. At the end of this chapter, the voting dilemma is explicitly verbalised and an applicable solution is suggested.

3.1 A Typical Voting Process

The basic process of any electronic election is almost standard although a wide variety of electronic voting systems and protocols exist. A general electronic voting process and the actors involved can be summarised as in Figure 3.1. Any voting system should include these actors:

- *Voter:* Voter has the right for voting, and he votes in the election.
- *Registration Authority(ies):* Registration authority or authorities register eligible voters before the election day. These authorities ensure that only the registered voters can vote, and they vote only once on the election day. Registration

authorities may be the registrar, authenticator, authoriser, ballot distributor and/or key generator.

- *Tallying Authority(ies):* The tallying authorities collect the cast votes and tally the results of the election. Tallying authorities may be counter, collector and/or tallier.

Any voting system should also involve these four stages:

- *Registration:* Voters register to vote, and the registration authorities compile the list of eligible voters before the election day.

- *Authentication and Authorisation:* On the election day registered voters request ballots or voting privilege from the registration authorities. Registration authorities check the credentials of the voters attempting to vote and only allow those who are eligible and have registered before.

- *Voting:* Voter casts his vote.

- *Tallying:* The tallying authorities count the votes and announce the election results.

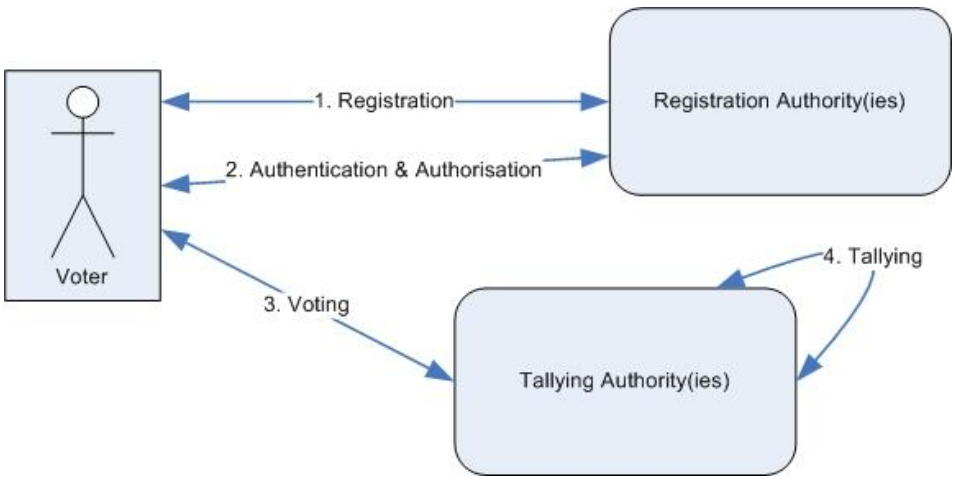

Figure 3.1: A typical voting process.

3.2 E-voting Security Requirements

A secure and complete cryptographic voting protocol should satisfy these requirements.

- *Voter Privacy*: It is the prevention of associating a voter with a vote [88], [28]. Voter privacy must be preserved during the election as well as after the election. In order to assure privacy, both unlinkability and untraceability should be fulfilled.

  - There are two identities which directly identify voter and are probably known publicly. They are voter's registration identity (RegID) and voter's public key. No one should be able to relate these two identities to voter's cast vote. This is called as unlinkability.

  - A voter may have one more indirect identity, which is the IP address of the computer via which the voter casts his vote. The IP address should be perfectly untraceable so that no one can draw a relation between a voter and his vote. This is called as untraceability.

- *Eligibility*: Only eligible voters participate in the election [13], [42]. They should register before the election day, and only the eligible voters who have registered to vote can cast votes.

- *Uniqueness*: Only one vote for a voter should be counted [40]. It is important to notice that uniqueness does not mean unreusability (i.e. voters should not vote more than once).

- *Fairness*: No partial tally of results is revealed before the end of the voting period to ensure that all candidates are given a fair decision [3]. Even the counter authority should not be able to have any idea about the results.

- *Uncoercibility*: Any coercer, including the authorities, should not be able to extract the value of the vote [13] and should not be able to coerce a voter to cast his vote in a particular way. Any voter must be able to vote freely.

- *Receipt-freeness*: It indicates that the system does not provide a confirmation of the receipt of the vote which may yield its content. In other words, voters should not obtain a receipt, nor can they construct one, which can be used to prove the

content of their votes a third party [9] both during the election and after the election ends. This is to prevent vote buying or selling.

- *Accuracy*: The published tally should be correctly computed from correctly cast votes [13]. Accuracy can be analysed in two ways:

  - All valid votes should be counted correctly. Any vote cast cannot be altered, deleted, invalidated or copied [9]. Any falsification on the votes should be detected.

  - All counted votes should be valid and correct, i.e. eligibility and uniqueness should be satisfied. No participant, voter or authority can disrupt or influence the election and final tally by adding false votes (a.k.a. Soundness and Completeness). Nobody should be able to vote in the place of others, even if they are eligible voters but they do not vote for some reasons (a.k.a. Abstaining Voter problem) or they abandon the voting process in any stage.

  Remark about universal verifiability: The literature highlights universal verifiability as another common requirement. The definition of universal verifiability is very similar to the definition of accuracy. It can be stated that universal verifiability as the provability that the election is accurate. If a protocol claims that it satisfies accuracy, it should be able to prove its claim. In this perspective, any protocol claiming to satisfy accuracy should also satisfy universal verifiability. Evidently, universal verifiability is not an e-voting requirement, whereas accuracy is. Thus, in this thesis only accuracy is listed as a requirement. Further discussion can be found in Section 3.5.

- *Individual Vote Check (a.k.a. Individual Verifiability)*: The voter should be able to check that his encrypted vote was counted and tabulated correctly in the final tally [40]. In traditional paper-based voting systems, people cannot make individual vote check directly. However, the voter casts his vote into the ballot box by himself. Since the security of the ballot box is guaranteed, individual vote check is, in a way, assured. Although this requirement is not directly satisfied in paper based voting, it should explicitly be fulfilled in electronic voting protocols due to the nature of computer systems and electronic equipment.

3.3 E-voting System Requirements

Any voting system founded on the software implementation of a cryptographic voting protocol should satisfy these requirements. It is worth noting, however, that this list can be extended.

- *Robustness*: The election process and final tally cannot be disrupted or influenced by any party or participant including authorities [9]. This is the system level requirement for accuracy. To have confidence in the election results, robustness should be assured. However, they are prone to corruption in numerous ways. For example, registration authorities may cheat by allowing ineligible voters to register; ineligible voters may register under the name of someone else; ballot boxes, ballots and vote counting machines may be compromised [28]. In order to meet the robustness requirement, the system should be protected against any kind of active and passive attacks [13], [61]. The voting system should be backed up and use a manifold [61] of important components against the failures and attacks. Any voter should be helped to recover from an interruption in the voting process. If voting is performed over a network, then all of the security requirements should be guaranteed even if the network is monitored, timing attacks or DoS attacks are anticipated.

- *Efficiency*: In all phases of authentication & authorisation, voting and tallying, the processes should be done efficiently (in a very short time). It is desirable to get the results as soon as possible after the voting phase ends.

- *Convenience*: A convenient system allows voters to cast their votes quickly and in one session, without any dependence on any extra equipment or special skills. No particular computer knowledge, for example, should be necessary to cast a vote [28], [40]. User interfaces should be clear and easy to use. The system should not be conducive to any misunderstanding or contain ambiguous information.

- *Equality of candidates*: The voting system should give equal opportunity to the candidates [74].

- *Open Source*: All source codes should be allowed to be publicly known and verified [88]. The security and reliability of the system must not rely on secrecy

of its source codes, which cannot be guaranteed. Only keys must be considered secret.

- *Transparency*: The whole voting process must be transparent. Bulletin boards may be used to publicise the election process. The security and reliability of the system must not rely on the secrecy of the network, which cannot be guaranteed.

- *Recounting and Auditing*: The election data and results should be saved. The system should allow off-line recounting and auditing after the election ends without compromising the election integrity or voter privacy [88].

- *Technical Adequacy*: Technical infrastructure and hardware should be adequate. Well established cryptographic techniques which are effective both in the short and long term should be used.

- *Announcement of Results*: The tally and election results and other information which is eligible to be known publicly should be announced after the election.

- *Design Independence*: The electronic voting system design should not depend on the programming language, operating system, development environment and technology.

- *Empty Ballot*: The system should represent blank votes, which means none of the candidates is selected. Voters may change choices from 'vote' to 'blank vote' and vice-versa before casting the ballot [88]. Blank votes should also be counted as empty ballots, and they cannot be filled, altered, deleted, invalidated or copied.

3.4 E-voting Properties

This section enumerates additional requirements that any voting protocol or system may have. They are desirable, if not mandatory. This list can be extended.

- *Scalability*: A voting system is scalable if it supports small, mid and large scale elections without any extra effort and is scalable with respect to storage, computation, and communication needs as a fraction of the number of voters.

- *Practicality*: A voting scheme is practical if it does not have assumptions and requirements difficult to implement on a large scale.

- *Mobility*: A voting system is mobile if the voter is not restricted to a particular location from where he can cast a vote [28].

- *Cheap Elections*: The cost of the electronic voting should be less than that of the paper-based voting.

- *Flexibility*: A system is flexible if it allows a variety of ballot formats such as write-in ballots and some survey questions [28].

3.5 Verifiability

The very nature of electronic voting necessitates the researchers to somehow persuade the voter that his vote has been really counted and the voting has been carried out properly. This requirement is named as verifiability and has been reported in the literature for many years. Unfortunately the definitions for verifiability are inadequate and unclear. Moreover, verifiability is categorised as individual verifiability and universal verifiability, and these are generally misused. Besides, validation has not been discussed properly yet, and there is no obvious consensus about the definitions.

Fujioka *et al.* [42] pioneered the verifiability in voting protocols by forcing voters to be involved in more than one round. Each voter has to participate in the counting stage by checking that his vote is listed correctly in the tallying list, and then sending a part of the vote in order to complete voting. In this protocol, verifiability is well defined in these words: “No one can falsify the result of the voting”.

Later, Sako and Kilian [92] introduced the concept of universal verifiability to emphasise the importance of auditing of the overall election, and they categorized verifiability as individual variability and universal verifiability. Thereupon, electronic voting studies applied this categorisation. Sako and Kilian define individual and universal verifiability respectively as “A sender can verify whether or not his message has reached its destination, but cannot determine if this is true for the other voters”, and “In the course of the protocol the participants broadcast information that allows any voter or interested third party to at a later time verify that the election was performed properly”.

Cranor and Cytron [28] define universal verifiability even more narrowly as simply as just counting the votes. They also identify verifiability as follows: “Anyone can independently verify that all votes have been counted correctly”. Most of the later studies used this definition since it is much more specific and measurable. [50] and [87] give a variant of the aforementioned definitions for verifiability. [50] takes verifiability as follows: “Every voter can make sure that his vote has been taken into account in the final tabulation”; and [87] characterises verifiability as “A system is verifiable if voters can independently verify that their votes have been counted correctly”.

Karlof *et al.* [62] combines the verifiability definition without distinguishing between universal or individual verifiability: “Verifiably cast-as-intended means each voter should be able to verify his ballot accurately represents the vote he cast. Verifiably counted-as-cast means everyone should be able to verify that the final tally is an accurate count of the ballots.”

Obviously, the definitions are not unique and comprehensive. However, when they are examined in detail, it becomes evident that they all imply the same meaning. They use verifiability in the sense of the validation of the final tally by the actors of the voting system (e.g. the voters, authorities, passive observers or trusted third parties). Unfortunately, this explanation is not adequate. “Validating the final tally”, “verifying that all votes have been counted correctly”, and “assuring the result of the voting” …etc can be regarded as some activities of the verification and validation (V&V) processes. As a result, comprehensive definitions should be stated for the verifiability requirement. Moreover, validation should be taken into consideration; the difference between verification and validation should be pointed out; and validity requirement should be introduced in electronic voting.

The individual verifiability and universal verifiability definitions used in the literature can be summarised respectively as in the following quotations “every voter can check if his vote has been properly counted” and “anyone can check that the calculated result is correct and election is performed correctly” [42], [28], [92], [50], [87], [62]. That is, clear and formal definitions are needed.

Delaune *et al.* [33] formalises some of the e-voting requirements and then verifies whether the requirements hold on particular voting protocols. Specifically they use the formalism of the applied pi calculus, which is a formal language similar to the pi calculus

but with useful extensions for modelling cryptographic protocols and which has been used to analyse a variety of security protocols in other domains. Verification of the requirements is illustrated in two case studies and has been partially automated using the Blanchet's ProVerif tool [11]. Delaune *et al.* lays out the formal verification methods for some of the e-voting requirements; however, they do not mention anything about the validation issues. Formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics. This research seems to have important implications for future studies since it meets formal verification with electronic voting. As well as this, a recent study of Cansell *et al.* [15] recommends application of formal methods to guarantee tamper evident storage of votes.

Another well known study is the concept of Voter Verified Paper Audit Trail (VVPAT), introduced by Mercuri [72]. VVPAT refers to a kind of "vote receipt" printed by an electronic voting machine. For audit and recount purposes, the VVPAT is kept by the election official as the record of votes cast. Although VVPAT is commonly accepted in U.S., it can be easily seen that VVPAT does not guarantee the accuracy of the system. A voter, actually, does not verify his vote with VVPAT for looking at a piece of paper does not mean verification.

In addition to these theoretical studies, there are also a few implementations which focus on verifiability, within the context of above definitions. VoteHere VHTi [100] is a commercial software which is an independent verification and validation technology. It works with DRE machines and based on Neff's cryptographic algorithm [77]. However, it has some drawbacks as to integration and usage, rendering it unpractical [96].

The importance of the verification and validation in electronic voting is discussed in [16], and proper definitions for verification and validation for electronic voting are stated. This not being within the scope of this thesis, just short definitions are given in this section.

In electronic voting, verification is the process of verifying that the voting system complies with design specifications and with the formally specified system requirements, such as accuracy, robustness and fairness; validation is the process of validating that the voting system satisfies its intended use and fulfils the user requirements, such as privacy and eligibility. Verification also includes the review of interim work steps and interim

outputs during the voting process to ensure they are acceptable. Therefore, verification tries to answer the question “Do we apply the protocol and build the system right?”, and validation tries to answer the question “Do we apply the right protocol and build the right system?”. Verification and validation in electronic voting is illustrated in Figure 3.2.

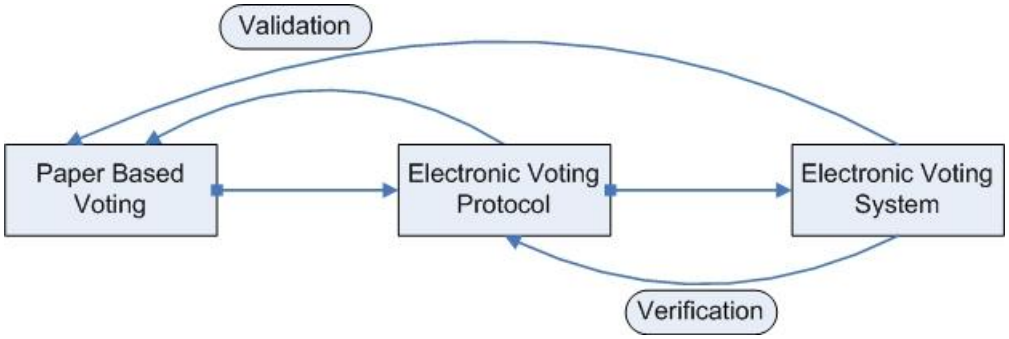

Figure 3.2: Verification and validation in electronic voting.

Based on these definitions, it can be stated that individual verifiability used in the literature can be treated as a part of the validation process since the voter checks whether his vote has been really counted in the final tally. Besides, universal verifiability can be a part of the verification process as it is employed to check dishonest authorities and some internal processes.

While V&V are parts of the overall system development process, they are extremely important because they are the only way to produce the right system in the right manner. The V&V of electronic voting protocol or system are parts of the overall design and development processes. So, in an ideal case, V&V should not be handled as voting requirements such as verifiability or validity, since it is expected that V&V should be performed by default. However, this is not currently achievable in practice and therefore there are many studies which define verifiability as a requirement. Thus, within the mentioned e-voting context, the definitions of verifiability and validity are given. Verifiability is the ability to perform the verification process of the electronic voting protocol or system; and validity is the ability to perform the validation process of the electronic voting protocol or system.

In order to fully perform validation in voting protocols and systems, a voter should be an active participant. The reason is that nobody can know the voter's cast vote except the voter himself. Thus, to validate the voting system completely, voters should be involved in V&V processes during or at the end of the election. Allowing passive observers to monitor the election can be a reasonable approach to achieve some V&V activities.

Apart from this, in order to cover individual verifiability as an e-voting requirement we offer an alternative naming for that requirement to prevent any misunderstanding: individual vote check. It means that the voter should be able to check that his vote is counted correctly in the final tally. However, in order to be comparable with existing protocols and to be backward compatible, *individual verifiability is used throughout the thesis instead of individual vote check*.

3.6 Receipt-freeness

In traditional elections, a voting booth provides voters with vote secrecy, as well as preventing vote-selling and coercion. Preventing such misuses in electronic voting schemes has been the subject of recent research studies. Receipt-freeness is a special security requirement of electronic voting protocols, and it makes e-voting different than other cryptographic protocols. It is thought that in order to perform real political elections, receipt-freeness should be provided because vote buying and coercion are common experiences in real world election scenarios.

The concept of receipt-freeness was first introduced by Benaloh and Tuinstra [9]; it means that the voter cannot prove the content of his vote to any third party. A voting system should ensure that neither a voter could sell his vote nor someone else could coerce him; that is, a voter must neither obtain nor be able to construct a receipt which can prove the content of his vote. The same idea was also introduced independently in [78]. Okamoto [80] proposed a voting scheme which he himself later proved to lack receipt-freeness; an improved version by the same author, making use of blind signatures, appears in [81]. Sako and Kilian [92] proposed a multi-authority scheme employing a mix network to conceal candidate choice, and a homomorphic encryption scheme to produce the final tally. The modelling of their scheme was clarified and refined by Michels and Horster [73]. The study in [92] served as a conceptual basis for the later work of Hirt and Sako [53], followed by the more efficient approach of [7];

these two are the most efficient (and correct) receipt-free voting schemes to date. A recently proposed scheme by Magkos *et al.* [70] distinguishes itself by an approach relying on tamper-resistant hardware. However, they assume untappable channels.

The existing receipt-free protocols in the literature make some basic assumptions about the communication channel between the voter and the voting authorities depending on the design of the protocol, and about the voting process. These assumptions can be modelled by the following primitives:

- An untappable channel: This channel models a physical apparatus by which the voter and voting authorities can exchange message. This message will be perfectly secret to all other parties. Untappable channels are used in three ways:

  - One-way untappable channel from the voter to the authority [81], [80].

  - One-way untappable channel from the authority to the voter [53], [92].

  - Two-way untappable channel (voting booth) between the voter and the authority [9], [67].

- A voting booth, in which the voter casts the vote. This models a physical booth and guarantees the secrecy of the communication between the voting authority and the voter.

Several authors in the literature have pointed out the difficulty of implementing untappable channels [53]. Such channels can be quite cumbersome, particularly for large-scale voting with geographically distributed voters. Note that untappable channels will also force the voter to use specified voting locations. In short, we can say that these assumptions are not applicable for electronic voting over a network.

The property of receipt-freeness also ensures that an attacker cannot determine the exact voter behaviour and therefore cannot coerce a voter. This being so, receipt-freeness has a strong relationship with uncoercibility. Uncoercibility avoids even the scenarios where the voter cooperates with the coercer, and they both try to find a strategy where the voter can prove that he followed the coercer's instructions (e.g., they can choose specific private keys and a strategy through which the voter can prove that he voted a specific value or a random value).

Moreover, all of the receipt-free schemes (except [81]) lose the property of coercion-resistance if one of the tallying authorities corrupts. The scheme in [81] makes an even stronger assumption of an anonymous untappable channel. The scheme of Hirt and Sako [53] still retains coercion-resistance when such corruption takes place, but only under the strong assumption that the voter knows which tallying authorities have been corrupted; the proposal of Baudron *et al.* [7] has a similar property.

3.7 Voting Dilemma

Voting requirements are explained in the previous sections. Designing secure voting systems is tough since the requirements are apparently contradictory. In this section, the voting dilemma is explicitly stated.

According to the definitions of receipt-freeness and uncoercibility, we can conclude that a voter, even because of coercion or by his own will, could neither obtain nor be able to construct a receipt that proves the content of his vote. This is to allow voting freely and to prevent vote buying or selling.

According to the definitions of individual verifiability and accuracy, we can conclude that the published tally should be correctly computed from correctly cast votes in a verifiable manner and a voter himself should be able to check that his vote has been counted correctly in the final tally.

The voting dilemma arises from the combination of these requirements. Specifically, the dilemma between receipt-freeness and individual verifiability will be described since they conflict with each other obviously. There is a noticeable contradiction between receipt-freeness and individual verifiability. If a voting system provides any receipt which enables the voter to verify his vote in the final tally, then that receipt can also be used for vote buying or selling. Checking a receipt is more convenient for a coercer than buying or stealing access keys and casting all votes himself. Thus there is a trade-off between receipt-freeness and individual verifiability.

Individual verifiability also contradicts with privacy and uncoercibility because they have close relation with receipt-freeness. If receipt-freeness is not fulfilled, then uncoercibility and privacy cannot be assured. Delaune *et al.* [33] show the strong relationship between privacy, receipt-freeness and uncoercibility in applied pi calculus.

Chevallier-Mames *et al.* [23] show that it is not possible to achieve universal verifiability of the tally and unconditional privacy of the votes simultaneously, unless all the registered voters actually vote and that it is not possible to achieve universal verifiability of the tally and receipt- freeness, unless the voting process involves interactions between several voters and possibly the voting authority.

Although individual verifiability is not directly satisfied in paper based voting, it should be fulfilled in electronic voting protocols due to the nature of computer systems and electronic equipments. When paper based voting is applied, voters can be easily persuaded that their votes are counted in the final tally since observers participate in the voting process which can be summarised as follows: On the election day, the voter, after being authenticated by an authority, receives a blank ballot, makes his choice in a polling-booth and casts it into a ballot box in front of the authority. Then, the voter signs the record list to indicate that he has voted. After the voting period is completed, the ballot box is opened and the ballots are counted by the authorities. The counting result is announced. After all counting results are combined, election result is publicised. Each voter casts his vote by himself without any influence and nobody can see his vote except himself. A voter cannot cast more than one vote. Vote collecting, counting and tabulating are done in front of observers publicly. Meanwhile, representatives of political parties, observers of independent non-governmental organisations and international organisations are welcome observe the election process.

When voting takes place in an electronic environment, possibility of fraud is unavoidable since ensuring trust is not an easy task. At any step in the voting process, voting results can be manipulated if there is lack of verification and validation. Majority of people may accept and use electronic voting, but people have some concerns about the privacy, security and accuracy of the e-voting. They cannot easily trust the electronic voting system unless they individually verify that their votes are cast, recorded and counted correctly. Individual verifiability is important to raise public trust in especially Internet voting.

Most of the research studies in the literature do not consider the voting dilemma; they generally choose to focus on either individual verifiability or receipt-freeness. Most of them sacrifice receipt-freeness at the cost of accuracy and individual verifiability. A few protocols, which claim that they satisfy receipt-freeness, provide only universal verifiability or even worse no verifiability. In the literature, there is no protocol which

satisfies receipt-freeness, uncoercibility and individual verifiability at the same time, even with conditions or assumptions.

3.8 A Solution to Voting Dilemma: Predefined Fake Vote (PreFote) Scheme

In this section, an applicable solution namely Predefined Fake Vote (PreFote) scheme is proposed in order to overcome the voting problem. PreFote scheme uses an intentionally prepared predefined fake vote list where each PreFote consists of a unique code and an associated candidate from the candidates list.

PreFote list is prepared just before the election starts. Authorities participate in the PreFote list generation process. For each candidate, a constant threshold number of PreFotes are generated and listed in the PreFote list. In order to use the PreFote list, every voter obtains a unique check code (CCode) with his ballot. Voters also learn a set of PreFotes in order to use in case of coercion. At the end of the election PreFote list and real CCodes with revealed actual votes are published together in random order. Voter uses his real CCode for individual verifiability and directly checks his vote from the published list. The PreFote list does not affect the result of the tally, since the published list is only used for individual verifiability. Any protocol which uses the PreFote list should also announce another election result list without CCodes. Figure 3.3 depicts the PreFote list structure.

The CCode does not allow any voter to prove to anyone else how he voted, as nobody except the voter knows which CCode belongs to him. The voter can give a fake CCode to a coercer or vote buyer. The difference cannot be understood in the published list. It is not possible for any coercer or vote buyer to reveal the actual vote.

As an implementation detail, the PreFote list should not be announced directly. However, it can be known by some authorities. Section 6.2 and Chapter 8 demonstrates the usage of PreFote scheme.

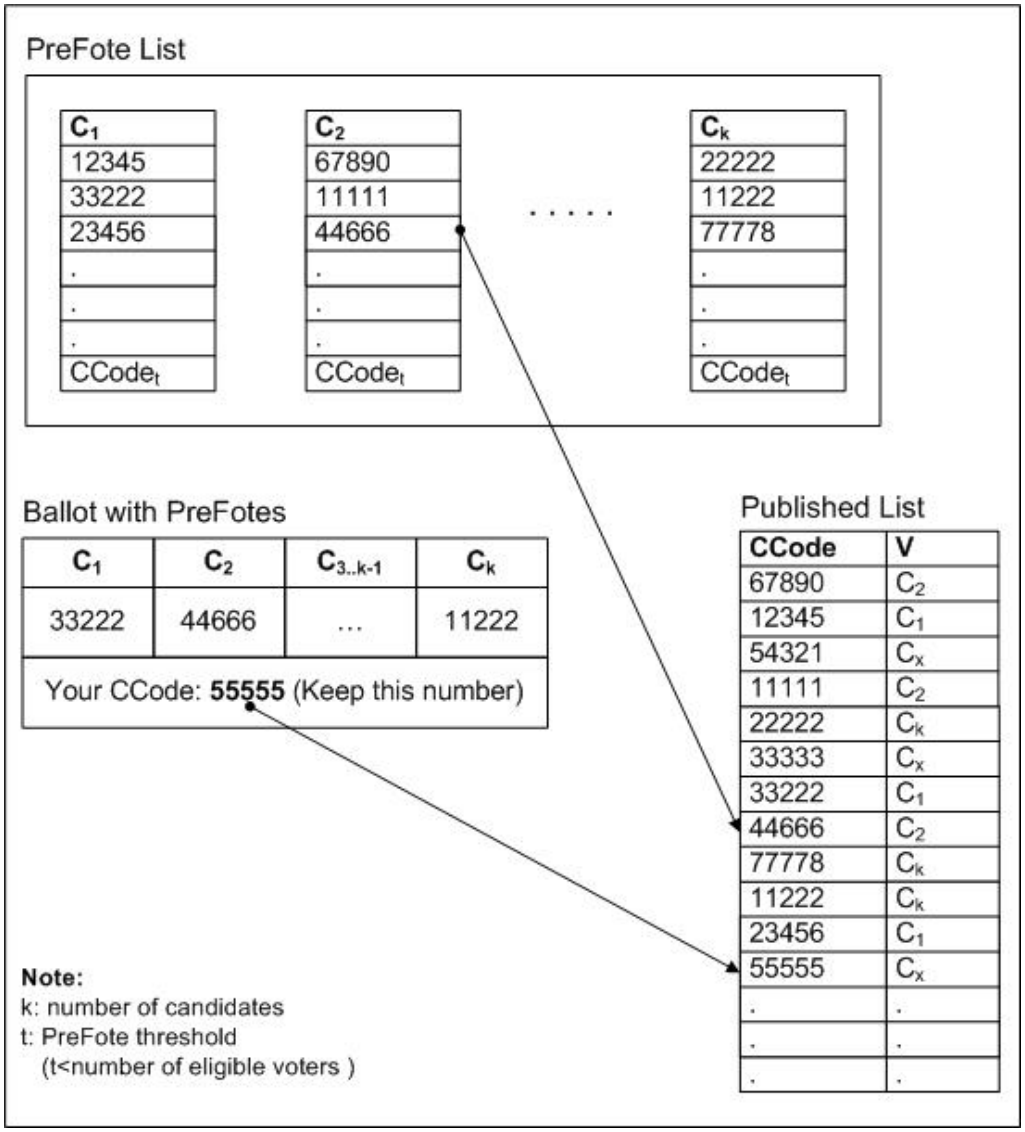

Figure 3.3: Predefined Fake Vote list structure.

CHAPTER 4

PRIVACY PRESERVING APPROACHES IN ELECTRONIC VOTING

This chapter provides a literature survey of privacy preserving approaches in electronic voting and presents a new classification of the voting protocols proposed in the literature.

4.1 Privacy in Voting Systems

A secure electronic voting protocol should be designed to counteract fraud and should not sacrifice voter privacy which can also be stated as unlinkability between any particular voter and his cast vote. Therefore, keeping the voter identity hidden is crucial in voting. Consequently, it should be impossible to associate a vote with a voter.

Anonymity is the primary requirement of e-voting protocols in order to safeguard voter privacy. Anonymity requirement makes electronic voting different from other electronic applications. It also makes fraud easier since addition, deletion, or modification of anonymous votes is harder to detect. Hence, various techniques have been proposed in order to satisfy anonymity and they are used in many electronic voting protocols in order to assure privacy.

Privacy preserving approaches used in electronic voting protocols could be classified into three categories: a) Using mix-nets, b) Using homomorphic encryption and c) Using blind signature. Unlike other classifications, blind signature approach is divided into three sub categories: i) Assuming anonymous channels, ii) Using blind signature without anonymous channels and iii) Using blindly signed identities. Though each approach has pros and cons, the common drawback is that they are unpractical due to having large computational and communicational complexity. Therefore, recent studies try to improve the efficiency of voting protocols [4], [22], [49].

4.2 Using Mix-nets

Mix-networks (mix-nets) are the most common approach to achieving anonymity. The general concept of mix nets is based on permuting and shuffling the messages in order to hide the relation between the message and its sender. However, the details, as to the implementation of mixing protocols, change depending on configurations and arrangements of mix-nets.

A mix-net typically consists of a set of mix servers which are responsible for mixing the incoming inputs and producing a shuffled output. In mix-nets, there are n mix-servers $M_1, \ldots, M_n$; each with its own public key E_i and private key D_i. Each server processes the input messages. The process can be either re-encryption or decryption depending on the mix-net types. Then, each server permutes the processed messages and forwards them to the next mix server.

The first mix-nets are decryption mix-nets [18], [83], [56] where messages are wrapped in several layers of encryption and then are routed through mix servers, each of which peels off a layer of encryption and then forwards them in random order to the next one. In decryption mix-nets, decryption in each mix server is repeated until all layers are removed. One of the well-known implementation of decryption mix-nets was Onion routing [14], [46]. Later re-encryption mix-nets were introduced [92], [47], [57] where the incoming messages are not decrypted, but re-encrypted in each mix server. In re-encryption mix-nets, decryption occurs after shuffling is completed.

The major drawback of the decryption and re-encryption mix-nets is that one server may compromise and cheat by removing or replacing any number of items. Therefore, they are extended to be verifiable. In verifiable mix-nets, a mix server additionally has to prove in zero knowledge that it decrypts/re-encrypts and shuffles the inputs correctly. There are several approaches to obtaining verifiable mix-nets; the main difficulty in these approaches is inefficiency of proof techniques [84], [1], [43], [77]. The call for proving that the mixing is correct causes an excessive computational cost for mix servers, so their implementation is not practical.

Using mix-nets in voting protocols is generally called as mix voting. As a general approach, a voter casts his vote over a mix-net, and it is assumed that a vote cannot be linked to a particular voter. In mix-net based voting protocols, voters prepare their ballots stating for whom they wish to vote and encrypt their ballots. Then, they send their cast

ballots to the mix-network. Firstly, mix server takes the list of the encrypted votes and mixes them in a random order. Later, it re-encrypts/decrypts the votes and forwards all votes to the next mix server. The next mix server takes the votes and shuffles them in the same way as the first server. Successively, each mix server takes the votes sent by the previous server, shuffles them and sends the produced list to the next mix server. The list produced by the last mix server is called the final votes list. The list is counted after the final decryption/encryption and published. Figure 4.1 shows a general view of mix-net based voting protocols.

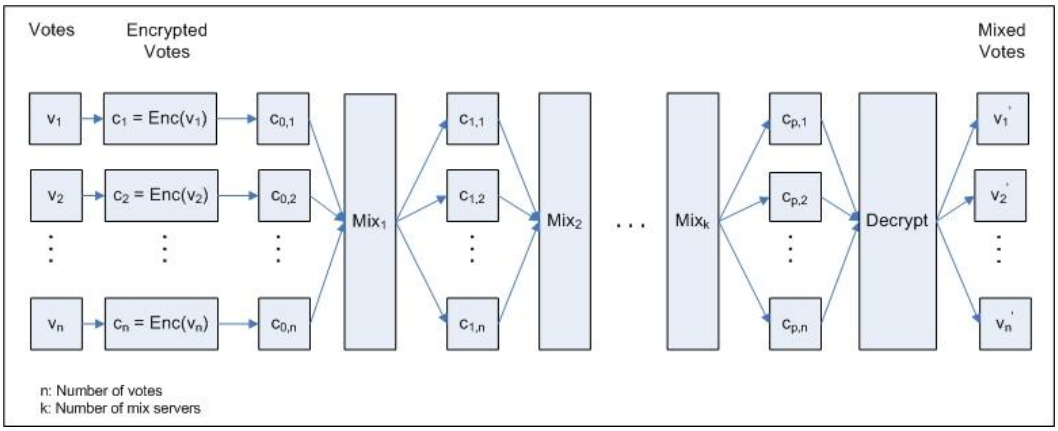

Figure 4.1: Mix-net based voting protocols.

Some of the protocols in this type have different implementations. VoteHere VHTi [100] is a commercial implementation which focuses on voter-verifiability. SureVote is an enhancement of the mix-net approach by Chaum, which incorporates a voter-verifiable component and uses proprietary printing equipment [21].

4.3 Using Homomorphic Encryption

Another commonly proposed way of achieving privacy in voting protocols is to use homomorphic encryption. A cryptosystem is homomorphic when $E(s_1) \circ E(s_2) = E(s_1 \lozenge s_2)$, where E is a public encryption function, s is a secret message, and $\circ$ and $\lozenge$ are some binary operators. Note that the binary operators may be equal. Thus, it is possible to compute the combination of the individual messages without having to retrieve the

individual messages themselves. Thereby, the individual messages can remain confidential. Two popular examples of homomorphic cryptosystems are ElGamal [39] and Paillier [82] cryptosystems.

Homomorphic encryption can be described in formal as follows. The probabilistic encryption function is $E_{pk} : R \times P \rightarrow C$, where R is the randomness space, P is the plaintext space and C the ciphertext space. The basic property of the encryption scheme is that $D_{sk}(E_{sk}(\cdot, x)) = x$ for all x. For homomorphic encryption, we assume additionally the operations $\Diamond$, +, $\circ$ defined over the respective spaces P, R, C, so that $<P, \Diamond>$, $<R, +>$, are additive groups and $<C, \circ>$ is a multiplicative group. An encryption function E is homomorphic if, for all $r_1, r_2 \in R$ and all $x_1, x_2 \in P$, it holds that:

$$E_{pk}(r_1, x_1) \circ E_{pk}(r_2, x_2) = E_{pk}(r_1 + r_2, x_1 \Diamond x_2)$$

In voting protocols based on homomorphic encryption, as the encrypted votes gather, it results in the accumulation of votes. The voting result is then obtained from the accumulation of votes while no individual ballot is opened and the corresponding individual vote remains secret. Figure 4.2 displays an overall view of homomorphic encryption based voting protocols.

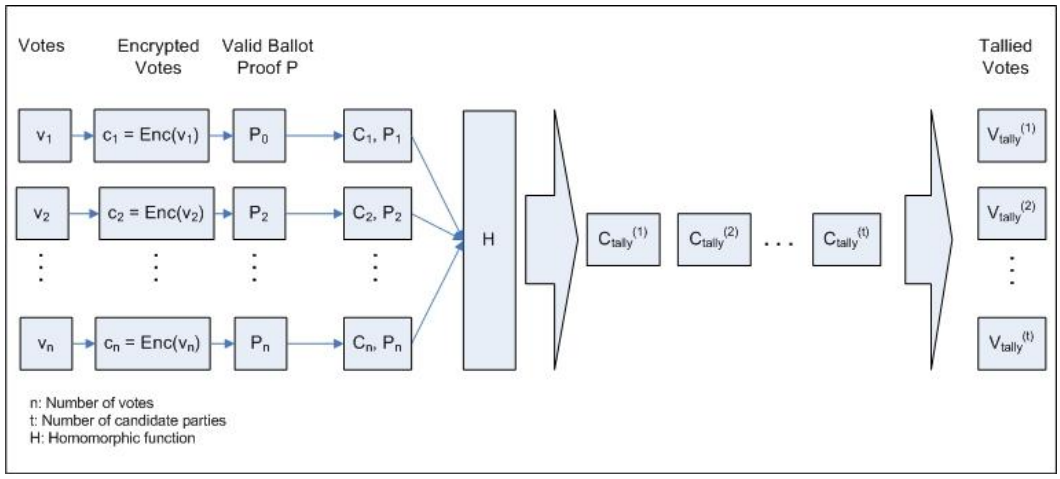

Figure 4.2: Homomorphic encryption based voting protocols.

In homomorphic encryption based protocols [9], [27], [2], [53], [91], [7], voting results are obtained easily so ballot tabulations are conducted more efficiently when the number of candidates or choices is small. However, homomorphic voting has a drawback where each vote must be verified to be valid since correctness of the tallying cannot be guaranteed without validation. When the number of candidates or choices is large, computational and communicational cost for the proof and verification of vote validity is so large that homomorphic voting actually becomes inefficient for large scale elections. A great advantage of this approach is that voters may openly authenticate themselves to the voting servers; there is no need for anonymous channels to ensure voter privacy.

Electronic voting protocols based on homomorphic encryption have more security properties than other protocols, but their communication complexity is quite high. They are most suitable for *yes-no* or *1-out-of-L* voting. A known implementation of this approach can be found in a European Union project; the CyberVote project [29], funded by the European Commission, has developed a prototype system.

4.4 Using Blind Signature

There are some e-voting protocols in the literature which use neither mix-nets nor homomorphic encryption; they use blind signature scheme in different stages of the voting process in order to assure voter privacy. Up to now, these protocols have employed blind signature on empty ballot, voter's vote or part of the vote.

Unlike other classifications, blind signature approach is divided into three sub categories: i) Assuming anonymous channels, ii) Using blind signature without anonymous channels and iii) Using blindly signed identities. Many researchers do not make such a sub categorisation and classify the protocols as described in [42], [28], [81] and some others as blind signature based voting protocols. However these protocols still require the existence of anonymous channels or apply some cryptographic techniques. For example, in many papers, [42] is stated as blind signature based voting protocol. However, the voter sends his vote to the counter authority through an anonymous communication channel in that protocol. In other words, it is not appropriate to classify this kind of protocols as blind signature based when they require anonymous channels besides blind signature.

4.4.1 Assuming Anonymous Channels

Anonymous channel is a communication channel guaranteeing the anonymity of the sender. The recipient that has been sent a message through the anonymous channel does not know the identity of the sender. No one is able to trace the message back to the sender.

Various techniques have been proposed in order to achieve anonymous communication; however the most common solution is mix-nets. An alternative (anonymous channel) solution to mix-nets is a system referred as "crowds" [86], participants of which want to protect each other's privacy. When one of them wants to send a message somewhere, he sends it to one of the members of the group. This member either sends it to its destination, or passes it on to another group member. It provides anonymity; however, it may not be practical for electronic voting since it requires group members, i.e. voters, to be available for each other during the voting process.

Most e-voting protocols assume an efficient anonymous channel. [42], [28], [6], and [81] assume that the voter has an access to an anonymous channel at any point during the voting process. [60] uses also anonymous channels, but assumes that it is provided by an untraceable e-mail system.

These types of protocols look like mix-net based protocols; however, they generally do not explain any detail about the implementation of anonymous communication, and they only assume the existence of an anonymous channel. The difference between these protocols and mix-net based protocols is that mix-net based protocols use mix-nets to disassociate the relation between the voter and his vote, i.e. mix-nets provide both unlinkability and untraceability. The protocols in this category use anonymous channels to provide untraceability only, whereas they provide unlinkability by using different means.

The idea behind these protocols is that the voter prepares a ballot stating for whom he wishes to vote. He either obtains an authorised ballot or interacts with an authentication authority to make his vote authorised. Finally, he sends his cast ballot to another authority that is responsible for counting votes through an anonymous channel in order to preserve the privacy. After all ballots have been collected, votes can be counted.

There are several implementations which use blind signature and have been piloted in small scale elections. For example, the SENSUS system [28] was the first to be implemented. The Davenport *et al.* is another system [31] which was used to conduct student governmental elections. The EVOX system [52] was used at MIT for undergraduate association elections. DuRette [38] improved EVOX system in order to eliminate single entities capable of corrupting the election. Both DuRette's system and EVOX are very sensible to failures in communication or servers; these problems were solved by REVS which is proposed by Joaquim *et al.* [59] as another implementation based on DuRette's work. However, the DuRette's system has problems concerning the authentication of voters, allowing an easy impersonation of voters by the servers running the election. In REVS, this problem was solved by means of redesigning the voters' authentication algorithm. Later, some improvements were done on REVS to make it more robust [66]. The Votopia project [65], created jointly by Korean and Japanese developers, was tested in the election of the MVP (most valuable player) in the Soccer World Cup of 2002. Votopia is not publicly accessible and does not provide anonymity.

Since these protocols assume the existence of an anonymous channel, their security depends on the reliability of the anonymous channel. Efficient and secure anonymous channel implementations can make these protocols applicable.

4.4.2 Using Blind Signature without Anonymous Channel Assumption

[75], [50], [85], [102] employ blind signature to obtain signed ballots or voting tickets in order to assure voter privacy. These protocols suffer from accuracy as corrupted participants can make fraud without being detected. Besides they have no solution for uncoercibility and IP traceability.

The idea behind blind signature based protocols is that the voter prepares a ballot stating for whom he wishes to vote. He then interacts with an authentication authority who issues a blind signature on the ballot. Informally, this means that the voter obtains the authority's digital signature on the ballot, without the authority learning any information about the content of the ballot. Finally, all voters send their ballots to another authority that is responsible for counting the votes, and that will only accept ballots signed by the authentication authority. After all ballots have been collected, votes can just be counted.

These protocols are not very popular since they generally lack accuracy. Thus, this approach will not comparatively be evaluated. Nevertheless, they are valuable since they try to get benefit of the practicality and efficiency of blind signatures.

4.4.3 Blindly Signed Identities

In computer networks, pseudonyms possess varying degrees of anonymity, ranging over highly linkable public pseudonyms (the link between the pseudonym and a human being is publicly known or easy to discover), potentially linkable non-public pseudonyms (the link is known to system operators but is not publicly disclosed), and unlinkable pseudonyms (the link is not known to system operators and cannot be determined). Blind signature is an efficient scheme to provide unlinkable pseudonyms.

In the next chapter, a privacy preserving approach using unlinkable pseudonyms, namely Pseudo-Voter Identity (PVID) scheme, is introduced. PVID scheme employs the blind signature, in a different way from the existing voting protocols. It employs the blind signature on voter's pseudo identities instead of empty ballots or votes. In this scheme, voters obtain blindly signed pseudo identities that nobody can map to the voter's registration identity. PVID scheme is one and only example of this category and first introduced in [17]. An electronic voting protocol, which uses blindly signed identities by employing PVID scheme, is proposed in Chapter 7 as well.

4.5 Taxonomy of Electronic Voting Protocols

Sampigethaya and Poovendran [93] presented a framework of electronic voting protocols, which enables a comparative analysis of them, and they compared the most well-known cryptographic voting protocols with each other. In this section we provide a comparison adapted from this work, and we will alter this comparison by adding DynaVote in Section 8.4.

We have made some minor changes in the original table since the requirement definitions are not standard, which could lead to a misunderstanding. The most important change is in Juels *et al*. [61] and Benaloh [8], we claim that they do not satisfy individual verifiability, but they only satisfy universal verifiability with respect to the definitions in Section 3.2. This change is done in accordance with the definition of individual verifiability in the original work.

We separated individual verifiability whereas it is handled with universal verifiability together as verifiability in the original work. The authors also note that universal verifiability is directly related to accuracy. We did not add universal verifiability to the table due to the fact that the result column is the same as accuracy column. This situation strengthens the remark we made in Section 3.2.

The authors state that dispute-freeness is related to universal verifiability. They define dispute-freeness in these words: “any voting scheme must provide a mechanism to resolve all disputes in any stage.” Any careful reader can notice that dispute-freeness should be in fact a part of the accuracy. However, we also listed it in the table in order to emphasise the contribution of DynaVote. Ambiguity in the definitions of the original work points out the value of our contribution in 1.3.6.

The definition of robustness in the original work is similar to accuracy. Furthermore, none of the protocols satisfy robustness directly since it is a system level requirement. Thus, robustness is not added to the table. Scalability and practicality are listed in the original work since they are the most important system requirements. So, they are listed in the table so as to stress DynaVote’s contribution. Voting protocols which use blind signature without anonymous channel assumption are not compared in the original work. These protocols and some recent studies are added as well.

By using the comparison in Table 4.1, the following observations can be made. All protocols satisfy privacy, eligibility and uniqueness. In the blind signature based approach, the actual votes are published, and anybody can count them, but nobody knows who sent which vote. However, a special care is required to achieve eligibility, to ensure that the voter cannot cast more votes and to prevent improper voters from voting. Cryptographic voting protocols using mix-nets have similar characteristics. On the other hand, in homomorphic approach, the eligibility is easily performed since there is no individual vote, and all votes are accumulated. However, anybody can see which voters have voted and which have not.

Table 4.1: Taxonomy of cryptographic voting protocols.

Privacy Preserving Approaches	Voting Protocols	Security Requirements								System Requirements	
		Privacy	Eligibility Uniqueness	Fairness	Individual Verifiability	Accuracy	Dispute-freeness	Receipt-freeness	Uncoercibility	Scalability	Practicality
Mix-Nets	Chaum, 1981 [18]	Com	Y	N	Y	N	N	N	N	N	N
	Benaloh, 1987 [8]	Com	Y	C	N	Y	N	N	N	N	Y
	Chaum, 1988 [20]	Com	Y	N	Y	N	N	N	N	N	N
	Sako and Killian, 1995 [92]	Com	Y	C	N	Y	N	Y	N	N	N
	Chaum, 2004 [21]	Com	Y	C	C	C	N	Y	N	Y	C
	Chaum, 2005 [22]	Com	Y	C	N	C	N	Y	N	N	C
Homomorphic Encryption	Cohen and Fischer, 1985 [24]	Com	Y	N	N	Y	N	N	N	N	N
	Cohen and Yung, 1986 [25]	Com	Y	C	N	Y	N	N	N	N	N
	Iverson, 1992 [55]	Com	Y	C	Y	C	N	N	N	N	C
	Sako and Killian, 1994 [91]	Com	Y	C	N	Y	N	N	N	N	Y
	Cramer et al., 1996 [26]	Com	Y	C	N	Y	N	N	N	N	Y
	Cramer et al., 1997 [27]	Com	Y	C	N	Y	N	N	N	C	C
	Schoenmakers, 1999 [94]	Com	Y	C	N	Y	Y	N	N	N	Y
	Hirt and Sako, 2000 [53]	Com	Y	C	N	Y	N	Y	N	N	C
	Baudron et al., 2001 [7]	Com	Y	C	N	Y	N	Y	N	C	C
	Lee and Kim, 2002 [67]	Com	Y	C	N	Y	N	Y	N	Y	N
	Kiayias and Yung, 2002 [64]	Com	Y	C	N	Y	Y	N	N	N	Y
	Acquisti, 2004 [2]	Com	Y	C	N	C	N	Y	C	N	N
Blind Signature with Anonymous Channels	Fujioka et al., 1992 [42]	Com	Y	Y	Y	N	N	N	N	Y	N
	Baraani et al., 1994 [6]	Com	Y	Y	Y	C	N	N	N	N	N
	Cranor and Cytron, 1997 [28]	Com	Y	C	Y	C	Y	N	N	C	N
	Okamoto, 1997 [81]	Com	Y	C	C	N	N	Y	N	N	N
	Juang et al., 2002 [60]	Com	Y	C	Y	C	N	N	N	Y	Y
	Golle et al., 2002 [48]	Com	Y	C	Y	C	N	N	N	C	Y
	Lee at al., 2003 [68]	Com	Y	C	N	Y	N	Y	N	C	N
	Juels et al., 2005 [61]	Com	Y	C	N	C	N	Y	C	N	N
Blind Signature without Anonymous Channels	Mu and Varadharajan, 1998 [75]	Com	Y	C	N	C	N	N	N	N	C
	He and Su, 1999 [50]	Com	Y	C	C	C	N	N	N	N	C
	Ray et al., 2001 [85]	Com	Y	C	Y	Y	N	N	N	C	N
	Yang et al., 2004 [102]	Com	Y	C	N	Y	N	N	N	N	N

Com: Computational
C: Conditionally satisfied
N: No, not satisfied
Y: Yes, satisfied

All protocols provide conditional fairness except [42], [28] and [6]. In these protocols, a voter should keep some part of the vote (e.g. encryption keys) until the end of the election, and he should participate in the counting stage after the election has been completed. However, this is not desired and thus, these protocols are far from practicality. Because of this fact, assuring fairness by voter participation in the counting stage is not preferred. As a consequence, conditional fairness is accepted as de-facto fairness standard for electronic voting protocols.

In general, existing voting protocols fail to provide a solution against coercibility. Ensuring receipt-freeness is another cumbersome task for many of the protocols.

Moreover, most of them could not provide scalability and practicality. The first practical electronic voting protocol for large scale elections ensuring both privacy and fairness is of Fujioka *et al.* [42]. However, accuracy can be violated by the malicious authority, if any, who can add votes in the event that some voters abstain from voting in counting stage. The voting protocol proposed by Baraani and Tuinstra [6] extends [42]. The model of the original protocol has been further modified with the addition of a trusted third party. Later, Okamoto [81] proposed a solution for large scale elections based on untappable channel and even stronger physical assumptions whereas the protocol suffers from practicality. In general, the voting protocols, stating that they satisfy practicality and privacy, have strong assumptions such as anonymous communication channels and mix-nets. They are disposed to computational costs as they have to prove that their anonymizing is correct.

Almost all homomorphic encryption based voting protocols provide accuracy by sacrificing individual verifiability. On the other hand, blind signature based and mix-nets based voting protocols can easily fulfil individual verifiability, but they have problems satisfying accuracy.

Achieving receipt-freeness is relatively easy by the help of zero-knowledge proof techniques in homomorphic encryption based voting protocols. On the other hand, in blind signature based or mix-net based protocols, voter chosen randomness can be used as a receipt. The voter can prove the content of his encrypted ballot using his knowledge of randomness.

Up to now, none of the electronic voting protocols has been able to satisfy receipt-freeness, uncoercibility and individual verifiability at the same time. As a result no practical and secure cryptographic voting protocol has been proposed which satisfies all electronic voting security requirements. DynaVote is the first protocol that achieves all of the security requirements.

CHAPTER 5

PSEUDO-VOTER IDENTITY (PVID) SCHEME

As discussed in the previous chapter, each existing privacy preserving approaches have both advantages and disadvantages and they suffer from the lack of practicality. In this chapter, we propose practical and low cost solution to satisfy voter privacy. The proposal mainly focuses on privacy in electronic voting; however, it may be used for other electronic applications such as electronic auction, electronic donation, etc.

This chapter explains Pseudo-Voter Identity (PVID) scheme in detail. Firstly, it provides definitions used in PVID scheme. Then, it introduces PVID scheme by explaining stages and properties. Later, it compares PVID scheme with the existing privacy preserving approaches. Implementation details are given at the end of the chapter.

5.1 Introduction

In this chapter, we propose PVID scheme based on blind signature in order to achieve voter privacy in electronic voting protocols. In PVID scheme, voter prepares a list of blinded identities and then he obtains blind signature for each of them separately by interacting with the approval authority in one session. Later, voter extracts anonymous pseudo identities (PVIDs) which are unlinkable to voter's registration identity. Each PVID is selected by the voter and blindly signed by the approval authority after verifying voter's eligibility. Thus, nobody knows the value of PVID except voter.

In existing voting protocols, voter generally uses his real identity while communicating with the authorities. On the other hand, in PVID scheme, voter uses pseudo identities, which have no relation with the voter's real identity and are unlinkable to it. Voter can use them throughout the entire communication and he can easily hide his real identity.

PVID is a practical scheme since it employs only blind signature to obtain PVID Authority's signature. RSA is used as a public key cryptosystem. A pseudo random number generator is used to feed PVID with a random number. Threshold cryptography is employed in order to prevent PVID Authority corruption. Hence, PVID scheme provides privacy without requiring any complex mechanisms and computational operations. In order to prove the practicality of the PVID scheme, we have implemented it with Java over Internet as a proof of concept.

Up to now, several election protocols employing blind signature in different stages of the voting process have been proposed [42], [28], [60], [69]. All these protocols employ blind signature on empty ballot, voter's vote or part of the vote. On the other hand, we employ blind signature on voter's pseudo identities and so, voter obtains blindly signed pseudo identities.

5.2 Definitions

Prior to explaining the PVID scheme, we briefly present the definitions of anonymity, anonymous, pseudonymity, pseudonym and pseudonymous:

Anonymity: "Anonymity ensures that a subject may use a resource or service without disclosing its user identity." [54].

Anonymous: A subject can be said to be anonymous towards another subject in a particular transaction if his identity in that transaction is concealed from that other subject. Anonymity of any subject is thus always considered and specified with respect to one or more specific other subjects in the transaction. [51]

Pseudonymity: "Pseudonymity ensures that a subject may use a resource or service without disclosing its identity, but can still be accountable for that use. The subject can be accountable by directly being related to a reference (alias), or by providing an alias that will be used for processing purposes, such as an account number." [54].

Pseudonym: A pseudonym is an identifier with a local meaning. A user may choose or create his pseudonym; or, organisations issuing certificates or credentials may create pseudonyms for users. [51]

Pseudonymous: A transaction carried out under a pseudonym is a pseudonymous transaction. The use of pseudonyms assumes that it is not trivial, for at least some participants in the system or for outsiders, to derive a real identity from the pseudonym. According to the definition of anonymity, the user in a pseudonymous transaction is anonymous towards the party or parties that cannot map the pseudonym used to the user's real identity. [51]

We proceed by stating the definitions used in the proposed scheme: Pseudo-Voter, PVID Authority, Pseudo-Voter Identity, and Pseudo-Voter Identity List.

Pseudo-Voter is a voter who has anonymous credentials to access to the voting system. It can be called as anonymous voter instead of pseudo-voter; however, this may cause some misunderstandings since voter selects his identity. So, we prefer to use "Pseudo".

PVID Authority is the approval authority which blindly signs pseudo-voter identities. It is responsible for verifying voters' eligibility and it carries out the authentication and authorisation stage in the voting process.

Pseudo-Voter Identity (PVID) is an identity used by Pseudo-Voter. More precisely, PVID is an anonymous pseudo identity which is unlinkable to voter's registration identity. In other words, PVID is an unlinkable pseudonym that nobody can map it to the voter's registration identity. PVID is selected by the voter and blindly signed by PVID Authority. Thus, nobody knows the value of PVID except voter. When voter employs PVID scheme, he obtains anonymous pseudo credentials, so we call him as pseudo-voter instead of voter. He is a real voter but the identity used is pseudo.

Pseudo-Voter Identity List (PVID-list) is a list of PVIDs used to interact with the voting authorities. PVID-list is employed so as to prevent the voting authorities' corruption and to strengthen the accuracy and fairness in the electronic voting protocols.

5.3 PVID Scheme Overview

Voter has a registration identity (RegID) which can be any widely used identity such as national identity number or social security number. RegID can be a government-issued voter ID as well. On the election day, voter uses his RegID to authenticate himself to the system. In almost all blind signature based voting protocols, voter tries to obtain blindly

signed ballot and/or his cast or part of them. In PVID scheme, voter obtains a list of blindly signed anonymous pseudo identities and uses them instead of real RegID while interacting with the authorities.

An approval authority, namely PVID Authority, is employed to issue blind signature on voter's PVID-list after checking voter's eligibility. The trustworthiness of PVID Authority is very important, since it can blindly sign ineligible people's PVID-lists without being detected. Therefore, we employ threshold cryptography to prevent corruption of PVID Authority. Threshold cryptography is applied to distribute the power over *n* participants. In order to sign any request at least *t* participants should come together. In this case, *t* over *n* participants should be corrupted to issue fake PVIDs.

As stated in Section 3.1, any voting process can be divided into 4 stages: registration, voter authentication & authorisation, voting and tallying. By using PVID scheme, the authentication & authorisation stage is clearly separated from voting stage. As soon as obtaining PVID-list, voter can vote at any time by providing PVIDs to the voting authorities. The voting authorities only check PVID Authority's signature on the PVIDs. From now on, voter becomes anonymous voter without need of anonymous channel. Voter uses the voting system twice by using RegID and PVIDs respectively. RegID is used in order to communicate with PVID Authority for authentication purposes and PVIDs are used for communicating with the remaining authorities.

PVID scheme has four stages: ID generation stage, blinding stage, signing stage and PVID obtaining stage. The details of the scheme are given in the following section.

5.4 PVID Scheme

The following notation is used:

(β_a, δ_a): {(e, n), (d, n)} PVID Authority's public and private keys.

(β_v, δ_v): Voter's public and private keys.

$\check{\mathbf{E}}_x(m)$: Encryption of message *m* with the public key of actor *x*.

$\check{\mathbf{D}}_x(m)$: Decryption of message *m* with the private key of actor *x*.

$\check{\mathbf{S}}_x(m)$: Sign of message *m* with the private key of actor *x*.

$\check{\mathbf{U}}_x(m)$: Unsign of message *m* with the public key of actor *x*.

ID-list = $\{ID_1, ID_2 \ldots ID_k\}$ where ID_i is i^{th} pseudo identity chosen by the voter.

PVID-list = $\{PVID_1, PVID_2 \ldots PVID_k\}$ where $PVID_i$ is i^{th} PVID which is blindly signed pseudo identity by PVID Authority.

PVID-list is a list of blindly signed identities and it is required to be random and unique for each voter. Hence, each ID contains a big random number in addition to the election data which uniquely specify the election. Election data can be some pre-determined keywords such as election name, election date, election id …etc. Voting authorities can simply verify PVID by applying PVID Authority's public key.

The number of PVIDs used in the voting protocol varies regarding to the protocol details. For instance, some protocols have more than one authority such as ballot distributor, key generator, counter, verifier … etc. We employ different PVIDs instead of a single PVID for each authority in order to prevent any corrupted authority to impersonate the voter. Hence, each ID contains authority data which specify the purpose of the PVID and can be authority name, authority's public key …etc.

If the voting protocol has just a single authority, ID-list and PVID-list become single element lists. Each voting protocol should have at least one authority; otherwise, voter could not cast his vote. So, the number of elements in PVID-list is at least one. Figure 5.1 illustrates how the PVID scheme works in summary. Details of each stage are explained the following sections.

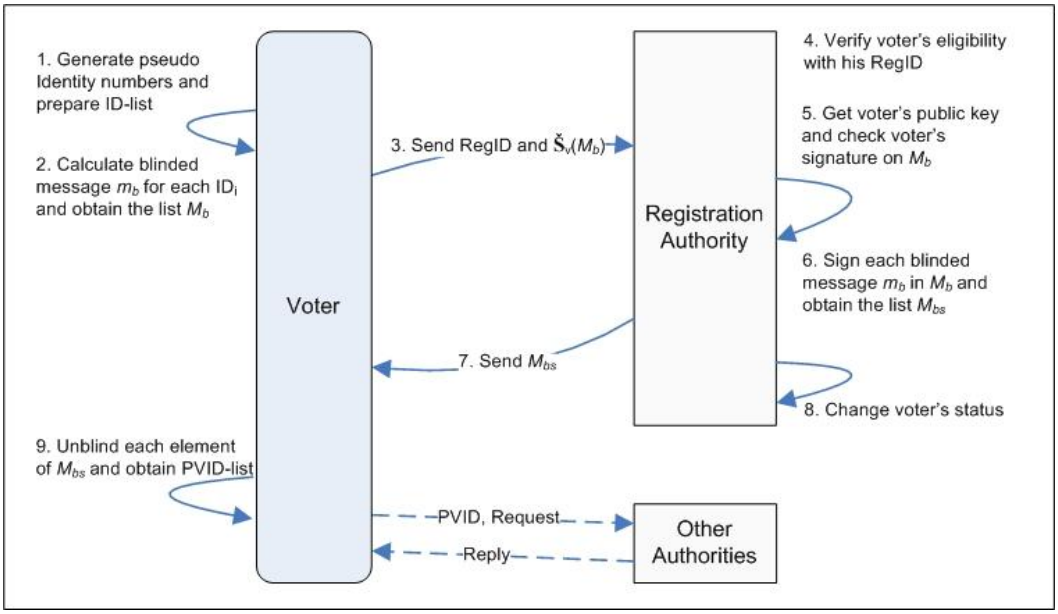

Figure 5.1: PVID scheme.

5.4.1 ID Generation Stage

Voter generates *k* pseudo identity numbers and prepares ID-list. Each ID contains the election data, authority data (the details about the usage purpose) and a big random number (generated by a PRNG) as shown in Figure 5.2, so it is constructed as following. For each ID, the authority data should be different whereas the random number should be same. Using same random number provides that IDs belong to one voter.

ID_i = *(Election Data, Authority Data, Random Number)*

ID-list = $\{ID_1, ID_2 \ldots ID_k \mid ID_i$ *is* i^{th} *pseudo identity}*

Now, voter has an ID-list that he wishes to have signed each ID_i in the list by PVID Authority. Voter does not want PVID Authority to learn anything about ID_i.

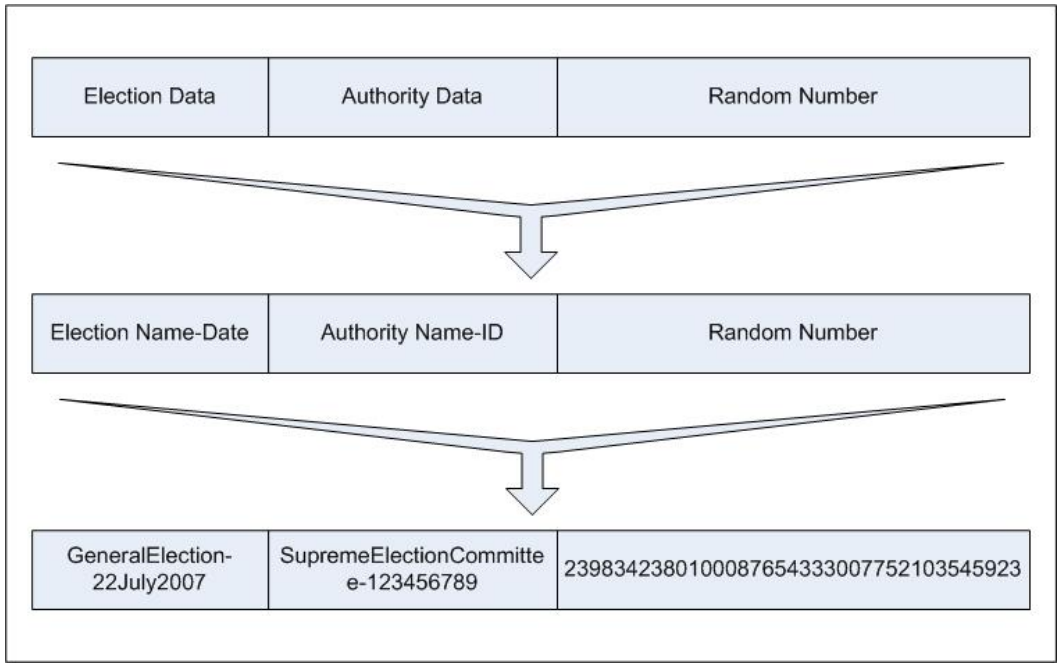

Figure 5.2: ID-list details.

5.4.2 Blinding Stage

Voter generates a random blinding factor number r and calculates blinded message m_b for each ID_i, and obtains a list of blinded IDs which is M_b as shown in Figure 5.3.

$$m_{bi} = (r^e[ID_i]) \bmod n \quad \text{where} \quad \gcd(r,n) = 1$$

$$M_b = \{m_{b_1}, m_{b_2}, ..., m_{b_k}\}$$

Voter signs the list M_b and obtains $\check{\mathbf{S}}_v(M_b)$. Then, he encrypts his RegID and $\check{\mathbf{S}}_v(M_b)$ with PVID Authority's public key and obtains message $\check{\mathbf{E}}_a(\text{RegID}, \check{\mathbf{S}}_v(M_b))$. Voter sends this message to PVID Authority. The value m_b is "*blinded*" by the random value r; hence PVID Authority cannot derive any useful information from it.

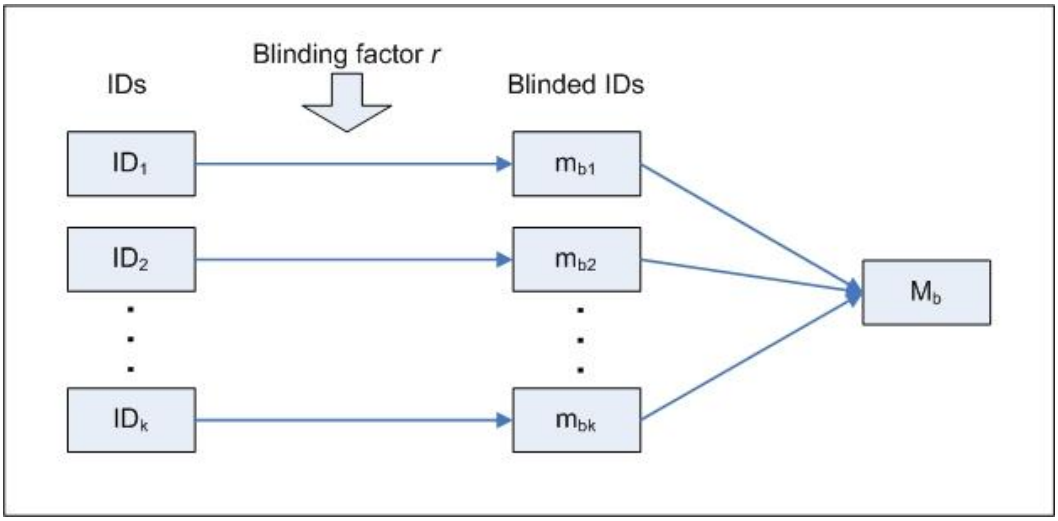

Figure 5.3: Blinding stage.

5.4.3 Signing Stage

PVID Authority decrypts the received message and obtains the voter's RegID and $\check{\mathbf{S}}_v(M_b)$. PVID Authority verifies voter's eligibility with his RegID. If voter is eligible and has not made any request yet, PVID authority employs voter's public key and checks voter's signature on M_b.

For eligible voters, PVID Authority signs each blinded message m_b in the list M_b and calculates m_{bs}. Subsequently, PVID Authority obtains a list of blindly signed IDs which is M_{bs}. The process is depicted in Figure 5.4.

$$m_{bs_i} = m_{b_i}{}^{d} \bmod n$$

$$M_{bs} = \{m_{bs_1}, m_{bs_2}, ..., m_{bs_k}\}$$

Then PVID Authority encrypts the list M_{bs} with the voter's public key and sends $\check{\mathbf{E}}_v(M_{bs})$ to the voter. At this point, in order to supply only one PVID for each eligible voter, PVID authority changes the voter's status.

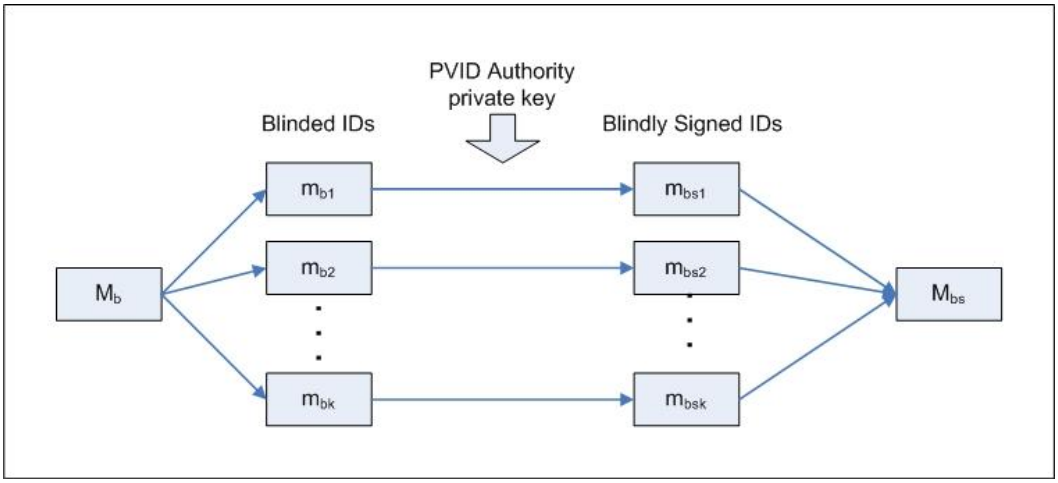

Figure 5.4: Signing stage.

5.4.4 PVID Obtaining Stage

Voter decrypts the received message and obtains the blindly signed ID list M_{bs}. Voter can easily obtain PVIDs, the true sign of IDs, by removing the blinding factor r from each m_{bs}. Voter carries out the following operations for each m_{bs} in the list M_{bs} in order to obtain $PVID_i$ for each ID_i.

$$m_{bs_i} = m_{b_i}{}^{d} \bmod n = (r^{e}[ID_i])^{d} \bmod n$$

$$m_{bs_i} = r^{ed}[ID_i]^{d} \bmod n = r[ID_i]^{d} \bmod n$$

$$PVID_i = r^{-1} m_{bs_i} \bmod n = [ID_i]^{d} \bmod n$$

$PVID_i$ is the sign of PVID authority on the voter's selected ID_i. Later voter populates PVID-list with PVIDs as illustrated in Figure 5.5.

PVID-list = {$PVID_1$, $PVID_2$ … $PVID_k$}

Now, voter has valid and signed pseudo identities that are unlinkable to his real RegID. Voter can use them in any electronic voting protocol without providing his RegID to the voting authorities. Moreover, he can directly communicate with the

authorities without requiring any anonymous channel since PVIDs are unlinkable to his real identities.

When voter uses his PVID, the authority only verifies the signature on PVID by unsigning it with PVID Authority's public key and simply checking the Election Data and the Authority Data.

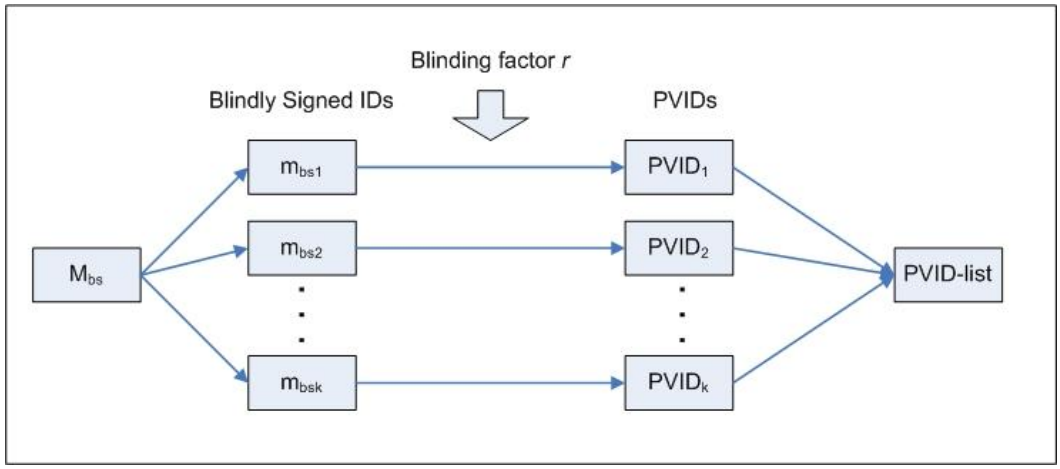

Figure 5.5: PVID obtaining stage.

5.5 Discussion

Privacy in PVID scheme relies on unlinkability between voter's pseudo identity and real identity. In order to prove any relation between them, the random number used to create blinded message should be known. Otherwise, adversary should break RSA cryptosystem since PVID scheme uses blind signature based on RSA public key cryptosystem, which is infeasible. The random number is generated by voter and nobody knows it.

In mix-net based protocols, voter could not communicate directly with counting authorities. On the other hand, in PVID scheme, voter is able to communicate directly with the counting authorities without any hesitation since PVIDs are unlinkable pseudo identities and voter himself is a pseudo-voter.

PVID scheme is highly flexible and is applicable for both voting pool type elections and wide area network based elections. In uncontrolled and unsupervised network environments, anonymous credentials can be source of the voter simulation problem; in other words, anonymous credentials can be used for vote buying/selling or can be usurped by coercer. So PVID scheme has similar problem. However, in fact, this is not the problem of PVID scheme; it is a general problem for all unsupervised network based voting protocols and other applications. Thus, vote buying/selling and coercion can happen even if PVID scheme is not applied. If the election takes place in uncontrolled and unsupervised environments e.g., in Internet, nothing could prevent coercibility and vote selling. In order to overcome this problem, PVID scheme provides firstly a list of PVIDs instead of a single PVID and secondly recasting of votes. By the help of vote recasting, there is no benefit for the vote buyer or coercer since voter can recast later. Thus, PVID scheme is an applicable and secure alternative solution for voting over a network problem.

5.5.1 Vote Recasting

When vote recasting is applied by using existing privacy preserving approaches, uniqueness could not be satisfied. Even worse, accuracy or privacy can be violated since recast votes could not be identified. The reason is that voter unlinkability is provided by using mix-nets or anonymous channels. By the nature of homomorphic encryption, recast votes are summed up so nobody can identify them. Thus existing privacy preserving approaches suffer from providing recasting and they could not handle it. Moreover, some protocols take measure against recasting to defend accuracy. When they provide recasting, they should renounce privacy and uniqueness. As a result, they are not suitable for recasting.

Due to the fact that PVID is not voter's real identity and counting authorities can keep PVIDs, PVID scheme allows vote recasting. Counting authorities store voter's vote with the associated PVID during the election period. It does not violate voter privacy as voter uses PVID. When voting authorities allow vote recasting then if someone coerces voter, voter casts by that way. Later, he can change his vote, by recasting a new one and overwriting the old one. Same idea can be applied to vote selling. So, practically there is no point to coerce voter or to buy vote from voter. PVID scheme makes vote selling more difficult, because the buyer now has to lock the seller until end of the election to

prevent the seller from changing his vote. Note that PVID scheme does not force recasting; one time voting can also be used. Whereas PVID Scheme provides unlinkability, untraceability should be handled as discussed in the following section in the protocols that employ PVID scheme.

5.5.2 IP Traceability

Up to now, almost none of the voting protocols discussed IP traceability in electronic voting since privacy is handled as unlinkability. It is because that untraceability is provided by default in mix-nets since they trust at least one mix-server. Homomorphic encryption provides better unlinkability and untraceability due to the nature of homomorphism; however it is not appropriate for individual verifiability. Blind signature is the weakest category about IP traceability. However most of the implemented protocols are based on blind signatures due to their remarkable practicality.

PVID scheme does not directly anonymize IP address, it anonymizes Voter ID. In order to achieve IP untraceability in the protocols which employs PVID scheme, some extra work should be done. Any of the methods described below can be applied. The protocols can assume that:

- Voters can use any IP anonymizer applications. ISP, Proxy or special software can provide this. It can be an implementation of anonymous channels. Note that implementing IP anonymizers is easier and more practical than implementing mix-nets.

- Voting can be done from a voting pool or any other public network. If voting pools are employed recasting can be prohibited. Official organisations can provide public voting pools. Voting pools does not to be closed and controlled networks.

- Authorities do not intend to reveal voters' IPs and they do not cooperate for this purpose. However, no single authority could be able to trace voter's IP; but a reasonable threshold value can be assumed.

In case of voter has a static IP and he did not take any care about it, if authorities corrupt, then IP untraceability may fail. However, this case is not usual and using a dynamic IP is encouraged.

Even though IP untraceability is discussed at protocol layer, the details are handled at implementation layer. PVID scheme has no constraints about IP addresses related to implementation and it can be implemented as open source. We assume that PVID authority has no communication with other authorities and it does not release any data.

5.5.3 PVID Support on E-voting Requirements

Any voting protocol which employs PVID scheme can easily fulfil some of the e-voting requirements in advance without requiring any extra work or with some small effort, such as privacy, eligibility uniqueness and uncoercibility.

Privacy (A particular voter and his cast vote are unlinkable.): PVID Authority issues a blind signature on voter's blinded ID. Since the blind signature scheme is used, any particular RegID is not linkable to any PVID and any particular PVID is not linkable to any RegID. Voter uses his PVID in voting process and does not use his RegID. Revealing the RegID is equivalent to breaking RSA.

Eligibility (Only eligible and authorised voters can vote.): PVID Authority issues a blind signature after verifying voter's eligibility. Only eligible voters' blinded IDs are blindly signed by PVID Authority. Ineligible people's blinded IDs cannot be signed without being detected since threshold cryptography is applied to distribute the authority over n parties. In order to sign any request at least t parties should assemble.

Uniqueness (Only one vote for each voter is counted.): Each encrypted vote cast to the counting authorities is attached with a unique PVID. Even if recasting is allowed in the voting protocol, in the counting stage only one vote, possibly the last vote depending on the election policy, is counted.

Uncoercibility (Voter cannot be coerced to cast his vote in a particular way.): When the voting authorities allow vote recasting, practically it is not possible to coerce the voter or to buy vote from the voter, since nobody can know whether the current vote will be the final one or not. Hence, there is always a trade-off between uncoercibility and vote recasting.

5.5.4 Comparison with Other Privacy Preserving Approaches

In Chapter 4 we have discussed mix-nets and homomorphic encryption based voting protocols. In this section we give a comparison of PVID scheme with these privacy preserving approaches. This comparison states that PVID scheme provides privacy in a more efficient and practical manner.

As stated before, voters need to prove the validity of the ballots in homomorphic encryption based voting protocols. Hence the computational cost for voter is relatively more in using homomorphic encryption approach compared with others. Homomorphic encryption is efficient when the number of candidates or choices is small. However, when the number of candidates or choices is large, computational and communicational cost for the proof and vote validation is quite high that homomorphic voting becomes less efficient.

Mix servers also suffer from computational cost for proving that their mixing is correct, in order to make the system trustworthy. All mix-net protocols and implementations need expensive operations and complex calculations. Moreover, mix-nets are not easy to set up and add substantial complexity to the protocol. For example, many mix servers are needed. The cast votes are forwarded via a sequence of mix servers. All incoming messages are rearranged before being forwarded to the next mix server and to the final destination. Depending on the number of mix servers and rearrangement computation, many encryption and decryption operations should be done. Furthermore in order to satisfy anonymity, the basic assumption is at least one mix server is trustable; otherwise some additional work should be done.

PVID scheme uses neither mix-nets nor homomorphic encryption in order to achieve privacy. It only employs blind signature and provides a practical way of assuring voter privacy in electronic voting protocols. The cost of blind signature operations is relatively small and inexpensive in terms of calculations and computations.

Mix-net based voting protocols support wide variety of voting types, even write-in ballots, where homomorphic encryption based voting protocols are just suitable for the selected voting types such as yes-no or 1-out-of-L voting. PVID scheme supports any voting types as mix-nets do.

In homomorphic encryption based voting protocols voting results are obtained easily so ballot tabulations are more efficient. Ballot tabulation in voting protocols which use PVID scheme is straightforward since counter directly collects cast votes with the associated PVIDs. Summary of all these comparisons and more is given in Table 5.1.

Table 5.1: Comparison of privacy preserving approaches.

	Using mix-nets	**Using homomorphic encryption**	**Using blind signature and assuming anonymous channels**	**Using blindly signed identities - PVID scheme**
Communicational complexity	High (Depends on the number of mix-servers)	High	Depends on the anonymous channel implementation	Low
Computational complexity	Low (High, when mix server verification is needed)	High	Depends on the anonymous channel implementation	Low
Scalability	Medium	Small	Large	Large
Practicality	No	No	Depends on the anonymous channel implementation	Yes
Supported voting types	Any type	Selected types (Yes-No, 1-out-of-L)	Any type	Any type
Allowing recasting	No	No	No	Yes
Achievability of individual verifiability	Yes	No	Yes	Yes
Tallying	Normal	Efficient	Normal	Normal
Need of vote validity proof	No	Yes	No	No

In homomorphic encryption and anonymous channel based voting protocols, eligibility and uniqueness requirements are handled before the voting process. In this case counter cannot have any idea about the voters' eligibility and cannot realise the double votes. This may cause serious problems in accuracy when authorities corrupt. On the other hand PVID scheme distributes the control of eligibility and uniqueness requirements between authorities. PVID authority provides mainly eligibility and uniqueness, moreover counter also checks double votes. Hence in voting protocols using PVID scheme, eligibility, uniqueness and as well as accuracy are achieved better than other protocols.

In homomorphic encryption based voting protocols encrypted votes are added, so no individual vote can be revealed. This effectively hides the contents of the original ballots,

but individual verifiability cannot be achieved. This is another disadvantage of these types of protocols.

PVID scheme is an alternative for mix-nets and homomorphic encryption. It slightly differs from the other blind signature based protocols, since it does not employ blind signature in a traditional way.

5.6 Prototype Implementation

As a proof of concept, a prototype has been developed that implements the entire PVID scheme. The main outcome of the implementation is that PVID scheme overcomes the limitations of traditional mix-nets. Implementation details are explained in Section 7.6.

CHAPTER 6

STRENGTHENED ACCURACY WITH DYNAMIC BALLOTS

This chapter presents dynamic ballots and explains their advantages.

6.1 Dynamic Ballot Mechanism

The existing electronic voting protocols generally use static ballot structure that is inherited from paper based voting. In these usual ballots, the order of candidates on ballot is pre-determined, so everyone, or at least the authorities, know the order of candidates; as the ballot is standard, a voter's casting hints at his actual vote. In dynamic ballots, however, the ordering of candidates changes randomly for each ballot. A voter's selection of a candidate has contextual meaning that shows his actual vote only with the corresponding dynamic ballot.

It is assumed that any ballot **B** contains *n* candidates: $\mathbf{B} = \{C_1, C_2, \ldots, C_n\}$, C_i representing a different candidate for each dynamically generated ballot. For *n* candidates, voters may take '*n!*' different ballots in ideal case. However, if n>9 (9! = 362880) then some optimisation should be done in dynamic ballot generation algorithm since generated dynamic ballot will be probably unique for each voter. Although this is not mandatory, it is highly encouraged to have a reasonable number of different ballots to make sure that different voters may take ballots on which the candidates are identically ordered. It is more efficient and truly random. Dynamic ballot generation algorithm is a permutation function which uses a PRNG.

Dynamic ballot mechanism is not a user interface implementation; it is a part of the protocol itself and employed in the protocol layer, not in the user interface layer. Chaum *et al.* [22] has mentioned randomness in ballots; however, they utilise randomisation of the candidate in order to provide VVPAT and equality of candidates.

An example set of dynamic ballots for four candidates can be as follows:

Table 6.1: A sample set of ballots.

B
$B_1 = \{C_2, C_1, C_4, C_3\}$
$B_2 = \{C_1, C_2, C_3, C_4\}$
$B_3 = \{C_4, C_1, C_3, C_2\}$
$B_4 = \{C_3, C_2, C_1, C_4\}$
$B_5 = \{C_2, C_1, C_4, C_3\}$

Now we can go one step further and define the *dynamic vote*. An actual vote **V** is the candidate selected from among the others shown on the ballot. Actual vote **V** directly shows the candidate whether there is a static ballot or dynamic ballot. We define a *dynamic vote* **V'** as the voter's selecting a candidate in the dynamic ballot. In other words, the dynamic vote has a contextual meaning depending on the ordering of candidates in the dynamic ballot **B**. Figure 6.1 demonstrates the dynamic ballot and its usage. In the figure, (a) and (b) represents different dynamic ballots, and although dynamic votes are the same, actual votes are different.

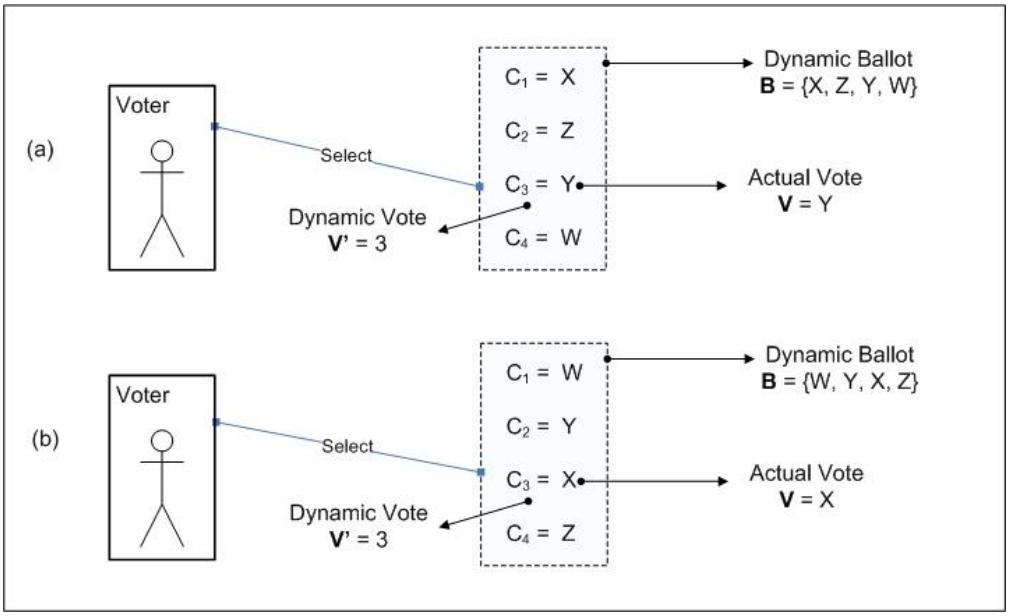

Figure 6.1: Dynamic ballots.

For example, the following dynamic votes may be chosen by voters for the given sample ballot set in Table 6.1:

Table 6.2: A sample set of dynamic votes.

B	**V'**
$B_1 = \{C_2, C_1, C_4, C_3\}$	$\mathbf{V_1'} = 2$
$B_2 = \{C_1, C_2, C_3, C_4\}$	$\mathbf{V_2'} = 2$
$B_3 = \{C_4, C_1, C_3, C_2\}$	$\mathbf{V_3'} = 3$
$B_4 = \{C_3, C_2, C_1, C_4\}$	$\mathbf{V_4'} = 3$
$B_5 = \{C_2, C_1, C_4, C_3\}$	$\mathbf{V_5'} = 3$

For the given sample ballot set in Table 6.1 and sample dynamic vote set in Table 6.2 the election result becomes as in Table 6.3. Then final tally becomes as: (C1, 2 votes), (C2, 1 vote), (C3, 1 vote), (C4, 1 vote).

Table 6.3: A sample election result.

B	**V'**	**V**
$B_1 = \{C_2, C_1, C_4, C_3\}$	$\mathbf{V_1'} = 2$	$V_1 = C_1$
$B_2 = \{C_1, C_2, C_3, C_4\}$	$\mathbf{V_2'} = 2$	$V_2 = C_2$
$B_3 = \{C_4, C_1, C_3, C_2\}$	$\mathbf{V_3'} = 3$	$V_3 = C_3$
$B_4 = \{C_3, C_2, C_1, C_4\}$	$\mathbf{V_4'} = 3$	$V_4 = C_1$
$B_5 = \{C_2, C_1, C_4, C_3\}$	$\mathbf{V_5'} = 3$	$V_5 = C_4$

6.2 Extension with Predefined Fake Votes (PreFotes)

The usage of dynamic ballots can be extended with PreFotes. In this case each candidate in dynamic ballot is associated with a unique CCode from the PreFote list. CCodes are chosen from the PreFote list in a random manner. A sample is given in Figure 6.2. Dynamic ballots can be implemented in different ways; dynamic ballot generation function and the layout of user interface are details of implementation. A sample ballot implementation can be as in Figure 6.3.

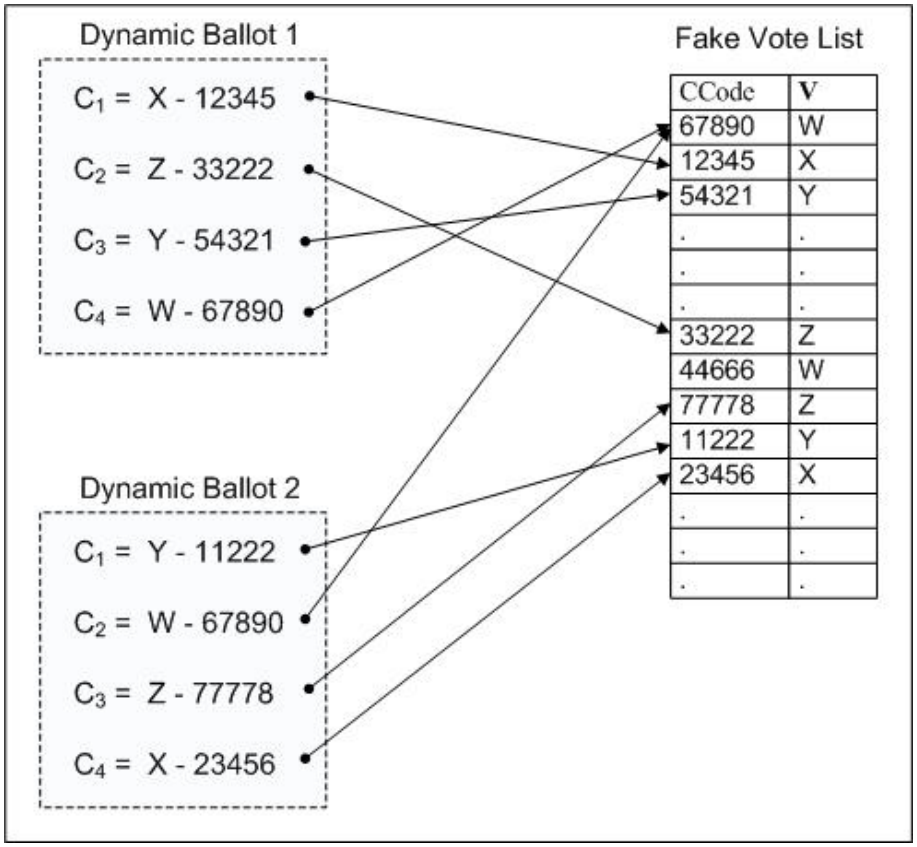

Figure 6.2: Extended dynamic ballots with PreFotes.

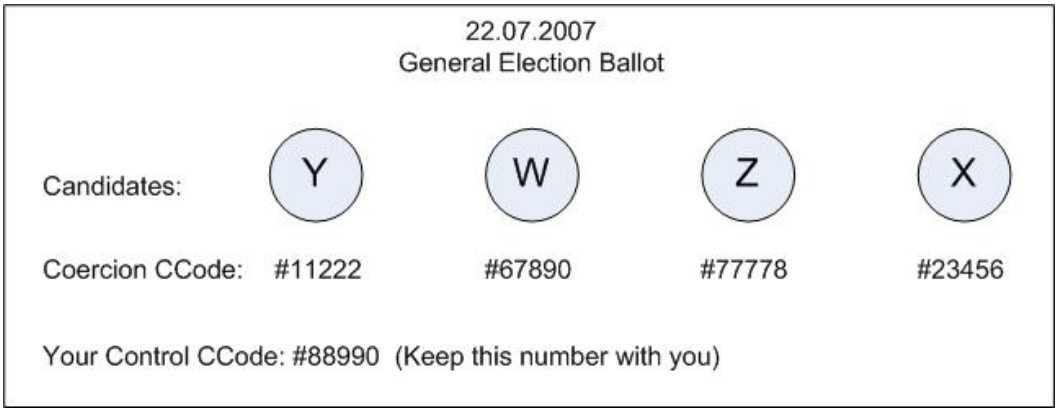

Figure 6.3: A sample dynamic ballot layout.

6.3 Dynamic Ballot Support on E-voting Requirements

Any voting protocol which uses dynamic ballots can fulfil some of the e-voting requirements, such as fairness, individual verifiability and accuracy, easily and with little effort.

Fairness (No partial tally is revealed before the end of the voting period.): If dynamic ballots are provided by a separate authority other than the counter and it does not share the generated ballots with the counter, then fairness can be achieved easily since Counter cannot count actual votes before the end of the election. Besides, in order to gain knowledge about the tally, any participant or authority should know both the dynamic ballot and the corresponding dynamic vote.

Receipt-freeness (Voters must neither be able to obtain nor construct a receipt which can prove the content of their vote to a third party): Dynamic votes can be used as receipts, or some other derived receipts can be defined on dynamic ballots.

Accuracy (The published tally should be correctly computed from correctly cast votes): Dynamic ballots increase accuracy. It is not possible to add fake votes without the dynamic ballot. No participants, voters or authorities can disrupt or influence the election and final tally by adding false votes or modifying the valid ones. Dynamic ballots prevent deliberate vote modification.

Individual Verifiability (The voter should be able to check that his encrypted vote was counted correctly in the final tally): As stated earlier, dynamic votes can be used as receipts, or some other derived receipts can be defined on dynamic ballots. These receipts do not directly reveal the actual vote, but they can provide voter to individually verify his vote.

CHAPTER 7

VOTER-VERIFIABLE AND RECEIPT-FREE VOTING PROTOCOL OVER A NETWORK

This chapter proposes a voter verifiable and receipt-free cryptographic voting protocol, namely *DynaVote*. DynaVote uses neither mix-nets nor homomorphic encryption since it successfully employs PVID scheme. The outline of this chapter is as follows. Firstly, the notation is given. Then DynaVote is proposed, and its stages are explained in detail. Prototype implementation of DynaVote is presented at the end of this chapter. The security analysis is performed in the next chapter.

7.1 Notation

Before explaining DynaVote in detail, the following notation is provided.

- DynaVote protocol actors:

  v: Voter

  a: PVID Authority

  b: Ballot Generator

  k: Key Generator

  c: Collector

  t: Counter

- Public-private key pairs:

  (β_a, δ_a): PVID Authority's public-private key pair.

  (β_b, δ_b): Ballot Generator's public-private key pair.

(β_k, δ_k): Key Generator's public-private key pair.

(β_c, δ_c): Collector's public-private key pair.

(β_v, δ_v): Voter's permanent public-private key pair used to communicate with PVID Authority.

(β_x, δ_x): Voter's session public-private key pair used to communicate with Ballot Generator.

(β_y, δ_y): Voter's session public-private key pair used to communicate with Key Generator and Collector.

(β_z, δ_z): Voting public-private key pair generated for Voter to cast his dynamic vote. The public one is called as *vote encryption key* in the rest of the thesis.

- Functions:

Ĕ$_x$(m): Encryption of message *m* with the public key of actor *x*.

Ď$_x$(m): Decryption of message *m* with the private key of actor *x*.

Š$_x$(m): Signing of message *m* with the private key of actor *x*.

Ŭ$_x$(m): Unsigning of message *m* with the public key of actor *x*.

H(m): One way cryptographic hash function of message *m* used by the voter and authorities.

- Elements:

B: Dynamic ballot.

Q: The number of dynamic ballot requests for a particular voter.

V': Dynamic vote, i.e. it is a voter's candidate selection depending on the dynamic ballot.

V: Voter's actual vote.

PVID-list: {$PVID_1$, $PVID_2$}, a list of approved anonymous pseudo identities which are unlinkable to the voter's real identity. PVID-list is obtained with PVID scheme.

7.2 DynaVote Overview

DynaVote protocol consists of three distinct stages: i) Authentication & Authorisation, ii) Voting, and iii) Counting. Authentication & authorisation are performed at the beginning of the election. Voting is carried out on the election day. Counting is performed at the end of the election. DynaVote allows remote Internet voting and poll-site Internet voting together.

In the authentication & authorisation stage, we employ PVID scheme. The voting stage consists of two phases: ballot obtaining phase and vote casting phase. In the ballot obtaining phase, Ballot Generator provides a dynamic ballot to the voter. In the meantime, Key Generator provides vote encryption key to the voter over Ballot Generator. In the vote casting phase, the voter selects his vote from the dynamic ballot and then encrypts his dynamic vote by using vote encryption key. Lastly, he casts his encrypted dynamic vote by using his PVID. In the counting stage, votes are decrypted and counted. Overview of DynaVote is shown in Figure 7.1.

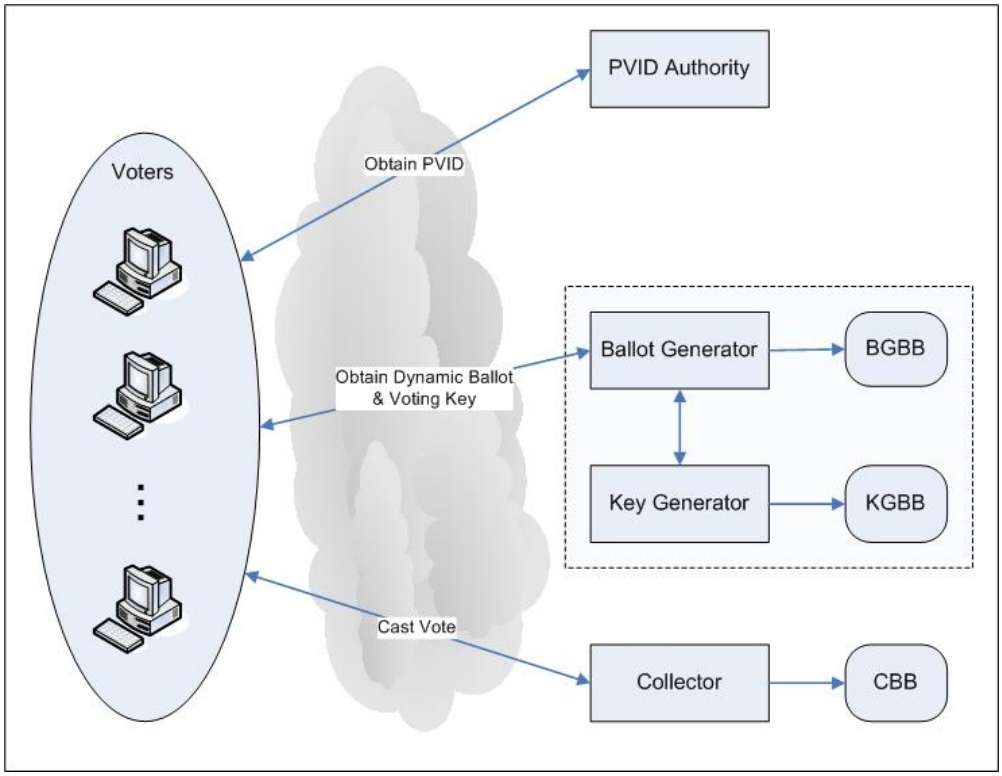

Figure 7.1: DynaVote overview.

Prior to authentication and authorisation, the registration stage takes place. List of all eligible voters, known as electoral roll, is created in the registration stage. Any eligible voter checks whether his identity is listed or not. The list of eligible voters is used by the PVID authority in the authentication and authorisation stage.

A PreFote list is prepared just before the election starts. Ballot Generator, Key Generator, Collector and Counter participate in the PreFote list generation process. PreFote list is simply a purposefully prepared list of fake votes. A PreFote consists of a unique CCode and an associated candidate from the candidates list. For each candidate, a constant threshold number of PreFotes are generated and they are listed in PreFote list.

The threshold value, *k*, should ideally be the number of voters participating in the election. However, in order to increase performance in the tallying phase, *k* can be a reasonable number up to a certain probability.

PreFotes do not affect the final tally and do not reduce accuracy since votes in the PreFote list are not counted in the final tally and are not published in the final tally. That is, the published tally only shows real votes. PreFotes are used for individual voter verifiability and are listed in the individual vote check list. Individual vote check list is not used as a tally list since it does not contain information about voting status for recasting. A voter also checks his vote and his PVIDs from the dynamic vote list in order to verify that his vote is really counted. In other words, a voter verifies his vote by using dynamic vote list and individual vote check list. As Key Generator checks both lists, Counter should publish individual votes check list consistent with the final tally.

In all stages bulletin boards are used in order to increase security and trustworthiness of the protocol. Authorities append information to their local bulletin boards in different steps of the protocol. In each stage, a voter can check and individually verify intermediate outcomes against bulletin boards. On suspicion of corruption, he can make an objection. All communications with the bulletin board are public and therefore can be monitored. Data already put on to a bulletin board cannot be altered or removed.

7.3 Authentication & Authorisation Stage

This stage is performed at the beginning of the election period. Each voter applies to the PVID authority to obtain a PVID-list by using his real RegID. PVID Authority checks the voter's eligibility by using the eligible voter list and issues the voter's PVID-list. RegID can be any widely used identity such as a national identity number, social security number. PVID-list is simply a list of approved anonymous pseudo identities which are unlinkable to voters' registration identities.

After completing this stage, the voter obtains a PVID-list and he can use PVIDs at any time and place throughout the election period. Voter's real registration identity is hidden to the voting authorities. Thus, any voter becomes anonymous while he is using the PVIDs in his communications with the voting authorities. Voting authorities can easily check the validity of any PVID by applying PVID Authority's public key on it. This stage is carried out is known as voter authentication & authorisation.

In PVID scheme, the voter performs blind signature with PVID Authority in order to obtain PVID-list. DynaVote employs PVID scheme for two identities. The voter creates an ID list $\{ID_1, ID_2\}$ where each ID contains the same random number as well as some

meaningful keywords such as *ID = (Election Data, Authority Data, Random Number)*. The voter blinds the IDs separately with different random blinding factors *r*, and obtains message M_b, which is the combination of blinded IDs.

Then, the voter sends $\check{\mathbf{E}}_a(\text{RegID}, \check{\mathbf{S}}_v(M_b))$ to PVID Authority. PVID Authority checks his eligibility. If the voter is eligible and has not made any request yet, the PVID Authority signs blinded IDs in message M_b and obtains M_{bs}. which is the combination of blindly signed IDs.

Then PVID Authority sends $\check{\mathbf{E}}_v(M_{bs})$ back to the voter. The voter checks PVID Authority's signature on M_{bs} and then unblinds each blindly signed ID in message M_{bs} and obtains PVID-list = $\{PVID_1, PVID_2\}$. $PVID_1$ and $PVID_2$ are anonymous pseudo identities signed by the PVID authority. Anyone can verify the signature on them and check whether they belong to the same voter.

PVID-list cannot be re-used in the DynaVote protocol. After the election ends, PVIDs are not valid anymore. Voters need to get new PVIDs for a new election. It is not a difficult task to obtain PVIDs anyway.

In order to satisfy IP untraceability, DynaVote accepts all of the cases that PVID scheme suggests, and it allows both remote Internet voting and poll-site Internet voting. DynaVote assumes that authorities do not cooperate with the intension of revealing the voter's IP address and that PVID authority has no communication with other authorities. The details are explained in Chapter 8.

7.4 Voting Stage

In the voting stage, each voter obtains a *dynamic ballot* and casts his dynamic vote. Dynamic ballot mechanism is the main building block of this stage, and it is explained in the previous chapter. This stage consists of two phases: Ballot obtaining phase and vote casting phase. Overview of voting stage is shown in Figure 7.2. Details of these phases are explained in the following sections.

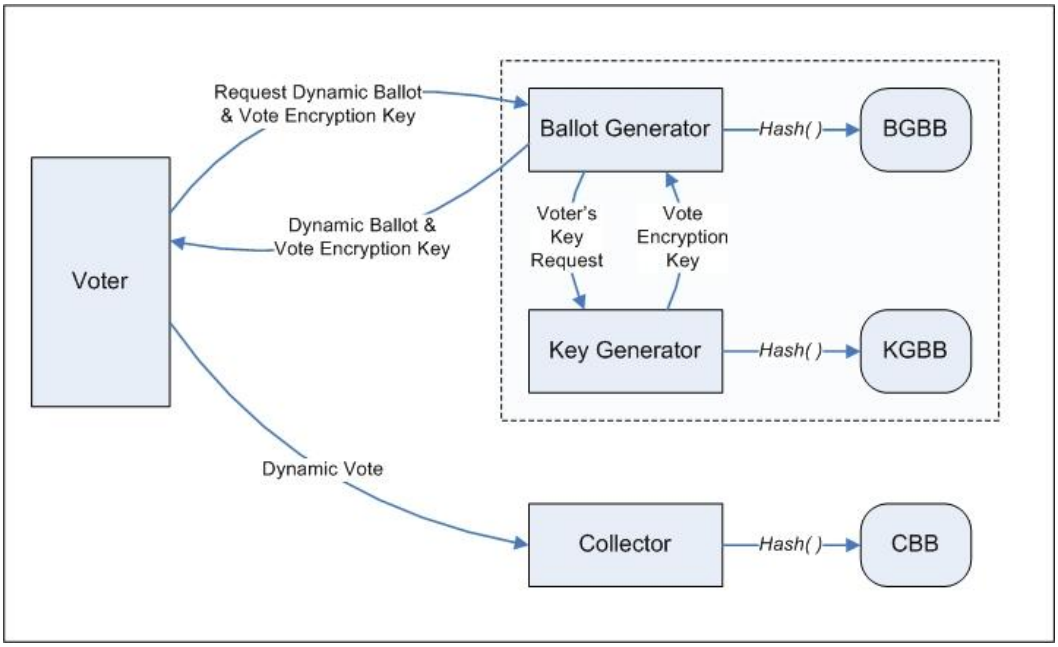

Figure 7.2: Overview of the voting stage.

7.4.1 Ballot Obtaining Phase

The voter creates session public-private key pairs (β_x, δ_x) and (β_y, δ_y). The former is used for Ballot Generator; the latter is used for Key Generator. He employs these keys in order to obtain a dynamic ballot and vote encryption key. Voter encrypts β_y and election data with Key Generator's public key and produces $\check{\mathbf{E}}_k(\beta_y, \text{ElectionData})$. Election data is used to make the message more meaningful for Key Generator and to make it more easily identified by Key Generator. Then, voter creates the message M_1:

$$M_1 = \check{\mathbf{E}}_b(\text{PVID}_1, \check{\mathbf{E}}_k(\beta_y, \text{ElectionData}), \beta_x)$$

The voter sends M_1 to Ballot Generator. As soon as receiving the message M_1, Ballot Generator decrypts it. Ballot Generator checks the PVID_1 by applying PVID Authority's public key. If the check fails, Ballot Generator discards the message. If it succeeds, Ballot Generator signs $\check{\mathbf{E}}_k(\beta_y, \text{ElectionData})$ and then generates the message M_2:

$$M_2 = \check{\mathbf{E}}_k(\check{\mathbf{S}}_b(\check{\mathbf{E}}_k(\beta_y, \text{ElectionData}), \text{ControlData}))$$

A ControlData is also encrypted inside the message body in order to identify any message corruption. ControlData can be any predefined value. Ballot Generator sends the message M_2 to Key Generator.

Key Generator decrypts the message M_2 and checks Ballot Generator's signature on it. If it is a valid message, Key Generator proceeds to further steps. Key Generator creates a voting key pair (β_z, δ_z) and a unique CCode. Voting public key (i.e. vote encryption key) is used by the voter to cast his dynamic vote to Collector. Key Generator saves generated key pair (β_y, β_z, δ_z, CCode) in VotingKeyList, which is an internal list of voting key pairs. It publishes hash values of the voter's public key with voting key's public one and private one separately as $\mathbf{H}(\beta_y, \beta_z)$ and $\mathbf{H}(\beta_y, \delta_z, \text{CCode})$ in Key Generator's Bulletin Board (KGBB). $\mathbf{H}(\beta_y, \beta_z)$ is used by the voter to verify the correctness of the vote encryption key. It is also controlled by all voting authorities and passive observers at the counting stage. $\mathbf{H}(\beta_y, \beta_z)$ and $\mathbf{H}(\beta_y, \delta_z, \text{CCode})$ are used by Counter just before starting the counting stage to prevent Key Generator's influence on the generated voting key pairs. Key Generator generates M_3 and M_4:

$M_3 = \check{\mathbf{E}}_y(\check{\mathbf{S}}_k(\beta_z, \text{CCode}, \text{ElectionData}), \text{ControlData})$

$M_4 = \check{\mathbf{E}}_b(\check{\mathbf{S}}_k(M_3, \text{ControlData}))$

Key Generator sends M_4 to Ballot Generator. Ballot Generator decrypts the message and checks Key Generator's signature. Afterwards Ballot Generator creates a dynamic ballot **B** by using a ballot generation algorithm relying on a random number generator function. Dynamic Ballot **B** orders candidates randomly, and each candidate in **B** is associated with a CCode from the PreFote list. CCodes are chosen from the PreFote list in a random manner. This feature enables the voter to learn and keep a set of PreFotes in order to prevent uncoercibility. He can select any number of CCodes he wants to take with him in case of coercion and he also withholds his real vote's CCode to perform individual verifiability. The way CCodes are explained is an implementation detail. Depending on the implementation, the system can print all CCodes or selected ones. However, the process of printing requires extra hardware and maintenance cost. All responsibilities can preferably be given to the voter. The voter can give any fake CCode to a coercer. The coercer cannot observe the difference between the fake CCodes and real CCodes since fake CCodes are also published in individual vote check list. Hence, this feature makes uncoercibility useless in practice.

Ballot Generator next saves the ($PVID_1$, M_3, **B**, **Q**, β_x) in BallotList, which is an internal list of dynamic ballots. **Q** is the number of dynamic ballot requests for $PVID_1$, and Ballot generator calculates it by counting the previous attempts according to $PVID_1$ saved in BallotList. The request number **Q** is used in order to handle recasting, and naturally its initial value is 1. If a voter recasts, then it becomes 2 and so on.

Later it publishes the hash of dynamic ballot **B**, the request number **Q** and voter's session public key β_x which is **H**(**B**, **Q**, β_x) and **H**(**B**, **Q**, $PVID_1$) in Ballot Generator's Bulletin Board (BGBB). **H**(**B**, **Q**, β_x) is published so that the voter is empowered to verify the correctness of dynamic ballot. It is also controlled by all voting authorities and passive observers at the counting stage. **H**(**B**, **Q**, β_x) and **H**(**B**, **Q**, $PVID_1$) are controlled by Counter just before starting the counting stage. Key Generator controls both of them after the counting stage is completed. At this step Ballot Generator produces M_5:

$M_5 = \check{\mathbf{E}}_x(\check{\mathbf{S}}_b(M_3, \mathbf{B}, \mathbf{Q}, \text{ControlData}))$

Ballot Generator sends M_5 to the voter. He decrypts the received message by applying Ballot Generator's public key and extracts M_3, dynamic ballot **B** and the request number **Q**. In order to verify the obtained dynamic ballot, the voter calculates **H**(**B**, **Q**, β_x) and **H**(**B**, **Q**, $PVID_1$) and then checks them against the BGBB.

Later, the voter decrypts the message M_3 and applies Key Generator's public key in order to extract vote encryption key β_z and CCode. He creates **H**(β_y, β_z) and verifies the result against the KGBB. At this point, the voter has dynamic ballot **B** and vote encryption key β_z; and he is ready to carry out vote casting. He will use CCode to individually verify his vote at the end of the election. Ballot obtaining phase is illustrated in Figure 7.3.

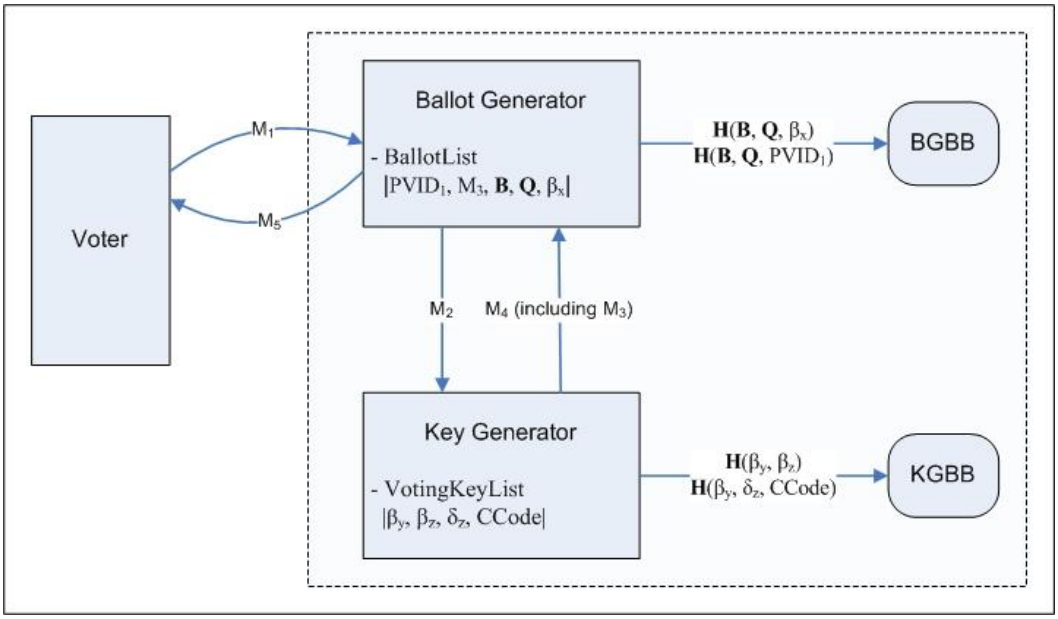

Figure 7.3: Ballot obtaining phase.

7.4.2 Vote Casting Phase

The voter selects his candidate and creates his dynamic vote **V'** using the dynamic ballot **B**. He encrypts **V'** with vote encryption key β_z. Then, he constructs his encrypted dynamic vote (**encV'**) and creates the message M_6:

$$\mathbf{encV'} = (\check{\mathbf{E}}_z(\mathbf{V'}, \mathbf{Q}, PVID_2, CCode), \check{\mathbf{E}}_z(\mathbf{V'}, \mathbf{Q}, \beta_x))$$

$$M_6 = \check{\mathbf{E}}_c(PVID_1, \beta_y, \mathbf{encV'})$$

The voter sends M_6 to Collector; in other words, the voter casts his vote. Nobody can relate between voter's real registration identity to PVIDs due to the essence of PVID scheme. Hence, the voter can easily send the encrypted dynamic vote **encV'** as well as PVIDs.

Collector decrypts the message M_6 and extracts $PVID_1$ as well as encrypted dynamic vote **encV'**. Collector performs PVID Authority's public key on $PVID_1$ to check the validity of $PVID_1$. If it is valid, Collector processes the request; if not, Collector discards the message. Collector saves encrypted **V'** (**enc V'**) by appending the date and time of it to VoteList as ($PVID_1$, β_y, $\check{\mathbf{E}}_z$(**V'**, **Q**, $PVID_2$, CCode), $\check{\mathbf{E}}_z$(**V'**, **Q**, β_x), DateTime). VoteList

is an internal list of voters' dynamic votes associated with PVIDs. Collector creates the hash $\mathbf{H}(\check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, \text{PVID}_2, \text{CCode}))$ and hash $\mathbf{H}(\check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, \beta_x))$; to later publish them with the associated DateTime in Collector's Bulletin Board (CBB). In turn, Collector sends an acknowledgement message $\check{\mathbf{E}}_y(\check{\mathbf{S}}_c(\mathbf{Ack}))$ to the voter in order to inform him. As soon as receiving the **Ack**, the voter checks the CBB to verify his vote individually. Then the voter's voting session is over. Vote casting phase is shown in Figure 7.4.

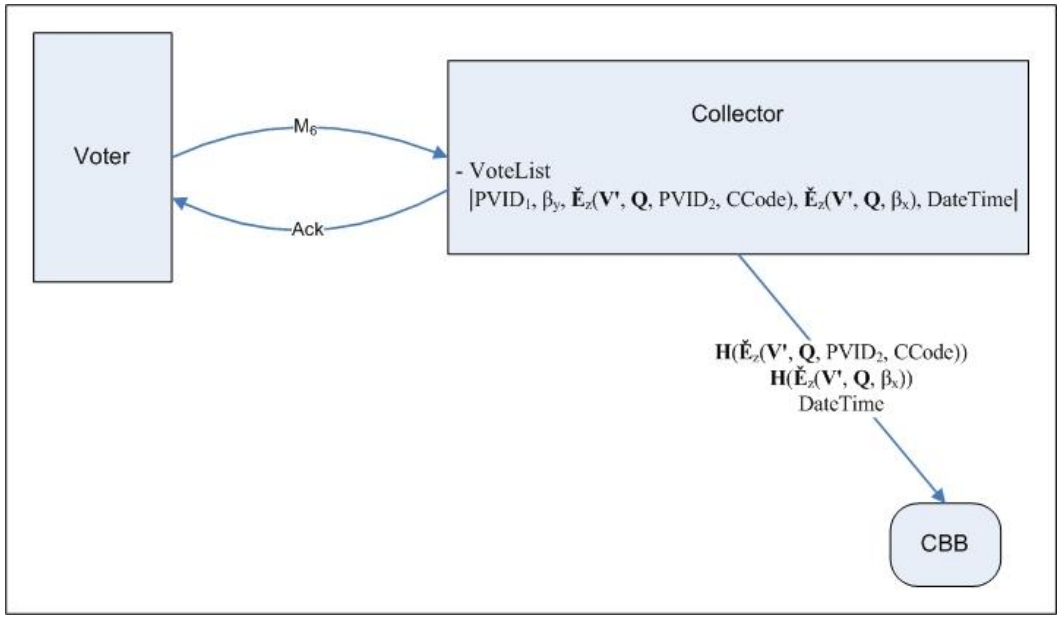

Figure 7.4: Vote casting phase.

Voter controls $\mathbf{H}(\check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, \text{PVID}_2, \text{CCode}))$ and $\mathbf{H}(\check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, \beta_x))$ against CBB in the vote casting phase. $\mathbf{H}(\check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, \beta_x))$ hash value and DateTime are also controlled by all voting authorities and passive observers at the counting stage. Both hash values and DateTime are controlled by Counter just before starting the counting stage. Key Generator controls consistency of the published values after the counting stage is completed. Announcing DateTime does not allow timing attacks due to the fact that Counter announces DateTime only at this phase. DateTime is not announced with final tally.

7.5 Counting Stage

Counting stage is performed after the election period has been completed. During the election period, Ballot Generator, Key Generator and Collector publish hash of subsets of relevant information on bulletin boards. Before proceeding the counting of votes, Ballot Generator announces the generated ballot list (|**B**, **Q**, β_x|); Key Generator announces the generated vote encryption key list (|β_y, β_z|); and Collector announces the encrypted dynamic votes (|$\check{\mathbf{E}}_z$(**V'**, **Q**, β_x), DateTime|). Later Ballot Generator, Key Generator and Collector send to Counter the BallotList (|$PVID_1$, M_3, **B**, **Q**, β_x|), VotingKeyList (|β_y, β_z, δ_z, CCode|) and VoteList (|$PVID_1$, β_y, $\check{\mathbf{E}}_z$(**V'**, **Q**, $PVID_2$, CCode), $\check{\mathbf{E}}_z$(**V'**, **Q**, β_x), DateTime|), respectively. Announced lists are illustrated in Figure 7.5.

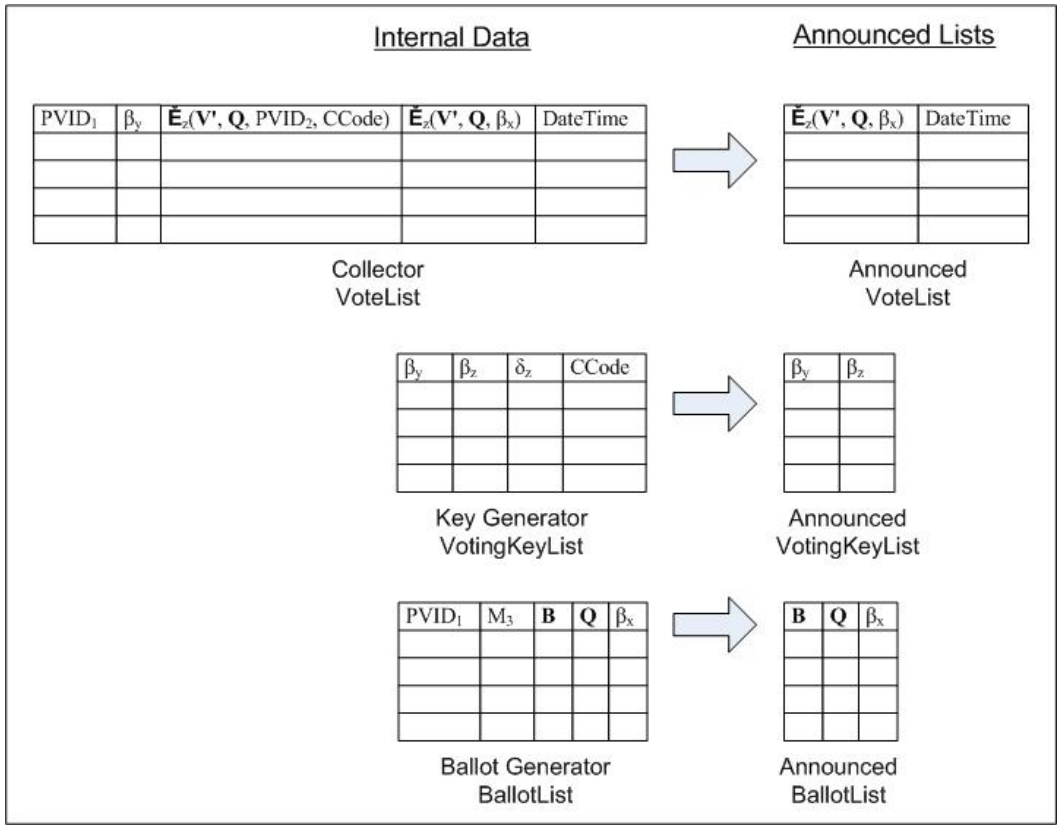

Figure 7.5: Announced authority data.

Counter compares the VoteList, BallotList, VotingKeyList and announced lists for consistency and checks against the hash values in the bulletin boards. Any passive observer or organisation can also check the consistency of the election by using the announced lists and bulletin boards.

Then Counter starts counting. Firstly, it matches each item in VoteList $|PVID_1, \beta_y, \check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, PVID_2, CCode), \check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, \beta_x), DateTime|$ with corresponding items in VotingKeyList $|\beta_y, \delta_z, CCode|$ over voter's session key β_y. Afterwards Counter obtains a list $|PVID_1, \beta_y, \check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, PVID_2, CCode), \check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, \beta_x), \beta_z, \delta_z, CCode, DateTime|$.

Counter processes the list by decrypting the encrypted dynamic votes with the corresponding private keys (δ_z) and produces the list $|PVID_1, PVID_2, \beta_y, \beta_x, \beta_z, \delta_z, \mathbf{V}', \mathbf{Q}, \check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, PVID_2, CCode), \check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, \beta_x), CCode, DateTime|$ which is a list of voters' dynamic votes. The process of calculating the dynamic vote list is shown in Figure 7.6.

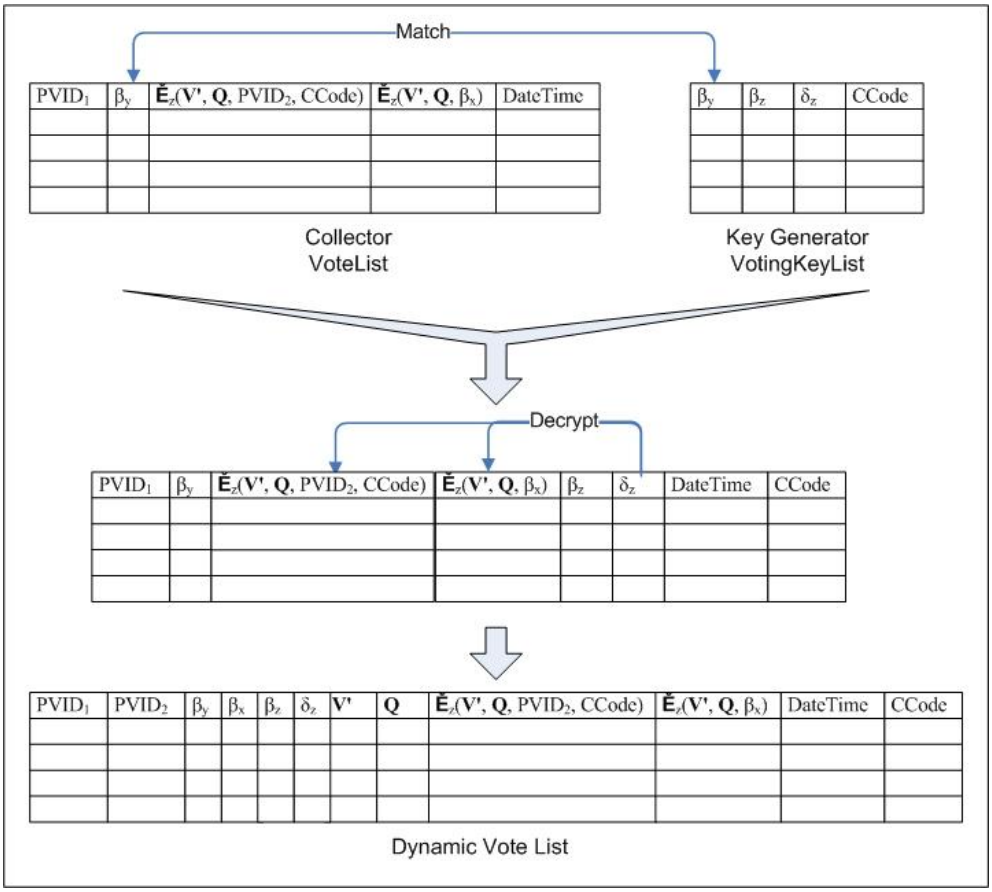

Figure 7.6: Dynamic vote list.

Before revealing the actual votes, Counter performs some checks on dynamic vote list and determines if the vote is valid or discarded. These checks are listed below:

- Counter checks the $PVID_1$ and $PVID_2$ by applying PVID Authority's public key. If the check fails, Counter discards the vote.

- Counter checks the relation between $PVID_1$ and $PVID_2$. They must belong to same voter, or else the vote is discarded.

- Since PVID scheme is employed and the DynaVote allows recasting, voter can vote several times. Collector keeps track of date and time of each casting, and Ballot Generator associates a number **Q** to each ballot **B**. Consequently, Counter checks for recasts for any ($PVID_1$, $PVID_2$) in dynamic vote list. Only the latest cast is taken into consideration and previous casts are discarded.

Subsequently Counter announces the dynamic vote list with status information, indicating whether the votes are valid or discarded. However, this does not contain DateTime information like |$PVID_1$, $PVID_2$, **V'**, status| as shown in Figure 7.7. The votes can be of valid or discarded status. If any vote is discarded, the reason for it is also given such as in "Invalid $PVID_1$", "Recasting", etc. Published dynamic vote list enables voters to individually check their dynamic votes. This list also prevents Counter from discarding valid votes as actual votes are not revealed yet and voter can verify the published dynamic votes. **Q** is not announced by intentionally to support uncoercibility.

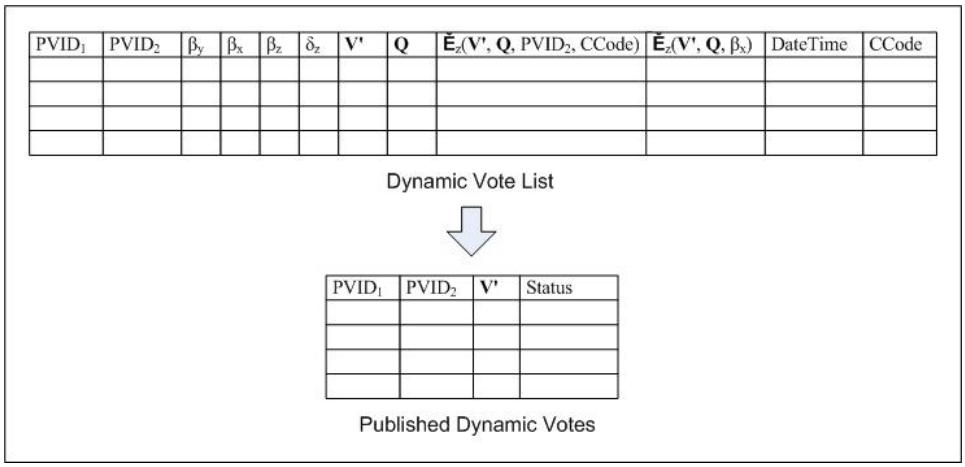

Figure 7.7: Published dynamic votes.

Later, Counter matches the dynamic votes (**V'**) in dynamic vote list with corresponding dynamic ballots (**B**) in BallotList over $PVID_1$, **Q** and β_x. If any recasting occurs for $PVID_1$, then they are matched according to their request number **Q**, and the associated dynamic ballot is found.

Then, Counter obtains a list |$PVID_1$, $PVID_2$, β_y, β_x, β_z, δ_z, **V'**, **Q**, **B**, **V**, $\check{\mathbf{E}}_z$(**V'**, **Q**, $PVID_2$, CCode), $\check{\mathbf{E}}_z$(**V'**, **Q**, β_x), DateTime, Status, CCode| which is in fact the list of voters' actual votes. Calculating the actual vote list is shown in Figure 7.8. An actual vote **V** is defined as:

$\mathbf{V} = C_i \in \mathbf{B}$ where i = **V'** and $\mathbf{B} = \{C_1, C_2, \ldots, C_n\}$

Counter generates $\check{\mathbf{E}}_z(PVID_1, PVID_2, \beta_x)$ by using β_z in order to provide the data necessary for Key Generator to verify the published tally. Key Generator does not know β_x, but it can obtain β_x by decrypting $\check{\mathbf{E}}_z(PVID_1, PVID_2, \beta_x)$.

At the end of the counting stage, Counter announces the list of |β_z, **V'**, **Q**, **B**, **V**, Status, $\check{\mathbf{E}}_z(PVID_1, PVID_2, \beta_x)$|. Now votes are easily tallied, and the election result is announced. Key Generator verifies the published result list.

The published tally and individual vote check list do not give a receipt to the voter; however, the lists provide voters to individually verify their votes. Extending dynamic ballots with predefined fake votes provides individual verifiability without sacrificing receipt-freeness and uncoercibility. Next chapter analyses DynaVote in detail and explains how e-voting security requirements are satisfied in DynaVote.

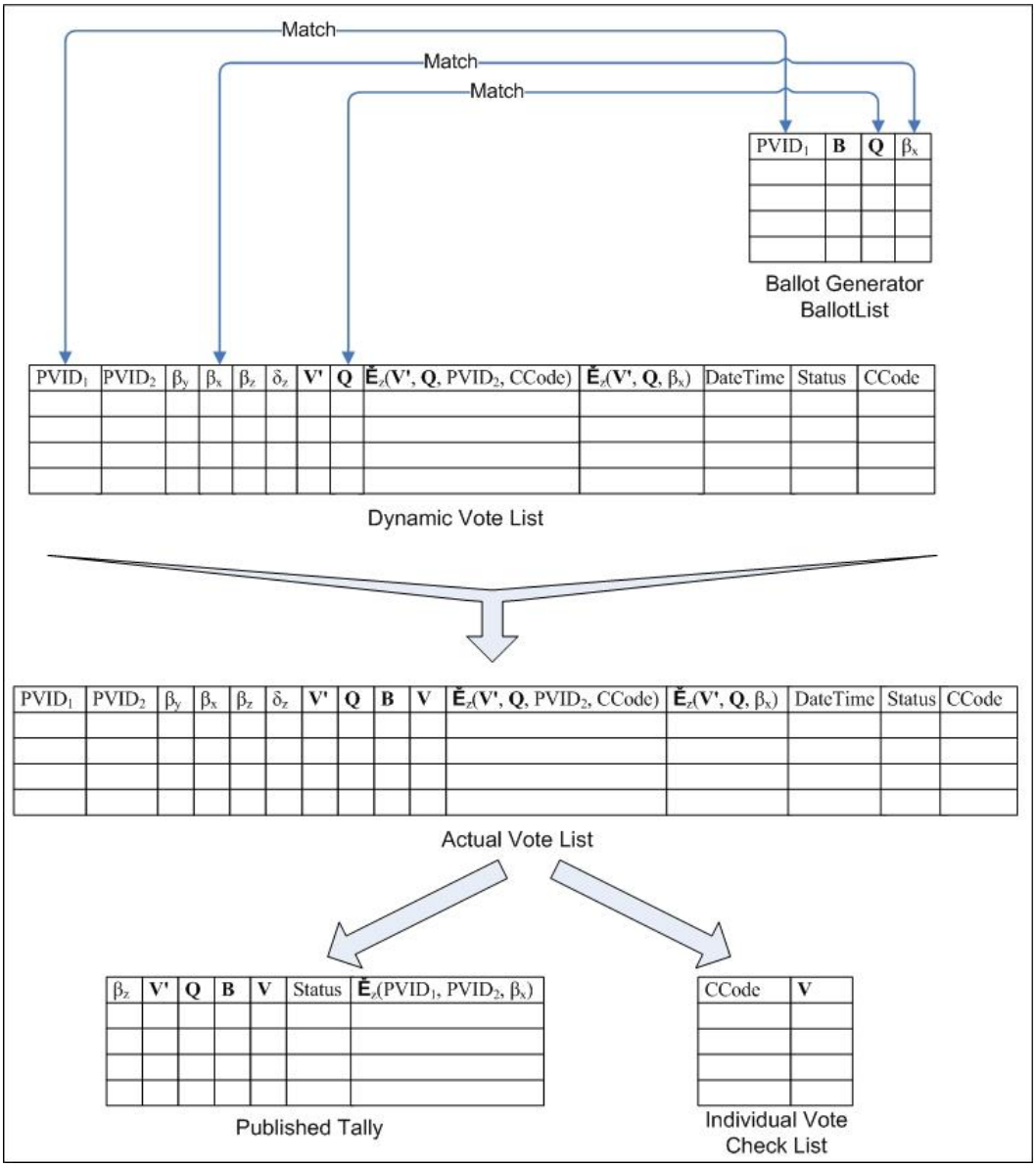

Figure 7.8: Actual vote list.

7.6 Prototype Implementation

As a proof of concept, a prototype has been developed that implements the entire DynaVote protocol over Internet. Dynamic ballot mechanism is implemented without PreFote extension. PVID scheme prototype is also implemented separately. However, they use the same programming infrastructure. This section explains the implementation of both prototypes. In its current state, the prototypes mainly serve experimental

purposes of testing the DynaVote protocol and PVID scheme. This study shows that DynaVote protocol over Internet is practical and applicable for large scale elections. As well as proving these strengths, it illustrates that PVID scheme provides unlinkability.

The prototype simulates a typical voting process. The basic scenario of the protocol over Internet is as follows: i) Voter obtains two PVIDs by using PVID application web page on the Election web site. ii) He accesses to the Voting web page on the Election web site by using $PVID_1$. He chooses his favourite candidate from the ballot list provided by Ballot Generator and casts his vote by using $PVID_2$. iii) When the election times out, Counter application is used to count the votes and to announce the election result. In order to implement this scenario, we have developed a client/server web application with Java. Voters represent the client side and authorities represent the server side. Servers are designed as Java applications and clients are designed as Java applets embedded in HTML files. Java applets are executed in a sandbox by web browsers, preventing them from accessing to local file system.

The voter should provide his private key while establishing a connection with PVID Authority server. Furthermore, on the voting stage he provides his PVIDs. So as to maintain the implementation user friendly, we should not force the voter to memorise his public-private key pair and the PVIDs. Thus, prototype allows the voters to save and load those data into files stored in flash disks.

Due to the fact that a Java applet is executed in a sandbox, we used a signed applet to be able to access to the local file system. This is a facility that the current web browsers allow so that an applet's execution space can go beyond the sandbox. When a signed applet arrives on the user's system, the user is notified of the identity of the applet's signer and of the capabilities that the applet requests. Then, the user can give permission only for the required capabilities. Another way to allow the voter's applet to access local data could be to define the policy for the applet. However, this is not appropriate in client-server applications due to the fact that the policy file should be defined for all of the clients.

We have used JDK 1.6 [58] for software development. Therefore, the system can be installed and executed on any computational platform with Java Virtual Machine (JVM). For the cryptographic functions, Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE) frameworks are used. In our implementation Sun JCA,

which includes JCE, is used since it's available with JDK 1.6. JCA consists of set of packages and provides several cryptographic services. In this architecture, a variety of cryptographic algorithms are supported. We have used RSA asymmetric key algorithm, DES symmetric key algorithm, SHA1 PRNG, and SHA-256 hash function. For database operations, we have utilised sql package of Sun. In addition to those packages, we have also used java.math.BigInteger and java.math.SecureRandom classes.

A voter connects to servers on a TCP socket. We used Sockets, ServerSockets, InputStream and OutputStream classes for communication between client and servers. We utilised the multi-thread support of Java in order to allow voters to connect simultaneously. For each request, servers create a thread and different voters may concurrently access the server.

We have used MySQL 5.0 database [76] to store the election data. MySQL provides an opportunity to export and import data. This opportunity is essential to transfer data between authorities since online data transfer between authorities is not preferable. There are five databases in DynaVote prototype. BallotGenerator, KeyGenerator, Collector and Counter databases are used to store server data. BulletinBoards database is used to implement bulletin boards and it is read-only accessible by all authorities and voters. As well as this, each authority can write only on its own bulletin board table in BulletinBoards database. The writing operation is disabled for unauthorised users.

Prototypes have been successfully tested with JRE 1.6 version on Windows XP with Internet Explorer. The implementation details about core functions and brief information on prototype usage are presented in Appendix B.

CHAPTER 8

ANALYSIS AND DISCUSSION

In the previous chapter, DynaVote protocol is explained in detail. This chapter provides an analysis of DynaVote. Firstly, a method of analysing the voting systems is suggested. Then, how DynaVote satisfies electronic voting security requirements is illustrated. At the end of this chapter some customisations on DynaVote are explained, and a comparison with the existing e-voting protocols is made.

8.1 A Method to Analyse Voting Systems

While electronic voting has been studied for the past two decades, research on analyzing voting systems has begun recently [93]. In this section, a method to analyse voting systems with respect to e-voting security requirements is proposed. This method helps to evaluate, as well as compare, the voting protocols and it is not protocol specific. In order to define a voting protocol *VP*, let:

$E = \{e_1, e_2, e_3 \ldots e_q\}$ be the set of all eligible voters where q is the number of eligible voters;

$A = \{a_1, a_2, a_3 \ldots a_n\}$ be the set of voters that performed a voting process where a_i is any voter and n is the number of voting attempts;

$B = \{b_1, b_2, b_3 \ldots b_n\}$ be the set of votes where b_i is the vote of voter a_i;

$D = \{d_1, d_2, d_3 \ldots d_n\}$ be the set of transactions in voting processes where d_i denotes all transactions of voter a_i during the voting process;

$V = \{v_1, v_2, v_3 \ldots v_m\}$ be the set of all valid votes (including all data) where m is the number of valid votes, $V \subseteq B$ and $m \leq n$;

$W = \{w_1, w_2, w_3 \ldots w_m\}$ be the set of published data at the end of the election, w_i denotes the published data for each valid vote v_i and $w_i \subseteq v_i$;

$C = \{c_1, c_2, c_3 \ldots c_k\}$ be the set of all candidates;

$f_{bv}: B \rightarrow V, f_{bv}(b_i) = v_j$ matches each b_i to a v_j if b_i is a valid vote;

$f_{ae}: A \rightarrow E, f_{ae}(a_i) = e_j$ matches each a_i to an e_j if a_i is an eligible voter;

$f_{vc}: V \rightarrow C, f_{vc}(v_i) = c_j$ matches each valid vote to an actual candidate;

$S = \{s_1, s_2, s_3 \ldots s_h\}$ be the set of all eavesdroppers;

$$T = \{(c_1, \sum_{i=1}^{m} add(f_{vc}(v_i), c_1)), (c_2, \sum_{i=1}^{m} add(f_{vc}(v_i), c_2)) \ldots (c_k, \sum_{i=1}^{m} add(f_{vc}(v_i), c_k))\}$$

be the tally.

Note that if any recasting occurs then it is handled as a new voting process, so it can be $n \geq q$. If recasting is not allowed, then it should be $n \leq q$. Besides, D does not require to be hidden.

8.1.1 Formal Definitions of E-voting Security Requirements

Lemma 1 Privacy (Voter-Vote relationship cannot be revealed): If $\forall d \in D \forall v \in V \forall e \in E[\neg(\exists f(S, W, d, v) = e)]$ for a voting protocol *VP*, then *VP* satisfies privacy.

Lemma 2 Eligibility (Each vote counted in the tally should be cast by an eligible voter): Let $f: V \rightarrow B, f(v_i) = b_j$ and $g: B \rightarrow A, g(b_j) = a_j$. If $\forall v \in V[f_{ae}(g(f(v))) \in E]$ for a voting protocol *VP*, then *VP* satisfies eligibility.

Lemma 3 Uniqueness (There should be at most one valid vote for each eligible voter in the final tally): Let $f: V \rightarrow B, f(v_i) = b_j$ and $g: B \rightarrow A, g(b_j) = a_j$. If $\forall v_i \in V \forall v_j \in V[f_{ae}(g(f(v_i)) = f_{ae}(g(f(v_j)) \leftrightarrow i = j]$ for a voting protocol *VP*, then *VP* satisfies uniqueness.

Lemma 4 Fairness (During the election none of the votes can be matched to an actual candidate): During the election, if $\forall b \in B[\neg(\exists c \in Cf(D,S,b) = c))]$ for a voting protocol *VP*, then *VP* satisfies fairness.

Lemma 5 Uncoercibility (No coercer can figure out a voter's vote by forcing him): If $\forall s \in S \forall a \in A \forall v \in A[\neg(\exists f(D,W,s,a) = v)]$ for a voting protocol *VP*, then *VP* is uncoercible.

Lemma 6 Receipt-freeness (Voters cannot prove their votes): If $\forall a \in A \forall v \in V[\neg(\exists f(D,W,a) = v)]$ for a voting protocol *VP*, then *VP* is receipt-free.

Lemma 7 Accuracy (Each vote cast by an eligible voter should be counted correctly in the final tally, and any fraud should be detected): Let $h: E \rightarrow A$, $h(e_i) = a_j$; $g:A \rightarrow B$, $g(a_j) = b_j$; $f:V \rightarrow B$, $f(v_i) = b_j$ and $g':B \rightarrow A$, $g(b_j) = a_j$. If $\forall e \in E[f_{bv}(g(h(e))) \in V] \wedge \forall v \in V[f_{ae}(g'(f(v))) \in E]$ for a voting protocol *VP*, then *VP* satisfies accuracy.

Lemma 8 Individual Verifiability (Each eligible voter should be able to verify his vote by using the published data): If $\forall e \in E \forall w \in W \exists! v \in V[\exists f(e,w) = v)]$ for a voting protocol *VP*, then *VP* satisfies individual verifiability.

Theorem 1: A voting protocol *VP* is a complete and secure protocol if and only if it satisfies *Lemma 1-8*.

Analysis of DynaVote is provided in Section 8.2.

8.1.2 Specific Cases of Security Requirements

This section provides a guideline to evaluating of the voting systems by explaining the specific cases of the security requirements and offers a systematic approach to analyse them. For each requirement, checklist items are given below, and which should be satisfied by cryptographic voting protocols. The given cases being already summarised from the definitions provided in Chapter 3, this section do not repeat them, but explain some specific attacks Table 8.1 illustrates the cases related to privacy, eligibility, uniqueness and fairness.

Table 8.1: Checklist for privacy, eligibility, uniqueness and fairness.

Main requirement	Requirement details	Satisfied (√)	Not satisfied (X)	Not Applicable (N/A)	How it is satisfied?	Assumptions
Privacy	Voter-Vote unlinkability					
	Voter-Vote IP untraceability					
	Voters cannot add identifiable information					
	Authorities cannot add identifiable information					
Eligibility	Eligible voters can vote					
	Ineligible voters cannot vote					
	Authorities cannot give voting credentials to ineligible voters					
	Authorities cannot usurp suffrage (voting right)					
Uniqueness	At most one valid vote is counted for each eligible voter					
	Each eligible voter has voted only once					
Fairness	Result is not published till the end of the election					
	Counting comes after the voting stage					
	No one can guess the content of any cast vote					
	No one can gain any partial knowledge about the tally before the counting stage					
	Encrypted votes are used and they are decrypted at the end of the election					

Table 8.2 illustrates the cases related to uncoercibility and receipt-freeness. In the randomisation attack, the attacker coerces a voter to submit a randomly generated vote [94]. The aim of the attack is to nullify the choice of the voter with a large probability. The forced-abstention attack is related to the previous one based on randomisation. In this case, the attacker coerces a voter by demanding that he abstains from voting. Most of the existing protocols are vulnerable to this attack. This is because the schemes authenticate voters directly. Thus, an attacker can see who has voted and use this information to coerce voters.

Table 8.2: Checklist for uncoercibility and receipt-freeness.

Main requirement	Requirement details	Satisfied (√)	Not satisfied (X)	Not Applicable (N/A)	How it is satisfied?	Assumptions
Uncoercibility	Nobody can force voter to vote in a particular way					
	Nobody can force voter physically being next to him					
	Coercer cannot receive any proof from the voter after voting					
	Coercer cannot force the voter to use a particular proof provided before voting					
	Coercer cannot vote instead of voter with his personal ID					
	Forced abstention attack is prevented					
	Randomization attack is prevented					
Receipt-freeness	Voter is not identifiable from the receipt					
	Vote is not revealed from the receipt					
	Voter cannot prove his vote					
	Vote selling/buying is prevented					
	Authority gives correct receipt					
	Any public data do not give any information about voter's vote					
	Voter cannot use a particular proof defined before voting					
	Voter cannot prove his vote even if he records his activity					
	Voter cannot obtain a particular proof after voting					
	Voter cannot use a personal ID such as RegID or private keys to prove his vote					

Table 8.3 and Table 8.4 illustrate the cases related to accuracy and individual verifiability respectively.

Table 8.3: Checklist for accuracy.

Main requirement	Requirement details	Satisfied (√)	Not satisfied (X)	Not Applicable (N/A)	How it is satisfied?	Assumptions
Accuracy	Ballot representation is correct					
	Authorities response correctly					
	Voter can vote as intended					
	Vote is recorded correctly					
	All valid votes are counted correctly					
	No valid votes are deleted					
	No valid votes are modified					
	No valid votes are spoiled					
	No valid votes are copied					
	No invalid votes are added					
	Nobody can vote instead of abstained voters					
	Any single authority corruption is detected					
	Any number of authorities' corruption is detected					
	The dishonest voter cannot disrupt the voting					
	Voter can make objection during the voting process if there is an error					
	Anyone cannot disrupt the voting					
	Voters can complete voting process even if there is a physical error					

Table 8.4: Checklist for individual verifiability.

Main requirement	Requirement details	Satisfied (√)	Not satisfied (X)	Not Applicable (N/A)	How it is satisfied?	Assumptions
Individual Verifiability	Voter can validate that the ballot is correct					
	Voter can validate that authorities response correctly					
	Voter can validate that his vote is recorded correctly					
	Voter can safely re-request data during the voting process if authority response time outs					
	Each eligible voter can verify that his vote is counted correctly by using published data					

8.2 Analysis of DynaVote

DynaVote solves the voting dilemma by using PVID scheme, dynamic ballots extended with PreFotes and bulletin boards. This section examines how DynaVote achieves voting security requirements. Before going into detail, first provided are Table 8.5 and Table 8.6, which summarise and depict each authority's internal lists, publicly announced lists and public hash values written on bulletin boards. Table 8.5 exhibits that all data in internal lists are written on bulletin boards in some way, except $PVID_1$ and β_y in Collector's VoteList. However, these two data are also checked by Counter at the counting stage. $PVID_1$ is checked from the BallotList and β_y is checked from VotingKeyList. If there is any inconsistency, then it denotes that Collector is corrupted, not other authorities. Any corruption is detected in DynaVote. Furthermore, corruptions during the voting stage are maintained.

Table 8.5: Voting stage process data.

Authority	**Internal Data**	**Bulletin Board**
Ballot Generator	BallotList: | $PVID_1$, M_3, **B**, **Q**, β_x |	BGBB: **H**(**B**, **Q**, β_x) **H**(**B**, **Q**, $PVID_1$)
Key Generator	VotingKeyList: | β_y, β_z, δ_z, CCode |	KGBB: **H**(β_y, β_z) **H**(β_y, δ_z, CCode)
Collector	VoteList: | $PVID_1$, β_y, $\check{\mathbf{E}}_z$(**V'**, **Q**, $PVID_2$, CCode), $\check{\mathbf{E}}_z$(**V'**, **Q**, β_x), DateTime |	CBB: **H**($\check{\mathbf{E}}_z$(**V'**, **Q**, $PVID_2$, CCode)) **H**($\check{\mathbf{E}}_z$(**V'**, **Q**, β_x)) DateTime

Table 8.6: Counting stage published data.

Authority	Publicly Announced Lists	Time
Ballot Generator	Announced BallotList: | **B**, **Q**, β_x |	after voting stage, before counting
Key Generator	Announced VotingKeyList: | β_y, β_z |	after voting stage, before counting
Collector	Announced VoteList: | $\check{E}_z$(**V'**, **Q**, β_x), DateTime |	after voting stage, before counting
Counter	Published Dynamic Votes: | $PVID_1$, $PVID_2$, **V'**, Status |	before revealing actual votes
Counter	Published Tally: | β_z, **V'**, **Q**, **B**, **V**, Status, $\check{E}_z(PVID_1, PVID_2, \beta_x)$ |	after counting completed
Counter	Individual Vote Check List: | CCode, **V** |	after counting completed

During the election period, authorities only publish hash values of the internal data. Owing to this, nobody, including the voter, could reveal any data about the votes. Ballot Generator possesses dynamic ballots; Key Generator maintains voting keys and CCodes; and Collector keeps encrypted dynamic votes. Next section explains how DynaVote fully satisfies e-voting security requirements. Section 8.2.2 discusses specific cases of some corrupt participants trying to disrupt the election. Section 8.2.3 describes how DynaVote satisfies some desirable e-voting properties as well.

8.2.1 Fulfilment of Requirements in DynaVote

This section explains that how DynaVote protocol fulfils the e-voting security requirements as they are defined in Section 3.2 and formalised in section 8.1.1. In order to analyse DynaVote, let:

$P = \{p_1, p_2, p_3 \ldots p_q\}$ be the PVID-lists for all eligible voters, where $p_i = (PVID_1^i, PVID_2^i)$; and

$R = \{r_1, r_2, r_3 \ldots r_n\}$ be the random CCode lists for all voters, where $r_i = (CCode_{c_1}, CCode_{c_2}, CCode_{c_3} \ldots CCode_{c_k}, CCode_i)$. $CCode_i$ is the real CCode for a voter and the rest are the fake CCodes provided by PreFote list.

Lemma 1.1: DynaVote satisfies unlinkability.

Sketch of Proof: PVID Authority issues the blind signature on a voter's blinded IDs after checking his eligibility. According to the definition of blind signature, there is no function f satisfying $\forall p \in P \forall e \in E[\exists f(p) = e]$ for DynaVote. That is, no RegID is linkable to any PVID and vice versa. The voter does not use his RegID after obtaining PVIDs; instead, he uses his PVIDs in next stages. All internal lists and published lists contain only PVIDs. Among the internal and published information, no adversary, including all authorities acting in unison, can find a function f such that $\forall v \in V \forall e \in E[\exists f(S, W, D, v) = e]$. Thus nobody can break the voter-vote unlinkability.

Lemma 1.2: Assume that authorities do not cooperate in order to reveal voter's IP. Then DynaVote satisfies IP untraceability.

Sketch of Proof: In DynaVote protocol, none of the authorities keeps IP of the voters and releases them. DynaVote assures IP untraceability in normal case. In order to prevent corrupted Ballot Generator and Collector to trace voter's IP, the published lists do not give any evidence to them. Key Generator verifies and checks the election results by using bulletin boards and published lists. Ballot Generator and Collector do not keep IP addresses, even when, they do not send them to Counter. Thus, Counter cannot directly trace the IP. Key Generator does not know IP addresses either, since it does not directly communicate with the voter. Counter and Key Generator have all the election data except for IP addresses, and they do cross check each other.

There is no point in trying to trace the voter IP since nobody can guarantee whether or not the voter accesses over a dynamic IP, he uses the voting pool or any other public network, and he employs any IP anonymizer application. In case of a voter having a static IP and not taking any care about it, then IP untraceability may fail if authorities corrupt. Specifically if Counter or Key Generator cooperates with Ballot Generator and Collector in order to reveal a voter's IP; and if the voter has a static IP, it is least likely that they can trace his IP address. If it is assumed, therefore, that authorities do not

cooperate because they do not intend to reveal voter's IP address; DynaVote satisfies IP untraceability.

Lemma 1: DynaVote satisfies privacy.

Sketch of Proof: By the help of Lemma 1.1 and Lemma 1.2, we can say that DynaVote assures privacy since it satisfies unlinkability and IP untraceability.

Lemma 2: DynaVote satisfies eligibility.

Sketch of Proof: We employ PVID scheme which guarantees that only eligible voters can obtain valid PVIDs. PVID Authority issues blind signature on voter's blinded IDs after checking voter's eligibility. Only eligible voters' blinded IDs are blindly signed by PVID Authority. Assume that there exists such a function $f:P \rightarrow E$, $f(p_i) = e_i$ which can be known by only the voter himself; then DynaVote satisfies $\forall p \in P[\exists! e \in E \mid f(p) = e]$. Ineligible people's blinded IDs cannot be signed without being detected since threshold cryptography is applied to distribute the authority over *n* parties. In order to sign any request at least *t* parties should come together. In DynaVote, a voter can vote multiple times, but only the latest one is counted; the rest is discarded. Recast votes are recognised by the associated PVIDs. Thus, DynaVote achieves eligibility requirement.

Lemma 3: DynaVote satisfies uniqueness.

Sketch of Proof: In the counting stage, Counter obtains a final list $|PVID_1, PVID_2, \beta_y, \beta_x, \beta_z, \delta_z$, **V'**, **Q**, **B**, **V**, $\check{\mathbf{E}}_z$(**V'**, **Q**, $PVID_2$, CCode), $\check{\mathbf{E}}_z$(**V'**, **Q**, β_x), DateTime, Status, CCode|. A voter can recast; however, the last vote is taken into consideration, and the previous ones are discarded by the help of **Q** and DateTime. Since the $PVID_1$-$PVID_2$ is unique in the list and can be verified using PVID Authority's public key, there is no chance that more than one vote is counted for any voter. There exists such a function $f:V \rightarrow P$, $f(v_i) = p_j$ known by the voter himself, Counter and Key Generator. Here, DynaVote satisfies $\forall v_i \in V \forall v_j \in V[f(v_i) = f(v_j) \leftrightarrow i = j]$. Therefore, uniqueness is achieved.

Lemma 4: Assume that one of the voting authorities does not conspire with others to get a partial result of the election during the voting stage. Then, DynaVote satisfies fairness.

Sketch of Proof: Counting comes after the voting stage is completed, so no one can gain any partial knowledge about the tally before the counting stage is completed. Since we are employing dynamic ballots, Collector just knows voters' encrypted dynamic votes during the election which do not reveal any information without dynamic ballots. Even if Ballot Generator provides Collector with the corresponding dynamic ballot **B**, Collector cannot extract the voter's cast vote since the vote encryption key, which is maintained by Key Generator, is required. Thus, Collector cannot obtain the partial result. None of the authorities send any data to Counter during the election period; Counter cannot start counting before the end of the election.

As a consequence, in order to reveal the actual candidate for a vote $b \in B$, one should know {**B**, **V'**, δ_z}, as well as the relation between these data. Due to the fact that DynaVote confides trust to different authorities and that voting data are partially known by each authority, any single authority cannot disrupt fairness. However, if all of the authorities conspire, then they can start counting. Still, however, they could not get the accurate partial result because of the recasting feature. As a result, fairness is achieved conditionally.

Lemma 5: DynaVote satisfies uncoercibility.

Sketch of Proof: The proposed protocol allows recasting. If someone coerces a voter, even by only being physically next to him, the voter will cast in way the coercer influences. Later, he can change his vote, by recasting a new vote which will automatically discard the old one in the counting stage. Even if the voter records his voting activity, still he cannot convince the coercer of the content of his vote due to recasting. That is, practically it is not possible to coerce or vote buy, since nobody can know whether the current vote will be the final one or not.

Furthermore, a voter can cheat the coercer by using the fake CCodes provided by PreFote scheme. He can keep any number of fake CCodes to show the coercer. Due to the recasting feature and fake CCodes provided by PreFote scheme, there is no function f satisfying $\forall a \in A \forall v \in A[\exists f(D, W, S, P, R, a) = v)]$ for DynaVote. In practice, there is no point in coercing either physically or socially. Therefore, uncoercibility is achieved.

Lemma 6: DynaVote satisfies receipt-freeness.

Sketch of Proof: Voter has RegID, his permanent public-private key pair, CCodes and PVID-list and these data do not give any information about a voter's vote. Ballot Generator, Key Generator, Collector and Counter have no information about RegID and voter's permanent public-private key pair. PVID list does not provide a receipt to the voter either. Even if the voter provides the random numbers used in PVID-list generation, they cannot be used as a receipt. These random numbers will only prove that PVIDs belong to that voter, but they will not disclose any information about the voter's actual vote. Counter announces |$PVID_1$, $PVID_2$, **V'**, status (valid/discarded)| list, but it only gives information about dynamic votes. Dynamic votes cannot be mapped to actual votes without corresponding to the dynamic ballot. Dynamic ballot mechanism provides great facility as to this requirement. Although all information necessary to verify the election system is publicly known, a voter still cannot construct a receipt which can prove the content of his vote to a third party. In addition, if PVIDs are sold or bought, then it has no sense since they can be used again during the election. Voter can copy the PVIDs and recast.

Individual vote check list |CCode, V|, provides CCodes and associated actual votes, i.e. the selected candidates. Each voter keeps his CCode and individually checks if his selected candidate is listed correctly or not in individual vote check list. He cannot use CCodes as a receipt since the voter may have fake CCodes. These are provided by extended dynamic ballot mechanism. Each candidate on the dynamic ballot has a CCode and a voter can keep these fake CCodes as well as his real one. All CCodes are listed in the individual vote check list, and any coercer cannot disclose the difference between fake CCodes and real ones. Only the voter knows the truth. Even if the voter records his voting activity, still he cannot convince anybody that it was the actual vote due to recasting feature.

There exists a Boolean function *f* on *R* such that, "*f(x,a)=1* if *x* is the real CCode for voter *a"*, the function *f* can be provided by authorities, but is not known by any passive observer. Assume that there exists a probabilistic function *g* such that "*g(x,a)* defines the probability of *f(x,a)=1"*. Due to PreFote scheme, DynaVote protocol satisfies $\forall a \in A \forall r_i \in R[x \in r_i \wedge y \in r_i \rightarrow g(x,a) = g(y,a)]$. Thus, a voter cannot prove his vote to any passive observer. This being impossible, vote buying or selling is prevented, so receipt-freeness is fulfilled.

Lemma 7: DynaVote satisfies accuracy.

Sketch of Proof: During the voting stage, a voter verifies each step before proceeding to the next one. When he obtains the dynamic ballot **B** and vote encryption key β_z, he checks KGBB and BGBB; in case of any corruption, he can object to Ballot Generator. After voting, he also verifies CBB to assure that his vote is listed. The detailed explanation is given in individual verifiability requirement analysis.

As bulletin boards are employed, each authority has its own bulletin board and hash of all information related with voter's vote is recorded publicly. Thus, any corruption on the side of the authorities can be detected. Counter counts votes using the lists provided by Collector, Ballot Generator and Key Generator. Counter compares the VoteList, BallotList, VotingKeyList and announced lists for consistency and checks against the hash values in the bulletin boards. Any passive observer or organisation can also check the consistency of the election by using the announced lists and bulletin boards.

Any cast vote cannot be altered, deleted, invalidated or copied since the modification causes inconsistency with the bulletin boards. If the corrupted Ballot Generator publishes extra dynamic ballots on the BGBB, it will be found out at the counting stage since both Key Generator and Counter do not publish corresponding values on KGBB and CBB, respectively. Similarly, corrupted Key Generator and Counter could not publish any extra data on the bulletin boards. Thus, any corrupted authority could not manipulate the result of the election. Furthermore, the voter verifies his vote and makes an objection. During the voting process, any fraud can be maintained by the help of PVIDs. An authority cannot add any vote since a vote consists of a dynamic ballot **B** and a vote encryption key β_z. Even if Ballot Generator, Key Generator, Collector and Counter conspire together, they cannot add a new vote since they cannot create fake PVIDs. PVID Authority cannot issue fake PVIDs since threshold cryptography is applied. PVID scheme assures that $\forall a \in A[\exists! \, p \in P \rightarrow f_{ae}(a) \in E]$. Table 8.5 indicates that $PVID_1$, $PVID_2$ and actual votes are kept secret during the voting process, and having partial knowledge about voting data is not enough to vote or to simulate voter.

The dishonest voter cannot disrupt the voting; he has just right over his own vote, so he may only disrupt his vote. Even if he sends more than one votes, in this case, the last one is counted. Before revealing the actual votes, Counter performs some checks, such as whether the $PVID_1$ and $PVID_2$ are issued by PVID Authority and whether they belong to

same voter or not, on dynamic vote list and determines if the vote is valid or discarded. The voter is aware of that his previously sent votes will be discarded if he sends more than one vote. Thus, accuracy is achieved.

Lemma 8: DynaVote satisfies individual verifiability.

Sketch of Proof: Each eligible voter can verify that his vote is counted correctly. One of the major contributions of DynaVote is that a voter is given the opportunity to perform individual verifiability while casting his vote and at the end of the counting stage without revealing his identity.

Key Generator publishes $\mathbf{H}(\beta_y, \beta_z)$ in KGBB and Ballot Generator publishes $\mathbf{H}(\mathbf{B}, \mathbf{Q}, \beta_x)$ in BGBB. Voter attempts to create same hash values by using dynamic ballot **B**, request number **Q**, vote encryption key β_z and his session keys β_x and β_y. If he obtains the same values, he proceeds to send his dynamic vote to Collector.

In the ballot obtaining phase, if a voter receives corrupted vote encryption key β_z, he could not generate proper $\mathbf{H}(\beta_y, \beta_z)$. He can object to this situation by showing (β_y, β_z) and $\mathbf{H}(\beta_y, \beta_z)$. If the voter does not receive proper dynamic ballot **B**, he can prove that the dynamic ballot does not match with the hash values published in BGBB by showing $(\mathbf{B}, \mathbf{Q}, \beta_x)$ and $\mathbf{H}(\mathbf{B}, \mathbf{Q}, \beta_x)$. Therefore, Ballot Generator and Key Generator are required to respond to the voter properly. Otherwise, the voter can easily prove any improper responses. In these stages, any fraud is detected and corrected.

In the vote casting phase, voter checks CBB as soon as receiving the acknowledgement from Collector by creating same hash value for **V'** as $\mathbf{H}(\check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, PVID_2, CCode))$ and $\mathbf{H}(\check{\mathbf{E}}_z(\mathbf{V}', \mathbf{Q}, \beta_x))$. If the values do not match, he can object to Collector by illustrating **V'** and β_z. He can directly communicate with Collector and he can object to any corruption or any modification on the CBB since he uses PVIDs instead of his real identity. Thus, Collector could not modify the voter's dynamic vote **V'**. Voter individually verifies each step of the voting process (Note that this feature is called as verifiability or universal verifiability by some researchers).

Furthermore, the voter can verify the counting process by using the announced lists as other passive observers; he can check his dynamic vote individually from the published dynamic vote list announced by the Counter. By checking the published dynamic vote list, the voter is convinced that nobody has voted instead of him. This is an

important facility because most of the people do not trust computer networks and authorities.

As well as, he can check his CCode and selected candidate from individual vote check list. In other words, voters have CCode receipts, where individual vote check list provides a function f with these receipts such that $\forall e \in E \exists! v \in V[\exists f(e, W, CCode_{e_i}) = f_{vc}(v))]$. Thus, DynaVote satisfies direct individual verifiability as well as accuracy without providing any receipt to the voter.

Theorem 2: DynaVote is a complete and secure protocol.

According to the above sketch of proofs DynaVote satisfies *Lemma 1-8*. Thus, DynaVote is a complete and secure protocol. □

8.2.2 Specific Cases of Security Requirements Discussion

In this section we illustrate that DynaVote is strong and secure in terms of all specific cases defined in Section 8.1.2. The details of some cases which are explained in the previous section are not repeated here.

DynaVote protocol is secure against all of the cases related to privacy, eligibility, uniqueness and fairness, and this is illustrated in Table 8.7. However, it suffers from two specific cases related to privacy.

- If Counter or Key Generator cooperates with Ballot Generator and Collector in order to reveal a voter's IP; and the voter has a static IP then they can trace his IP address with a low probability.

- If all voting authorities and PVID authority corrupt and they cooperate in order to reveal Voter-Vote relationship, then they can get some useful information by processing the transaction times. However, in case of all authorities' being corrupt, we assume that there is no need for a democratic election.

Table 8.7: Fulfilment of privacy, eligibility, uniqueness and fairness.

Main requirement	Requirement details	Satisfied (√)	Not satisfied (X)	Not Applicable (N/A)	Assumptions
Privacy	Voter-Vote unlinkability	√			
	Voter-Vote IP untraceability	√			Authorities do not cooperate in order to reveal voter's IP
	Voters cannot add identifiable information	√			
	Authorities cannot add identifiable information	√			
Eligibility	Eligible voters can vote	√			
	Ineligible voters cannot vote	√			
	Authorities cannot give voting credentials to ineligible voters	√			
	Authorities cannot usurp suffrage (voting right)	√			
Uniqueness	At most one valid vote is counted for each eligible voter	√			
	Each eligible voter has voted only once			N/A	
Fairness	Result is not published till the end of the election	√			
	Counting comes after the voting stage	√			
	No one can guess the content of any cast vote	√			
	No one can gain any partial knowledge about the tally before the counting stage	√			All of the authorities do not cooperate in order ro get partial result of the election
	Encrypted votes are used and they are decrypted at the end of the election	√			

DynaVote protocol is secure against all of the cases related to uncoercibility and receipt-freeness except three; and this is illustrated in Table 8.8.

- If any coercer forces a voter by being physically next to him and if this voter does not recast, then uncoercibility fails since the coercer can be convinced that the voter has not recast by using published dynamic vote list.

- In the other way around, if a voter records his voting activity and votes only once, then receipt-freeness fails since he can convince a vote buyer that he has not recast by using published dynamic vote list.

- If the coercer forces a voter by being physically next to him at the PVID obtaining stage, then coercer can learn the PVIDs and understand if the voter has voted or not by using published dynamic vote list. Thus, DynaVote prevents forced abstention attack to some extent, if the voter obtains PVIDs by himself and keep them secretly, then nobody can understand whether he has voted or not.

One can notice that all of the three cases are directly raised because of the published dynamic vote list. If we do not announce the published dynamic vote list, then DynaVote can overcome these cases. However this list is directly used by voters to check that nobody has voted with their PVIDs. Thus we prefer publishing this list on behalf of these three specific cases, which are not common.

Table 8.8: Fulfilment of uncoercibility and receipt-freeness.

Main requirement	Requirement details	Satisfied (√)	Not satisfied (X)	Not Applicable (N/A)	Assumptions
Uncoercibility	Nobody can force voter to vote in a particular way	√			
	Nobody can force voter physically being next to him	√			Voter recasts
	Coercer cannot receive any proof from the voter after voting	√			
	Coercer cannot force the voter to use a particular proof provided before voting	√			
	Coercer cannot vote instead of voter with his personal ID	√			Voter recasts
	Forced abstention attack is prevented	√			Voter keeps his PVIDs secret
	Randomization attack is prevented	√			Voter recasts
Receipt-freeness	Voter is not identifiable from the receipt	√			
	Vote is not revealed from the receipt	√			
	Voter cannot prove his vote	√			
	Vote selling/buying is prevented	√			
	Authority gives correct receipt	√			Counter checks
	Any public data do not give any information about voter's vote	√			
	Voter cannot use a particular proof defined before voting	√			
	Voter cannot prove his vote even if he records his activity	√			
	Voter cannot obtain a particular proof after voting	√			
	Voter cannot use a personal ID such as RegID or private keys to prove his vote	√			

DynaVote protocol is secure against all of the cases related to accuracy and individual verifiability, and it is illustrated in Table 8.9 and Table 8.10. Although DynaVote provides direct individual verifiability, it does not sacrifice receipt-freeness, privacy or accuracy. The voter receipt CCode is generated by Key Generator, such as the vote encryption key. Due to the fact that Key Generator does not directly communicate with the voter, it cannot reveal any information about the voter. If corrupted Key Generator gives fake CCode instead of a real one to the voter, this situation is detected at the counting stage by Counter, and that vote is marked as valid. In this stage Counter cannot count fake votes in the final tally as there is no data in any lists and bulletin boards related with the fake votes. Any distortion on the individual vote check list and CCodes only disrupts the voter's individual verifiability partially, which indeed provides no benefit. Such a distortion does not have any effect on the final tally and election results, neither does it on accuracy. In addition, the voter can still individually verify dynamic vote list.

Table 8.9: Fulfilment of accuracy.

Main requirement	Requirement details	Satisfied (√)	Not satisfied (X)	Not Applicable (N/A)	Assumptions
Accuracy	Ballot representation is correct	√			
	Authorities response correctly	√			
	Voter can vote as intended	√			
	Vote is recorded correctly	√			
	All valid votes are counted correctly	√			
	No valid votes are deleted	√			
	No valid votes are modified	√			
	No valid votes are spoiled	√			
	No valid votes are copied	√			
	No invalid votes are added	√			
	Nobody can vote instead of abstained voters	√			
	Any single authority corruption is detected	√			
	Any number of authorities' corruption is detected	√			
	The dishonest voter cannot disrupt the voting	√			
	Anyone cannot disrupt the voting	√			
	Voters can complete voting process even if there is a physical error	√			

Table 8.10: Fulfilment of individual verifiability.

Main requirement	Requirement details	Satisfied (√)	Not satisfied (X)	Not Applicable (N/A)	Assumptions
Individual Verifiability	Voter can validate that the ballot is correct	√			
	Voter can validate that authorities response correctly	√			
	Voter can validate that his vote is recorded correctly	√			
	Voter can safely re-request data during the voting process if authority response time outs	√			
	Voter can make objection during the voting process if there is an error	√			
	Each eligible voter can verify that his vote is counted correctly by using published data	√			

8.2.3 Discussion on E-voting System Requirements and Properties

In this section we define how DynaVote fulfils some e-voting properties and helps satisfy certain system requirements. DynaVote provides following properties:

Scalability: DynaVote is scalable and applicable to large scale elections. It has no physical assumption such as untappable channels, voting booths, special hardware…etc. and it has no computational complexity in any stage of the protocol.

Practicality: DynaVote is practical since it employs PVID scheme, which is based on blind signature. It can be performed over an uncontrolled network, such as the Internet. DynaVote has one reasonable condition, which is the recasting feature. Due to the fact that this is an acceptable high level condition related to the election policy and that is not a mathematical assumption; recasting can be allowed by election authorities. Thus, DynaVote is truly practical and applicable.

Mobility: DynaVote protocol provides mobility since the voting is performed over Internet. There is no restriction on the location from which a voter can cast a vote.

Cheap Elections: The cost of voting by using DynaVote protocol is reasonably less than the cost of other electronic voting systems which require special hardware equipment such as DRE machines, special printers …etc.

DynaVote helps satisfy the following system requirements:

Efficiency: We have run some performance tests with DynaVote prototype implementation for both voting and counting processes. For voting, we have simulated 1000 voters by making their candidate selection randomly on moderate computers. The response times are in the order of milliseconds and seconds, which is satisfactory for a single voter process. Complexity of the counting process is O(n). The detailed and comprehensive benchmark tests will further be carried out as a future work.

Convenience: A convenient system allows voters to cast their votes quickly and in one session, without reliance on any extra equipment or special skills. Anyone who is familiar to use Internet can easily vote via DynaVote protocol.

Transparency: The whole voting process is transparent, and bulletin boards are used to publicise the election process. The security and reliability of the system is not reliant on the secrecy of the network or any other physical assumptions.

Equality of candidates: The protocol gives equal opportunity to all candidates. This is easily accomplished by dynamic ballots.

Empty Ballot: The protocol supports the empty ballot requirements, which means that a voter may cast a blank vote. This is also easily provided with dynamic ballots easily.

Open Source: Protocol's security and reliability does not rely on secrecy of the source code. The system can be implemented as open source.

8.3 Customisation of DynaVote

We present some available customizations for DynaVote in case of specific needs.

- DynaVote is currently performed with remote Internet voting and poll-site Internet voting. If the supreme election committee does not accept Internet voting, then DynaVote protocol can be performed over a closed and controlled network successfully. Recasting can be prohibited, and voting booths can be used. DynaVote protocol can be customised in this way easily.

- If the voter trusts in the system since accuracy is achieved and he does not need to verify his vote after the counting stage with CCodes, then PreFotes may not be used in the protocol. Note that a voter can still individually verify each step of the voting process and verify his vote from dynamic vote list.

- In case of receipt-freeness being unimportant, all PVIDs can be listed in the final tally result. In this case, PreFotes are not used in the protocol since a voter can verify his vote with PVIDs instead of CCodes. In this case, IP traceability should be taken into account. Otherwise, corrupted Ballot Generator and Collector may trace a voter's IP address.

- In case of IP provision of untraceability, Ballot Generator and Collector also take place in the counting stage in addition to Counter and Key Generator. The announced lists are changed to give an opportunity to Ballot Generator and Collector to verify the counting stage. In this case, all authorities should be corrupted to change election results.

- Instead of carrying out the election as a single day, a several days election period can be employed. In this case more flexibility and more voter involvement can be achieved.

8.4 Comparison with Other E-voting Protocols

In this section, we make a comparison between DynaVote and some selected protocols. Sampigethaya and Poovendran [93] classify voting protocols according to how voters submit votes to the tallying authority as: i) Hidden voter: The voters anonymously submit votes; ii) Hidden vote: The voters openly submit encrypted votes; and iii) Hidden voter with hidden vote: The voters anonymously submit encrypted votes. DynaVote is in the last group with outstanding features. Table 8.11 indicates that DynaVote is a complete protocol which covers the largest set of the e-voting requirements. Table 8.11 extends Table 4.1 in order to emphasise the contributions of DynaVote. Providing direct individual verifiability, as well as receipt freeness and uncoercibility, is one of the major contributions of DynaVote.

Table 8.11: Comparison of DynaVote.

Privacy Preserving Approaches	Voting Protocols	Security Requirements								System Requirements	
		Privacy	Eligibility Uniqueness	Fairness	Individual Verifiability	Accuracy	Dispute-freeness	Receipt-freeness	Uncoercibility	Scalability	Practicality
Mix-Nets	Chaum, 1981 [18]	Com	Y	N	Y	N	N	N	N	N	N
	Benaloh, 1987 [8]	Com	Y	C	N	Y	N	N	N	N	Y
	Chaum, 1988 [20]	Com	Y	N	Y	N	N	N	N	N	N
	Sako and Killian, 1995 [92]	Com	Y	C	N	Y	N	Y	N	N	N
	Chaum, 2004 [21]	Com	Y	C	C	C	N	Y	N	Y	C
	Chaum, 2005 [22]	Com	Y	C	N	C	N	Y	N	N	C
Homomorphic Encryption	Cohen and Fischer, 1985 [24]	Com	Y	N	N	Y	N	N	N	N	N
	Cohen and Yung, 1986 [25]	Com	Y	C	N	Y	N	N	N	N	N
	Iverson, 1992 [55]	Com	Y	C	Y	C	N	N	N	N	C
	Sako and Killian, 1994 [91]	Com	Y	C	N	Y	N	N	N	N	Y
	Cramer et al., 1996 [26]	Com	Y	C	N	Y	N	N	N	N	Y
	Cramer et al., 1997 [27]	Com	Y	C	N	Y	N	N	N	C	C
	Schoenmakers, 1999 [94]	Com	Y	C	N	Y	Y	N	N	N	Y
	Hirt and Sako, 2000 [53]	Com	Y	C	N	Y	N	Y	N	N	C
	Baudron et al., 2001 [7]	Com	Y	C	N	Y	N	Y	N	C	C
	Lee and Kim, 2002 [67]	Com	Y	C	N	Y	N	Y	N	Y	N
	Kiayias and Yung, 2002 [64]	Com	Y	C	N	Y	Y	N	N	N	Y
	Acquisti, 2004 [2]	Com	Y	C	N	C	N	Y	C	N	N
Blind Signature with Anonymous Channels	Fujioka et al., 1992 [42]	Com	Y	Y	Y	N	N	N	N	Y	N
	Baraani et al., 1994 [6]	Com	Y	Y	Y	C	N	N	N	N	N
	Cranor and Cytron, 1997 [28]	Com	Y	C	Y	C	Y	N	N	C	N
	Okamoto, 1997 [81]	Com	Y	C	C	N	N	Y	N	N	N
	Juang et al., 2002 [60]	Com	Y	C	Y	C	N	N	N	Y	Y
	Golle et al., 2002 [48]	Com	Y	C	Y	C	N	N	N	C	Y
	Lee at al., 2003 [68]	Com	Y	C	N	Y	N	Y	N	C	N
	Juels et al., 2005 [61]	Com	Y	C	N	C	N	Y	C	N	N
Blind Signature without Anonymous Channels	Mu and Varadharajan, 1998 [75]	Com	Y	C	N	C	N	N	N	N	C
	He and Su, 1999 [50]	Com	Y	C	C	C	N	N	N	N	C
	Ray et al., 2001 [85]	Com	Y	C	Y	Y	N	N	N	C	N
	Yang et al., 2004 [102]	Com	Y	C	N	Y	N	N	N	N	N
Blindly Signed Identity	DynaVote, 2007	Com	Y	C	Y	Y	Y	Y	Y	Y	C

Com: Computational
C: Conditionally satisfied
N: No, not satisfied
Y: Yes, satisfied

Note: Even if the feature of an objection and recovery is not provided in paper based voting, DynaVote enables voters to object to any corruption or failure during the voting stage, so that any inconvenience is recovered. In other words, a voter can easily recover from an interruption in the voting process. Thus, DynaVote satisfies dispute-freeness.

Table 8.11 depicts that there are only two voting protocols which satisfy receipt freeness and uncoercibility; Acquisti [2] and Juels *et al.* [61]. Acquisti's protocol is based on the homomorphic properties of Paillier cryptosystem and applies mix-nets. Juels *et al.* assumes voter access to an anonymous channel at some point during the voting process.

They certainly do not satisfy individual verifiability. Moreover, they are not scalable and practical.

It is also seen that Chaum's 2004 protocol [21] is the only one protocol which conditionally satisfies individual verifiability, receipt-freeness and accuracy at the same time. It is a DRE-based voting protocol (with physical voting equipment assumptions) and uses a two-layer receipt based on transparent sheets. However, it is not coercion resistant and practical. Moreover, Karlof *et al.* [62] discovered several potential weaknesses in Chaum's protocol which only became apparent when considered in the context of an entire voting system. These weaknesses are directly related to accuracy and privacy which decrease security level of the protocol. For example, if an adversary can determine that certain ballots will not be verified, he can unnoticeably alter or replace these ballots; and accuracy fails.

Chaum *et al.* [22] presents an election scheme based on [21]. Although the authors claim that the protocol is voter verifiable, it does not provide direct individual verifiability indeed. The protocol allows voters to verify that their vote is accurately recorded; in other words, it satisfies some accuracy needs. However, it does not mean individual verifiability or voter-verifiability in terms of context of this thesis. The authors also state this by the following quotation: "Voter cannot directly link her input vote strip to any specific resulting vote, and so she cannot directly verify that her vote has been correctly decrypted. However, the fact that the votes are all correctly processed can be checked to a high degree of confidence provides voter with the assurance that her vote will be decrypted correctly". This means that a high degree of accuracy is enough to trust the system. We actually do not agree with this. Moreover, the protocol has a few conditions for receipt freeness, but they are reasonable.

CHAPTER 9

CONCLUSION AND FUTURE WORK

In this chapter conclusions are drawn and future work is suggested.

9.1 Conclusion

Electronic voting refers to the use of computers or computerised voting equipment to cast ballots in an election, and it is not an easy task due to the need of achieving electronic voting security requirements. In the literature different sets of requirements are defined, and almost all academic studies focus on a subset of these requirements. Based on a detailed review of secure election system characteristics, this thesis proposes an extensive electronic voting requirement set with clear definitions.

Verifiability and receipt freeness in cryptographic voting protocols are examined in detail and the trade-off between receipt-freeness and individual verifiability is pointed out. Then, an applicable solution in order to overcome the voting dilemma is suggested by introducing PreFote scheme, which provides direct individual verifiability without sacrificing receipt-freeness and accuracy.

A comprehensive literature review having been carried out, this thesis provided a classification of the existing privacy preserving approaches and taxonomy of the existing cryptographic voting protocols extending the previous studies. The literature review has made it clear that there is no complete and secure cryptographic voting protocol which satisfies all electronic voting security requirements (especially receipt-freeness, uncoercibility and individual verifiability) at the same time. As the need of an alternative privacy preserving approach has arisen, this thesis proposed a practical and low cost solution of satisfying voter privacy, which is PVID scheme. PVID scheme is based on blind signature and RSA. It allows recasting without sacrificing uniqueness. PVID scheme provides anonymous pseudo identities which are unlinkable to the voter's real

identity. By employing PVID scheme, practical and secure voting protocols can be proposed.

Furthermore, this study suggests replacing the usual ballot structure with dynamic ballot mechanism in order to strengthen accuracy, verifiability and fairness of voting protocols; and how to extend dynamic ballots with PreFotes is presented. By employing PVID scheme and extended dynamic ballots with PreFotes and using some cryptographic primitives (bulletin boards and cryptographic hash functions), this thesis proposes a complete and secure cryptographic voting protocol over a network for large scale elections, which is voter-verifiable, receipt-free and coercion-resistant. The proposed protocol, namely DynaVote, contributes to the literature mainly by presenting a practical and secure cryptographic voting protocol which fulfils all of the electronic voting security requirements: privacy, eligibility, uniqueness, fairness, uncoercibility, receipt-freeness, individual verifiability and accuracy. DynaVote has no physical assumption such as untappable channels, mix-nets, special hardware… etc., and it has no computational complexity in any stage of the protocol. Thus, it is suitable for large scale elections and it can be performed over an existing network such as the Internet. DynaVote is verifiable in each stage, and voter can object to any corruption without revealing his real identity.

Lastly, a method to analyse voting systems based on electronic voting security requirements is suggested and DynaVote protocol is examined in detail with this method. DynaVote and PVID prototype implementations also supported the analysis stage of the research studies. The thesis also discusses how DynaVote satisfies many of the electronic voting properties and system requirements such as transparency, efficiency and mobility.

9.2 Future Work

Electronic government transformation forces many applications to be done on an electronic environment, and electronic democracy is affected by this transformation. Electronic voting is the core of electronic democracy as an inter-disciplinary subject and should be studied together with the experts of different domains such as cryptography, software engineering, politics, law, economics and social sciences.

As a future work, the main aim is to make DynaVote an applicable alternative for paper based voting system by initiating and carrying out a comprehensive project which

covers every aspects of an election process. Actors other than cryptographers should participate in and contribute to the project.

In particular, we will first complete the performance tests of the prototype to increase its efficiency and quality. In the current state of the prototype, we have used error free inputs; thus, we will advance error handling of the software. The graphical user interface will be improved as well. Threshold cryptography is not implemented within the prototype since RSA threshold cryptography has been already implemented in Java and the source codes are available [98]. Thus, we would like to integrate that work into our prototype.

Furthermore, we intend to extend PVID scheme with Identity based (ID-based) blind signatures [103]. An ID-based blind signature scheme is considered to be the combination of a general blind signature scheme and an ID-based one; it is attractive since one's public key is simply his/her identity. ID-based public key setting can be a good alternative to certificate-based public key settings in voting protocols. As a future work, we will review ID-based blind signatures and examine their applicability to voting protocols.

REFERENCES

[1] M. Abe, "Universally verifiable mix-net with verification work independent of the number of mix-servers", *In Advances in Cryptology - EUROCRYPT'98*, volume 1403, pp. 437–447, 1998.

[2] A. Acquisti, "Receipt-free homomorphic elections and write-in voter verified ballots", *ISRI Technical Report CMU-ISRI-04-116*, Carnegie Mellon University, PA, 2004.

[3] R. Aditya, B. Lee, C. Boyd, and E. Dawson, "Implementation issues in secure e-voting schemes", *The 5th Asia-Pacific Industrial Engineering and Management Systems Conference*, Goldcoast, Australia, 2004.

[4] R. Aditya, B. Lee, C. Boyd, and E. Dawson, "An efficient mixnet-based voting scheme providing receipt-freeness", *First International Conference on Trust and Privacy in Digital Business*, Zaragoza, Spain, pp. 152-161, 2004.

[5] ANSI X9.17 (Revised), "American National Standard for Financial Institution Key Management (Wholesale)", *American Bankers Association*, May 1985.

[6] A. Baraani, J. Pieprzyk, and R. Safavi, "A practical electronic voting protocol using threshold schemes", *Centre for Computer Security Research, University of Wollongong*, Australia, 1994.

[7] O. Baudron, P. A. Fouque, D. Pointcheval, G. Poupard, and J. Stern, "Practical multi-candidate election system", *In Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC '01)*, Newport, RI, USA, pp. 274-283, 2001.

[8] J. Benaloh, "Verifiable secret-ballot elections", *PhD thesis, Yale University*, 1987.

[9] J. Benaloh, and D. Tuinstra, "Receipt-free secret-ballot elections", *In Proceedings of the 26th ACM Symposium on the Theory of Computing*, pp. 544-553, 1994.

[10] J. Benaloh, "Simple verifiable elections", *In Proceedings of the Usenix/Accurate Electronic Voting Technology Workshop*, Berkeley, CA, USA, 2006.

[11] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules", *In Proceedings of the 14th IEEE Workshop on Computer Security Foundations (CSFW)*, Canada, pp. 82-96, 2001.

[12] D. Boneh, and M. K. Franklin, "Efficient generation of shared RSA keys", *Journal of the ACM (JACM)*, vol. 48-4, pp. 702-722, 2001.

[13] M. Burmester, and E. Magkos, "Towards secure and practical e-elections in the new era", *Chapter in Information Security - Secure Electronic Voting*, Kluwer Academic Publishers, pp. 63-76, 2003.

[14] J. Camenisch, and A. Lysyanskaya, "A formal treatment of onion routing", *In Advances in Cryptology - CRYPTO'05*, pp. 169-187, 2005.

[15] D. Cansell, J. P. Gibson, and D. Mery, "Formal verification of tamper-evident storage for e-voting", *In Proceedings of the 5th IEEE International Conference on Software Engineering and Formal Methods (SEFM'07)*, London, UK, pp. 329-338, 2007.

[16] O. Cetinkaya, and D. Cetinkaya, "Verification and Validation Issues in Electronic Voting", *the Electronic Journal of e-Government (EJEG)*, vol. 5-2, pp 117-126, 2007.

[17] O. Cetinkaya, and A. Doganaksoy, "Pseudo-Voter Identity (PVID) scheme for e-voting protocols", *In Proceedings of the International Workshop on Advances in Information Security (WAIS'07) in conjunction with ARES'07*, Vienna, Austria, pp. 1190-1196, 2007.

[18] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms", *Communications of ACM*, vol. 24, pp. 84-88, 1981.

[19] D. Chaum, "Blind signatures for untraceable payments", *CRYPTO'82*, pp. 199-203, 1982.

[20] D. Chaum, "Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA", *In Advances in Cryptology - EUROCRYPT'88*, Springer Verlag, pp. 177-82, 1988.

[21] D. Chaum, "Secret-ballot receipts: True voter-verifiable elections", *IEEE Security & Privacy*, vol. 2-1, pp. 38-47, 2004.

[22] D. Chaum, P. Y. A. Ryan, and S. Schneider, "A practical, voter-verifiable election scheme", *ESORICS'05*, Milan, Italy, pp. 118-139, 2005.

[23] B. Chevallier-Mames, P. A. Fouque, J. Stern, D. Pointcheval, and J. Traore, "On some incompatible properties of voting schemes", *IAVoSS Workshop On Trustworthy Elections*, Cambridge, UK, 2006.

[24] J. D. Cohen, and M. J. Fischer, "A robust and verifiable cryptographically secure election scheme (Extended Abstract)", *In Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, Portland, OR, pp. 372-382, 1985.

[25] J. D. Cohen (Benaloh), and M. Yung, "Distributing the power of a government to enhance the privacy of voters", *In Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*, Alberta, Canada, pp. 52-62, 1986.

[26] R. Cramer, M. Franklin, B. Schoenmakers, and M Yung, "Multi-authority secret ballot elections with linear work", *In Advances in Cryptology - EUROCRYPT'96*, Springer Verlag, pp. 72-83, 1996.

[27] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme", *EUROCRYPT'97*, Germany, 1997.

[28] L. Cranor, and R. Cytron, "Sensus: A security-conscious electronic polling system for the Internet", *Hawaii International Conference on System Sciences*, Hawaii, 1997.

[29] CyberVote, http://www.eucybervote.org, last accessed 20.07.2007.

[30] I. Damgard, and M. Koprowski, "Practical threshold RSA signatures without a trusted dealer", *In Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, London, UK, pp. 152-165, 2001.

[31] B. Davenport, A. Newberger and J. Woodar, "Creating a secure digital voting protocol for campus elections", *Princeton University*, 1996.

[32] N. Dedic, L. Reyzin, and S. Vadhan, "An improved pseudorandom generator based on hardness of factoring", *3rd Conference on Security in Communication Networks (SCN '02)*, Amalfi, Italy, 2002.

[33] S. Delaune, S. Kremer, and M. D. Ryan, "Verifying Properties of Electronic Voting Protocols", *IAVoSS Workshop On Trustworthy Elections (WOTE'06)*, Cambridge, UK, pp. 45-52, 2006.

[34] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, "How to share a function securely", *In Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC'94)*, ACM Press, NY, pp. 522-533, 1994.

[35] Y. Desmedt, and Y. Frankel, "Threshold cryptosystems", *In Advances in Cryptology - CRYPTO'89*, Santa Barbara, CA, pp. 307-315, 1990.

[36] Y. Desmedt, and Y. Frankel, "Shared generation of authenticators and signatures (Extended Abstract)", *In Proceedings of the 11th Annual International*

Cryptology Conference on Advances in Cryptology, London, UK, pp. 457-469, 1992.

[37] Digital Signature Standard (DSS), "Federal Information Processing Standards Publication 186-2", *National Institute of Standards and Technology*, 2001.

[38] B. W. DuRette, "Multiple administrators for electronic voting", *BS Thesis, MIT*, 1999.

[39] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *In Advances in Cryptology - CRYPTO'84*, pp. 10-18. Springer-Verlag, 1985.

[40] O. Forsgren, U. Tucholke, S. Levy, and S. Brunessaux, "Report on electronic democracy projects, legal issues of Internet voting and users (i.e. voters and authorities representatives) Requirements Analysis", *European Commission CYBERVOTE Project*, D4 vol. 3, 2001.

[41] Y. Frankel, P. D. MacKenzie, and M. Yung, "Robust efficient distributed RSA key generation", *In Proceedings of the 30th Annual ACM symposium on Theory of Computing*, Dallas, TX, pp. 663 – 672, 1998.

[42] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections", *AUSCRYPT'92*, Australia, pp. 244-251, 1992.

[43] J. Furukawa, and K. Sako, "An efficient scheme for proving a shuffle", *In Advances in Cryptology - CRYPTO'01*, Berlin Germany, pp. 368-387, 2001.

[44] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust and efficient sharing of RSA functions", *In Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, London, UK, pp. 157-172, 1996.

[45] R. Gennaro, "An improved pseudo-random generator based on the discrete logarithm problem", *In Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pp. 469–481, 2000.

[46] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing for anonymous and private communications", *Communications of the ACM*, vol. 42- 2, pp. 39-41, 1999.

[47] P. Golle, M. Jakobsson, A. Juels, and P. Syverson, "Universal re-encryption for mixnets", *In Proceedings of CT-RSA'04*, pp. 163-178, 2004.

[48] P. Golle, S. Zhong, D. Boneh, M. Jakobsson, and A. Juels, "Optimistic mixing for exit-polls", *ASIACRYPT '02*, Springer-Verlag, pp. 451-65, 2002.

[49] J. Groth, "A verifiable secret shuffle of homomorphic encryptions", *In Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, Miami, FL, pp. 145-160, 2003.

[50] Q. He, and Z. Su, "A new practical secure e-voting scheme", *IFIP/SEC'98*, Austrian Computer Society, pp. 196-205, 1998.

[51] E. V. Herreweghen, "Unidentifiability and accountability in electronic transactions", *PhD thesis, Katholieke Universiteit Leuven*, 2004.

[52] M. A. Herschberg, "Secure electronic voting over the World Wide Web", *MS Thesis, MIT*, 1997.

[53] M. Hirt, and K. Sako, "Efficient receipt-free voting based on homomorphic encryption", *EUROCRYPT'00*, Bruges, Belgium, pp. 539-556, 2000.

[54] ISO/IEC 15408-2, "Information technology -- Security techniques -- Evaluation criteria for IT security -- Part 2: Security functional requirements", http://www.iso.org, 2005.

[55] K. R. Iverson, "A cryptographic scheme for computerized general elections", *In Advances in Cryptology - CRYPTO'91*, Springer-Verlag, pp. 405-19, 1992.

[56] M. Jakobsson, and A. Juels, "An optimally robust hybrid mix network", *In Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing (PODC 01)*, ACM Press, pp. 284-292, 2001.

[57] M. Jakobsson, A. Juels, and R. L. Rivest, "Making mix nets robust for electronic voting by randomized partial checking", *In Proceedings of the 11th USENIX Security Symposium*, pp. 339-353, 2002.

[58] Java, http://java.sun.com, last accessed 15.12.2007.

[59] R. Joaquim, A. Zuquete, and P. Ferreira, "REVS - A robust electronic voting system", *In Proceedings of the IADIS International Conference on e-Society*, Lisbon, Portugal, pp. 95-103, 2003.

[60] W. S. Juang, C. L. Lei, and H. T. Liaw, "A verifiable multi-authority secret election allowing abstention from voting", *The Computer Journal vol. 45-6*, Oxford University Press, UK, pp. 672-682, 2002.

[61] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections", *ACM Workshop on Privacy in the Electronic Society*, VA, pp. 61-70, 2005.

[62] C. Karlof, N. Sastry, and D. Wagner, "Cryptographic voting protocols: a systems perspective", *14th USENIX Security Symposium*, MD, 2005.

[63] S. S. Keller, "NIST-Recommended Random Number Generator Based on ANSI X9.31. Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms", 2005.

[64] A. Kiayias, and M. Yung, "Self-tallying elections and perfect ballot secrecy", *In Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems (PKC 2002)*, Springer-Verlag, pp. 141-58, 2002.

[65] K. Kim, "Killer application of PKI to Internet voting", *IWAP'02*, Springer Verlag, 2002.

[66] R. Lebre, R. Joaquim, A. Zuquete, and P. Ferreira, "Internet voting: improving resistance to malicious servers in REVS", *IADIS International Conference on Applied Computing*, Lisbon, Portugal, 2004.

[67] B. Lee, and K. Kim, "Receipt-free electronic voting scheme with a tamper-resistant randomizer", *In Proceedings of the ICISC'02*, Springer-Verlag, pp. 389-406, 2002.

[68] B. Lee, C. Boyd, E. Dawson, K. Kim, J. Yang, and S. Yoo, "Providing receipt-freeness in mixnet-based voting protocols", *In Proceedings of the ICISC '03*, pp. 261-74, 2003.

[69] H. T. Liaw, "A secure electronic voting protocol for general elections", *Journal of Computers & Security*, vol. 23, pp.107-119, 2003.

[70] E. Magkos, M. Burmester, and V. Chrissikopoulos, "Receipt-freeness in large-scale elections without untappable channels", *In Proceedings of the First IFIP Conference on E-Commerce, E-Business, E-Government (I3E)*, pp. 683–694, 2001.

[71] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, "Handbook of Applied Cryptography", *CRC Press*, 2001.

[72] R. Mercuri, "Rebecca Mercuri's Statement on Electronic Voting", online available: http://www.notablesoftware.com/RMstatement.html, last accessed 15.12.2007.

[73] M. Michels and P. Horster, "Some remarks on a receipt-free and universally verifiable mix-type voting scheme", *ASIACRYPT '96*, Springer-Verlag, 1996.

[74] L. Mitroud, D. Gritzalis, and S. Katsikas, "Revisiting legal and regulatory requirements for secure e-voting", *In Proceedings of the 16th IFIP International Information Security Conference*, Egypt, 2002.

[75] Y. Mu, and V. Varadharajan, "Anonymous secure e-voting over a network", *In Proceedings of the 14th Annual Computer Security Applications Conference*, AZ, pp. 293-299, 1998.

[76] MySQL, http://www.mysql.com, last accessed 15.12.2007.

[77] C. A. Neff, "A verifiable secret shuffle and its application to e-voting", *In Proceedings of the 8th ACM Conference on Computer and Communications Security*, Philadelphia, PA, pp 116-125, 2001.

[78] V. Niemi and A. Renvall, "How to prevent buying of votes in computer elections", *ASIACRYPT '94*, Springer-Verlag, pp. 164–170, 1994.

[79] NIST FIPS 180-3, Secure Hash Standard (SHS), http://csrc.nist.gov/publications/fips, last accessed 15.12.2007.

[80] T. Okamoto, "An electronic voting scheme", *In Proceedings of the IFIP World Conference on IT Tools*, Canberra, Australia, pp. 21–30, 1996.

[81] T. Okamoto, "Receipt-free electronic voting schemes for large scale elections", *In Proceedings of the 5th Security Protocols Workshop*, Springer-Verlag, pp. 125-132, 1997.

[82] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes", *EUROCRYPT'99*, pp. 223-238, 1999.

[83] C. Park, K. Itoh, and K. Kurosawa, "Efficient anonymous channel and all/nothing election scheme", *EUROCRYPT'93*, Lofthus, Norway, pp. 248-259, 1993.

[84] B. Pfitzmann, "Breaking efficient anonymous channel", *EUROCRYPT'94*, Perugia, Italy, pp. 332-340, 1994.

[85] I. Ray, and I. Ray, and N. Narasimhamurthi, "An anonymous electronic voting protocol for voting over the internet", *In Proceedings of the 3rd International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems*, San Juan, CA, 2001.

[86] M. K. Reiter, and A. D. Rubin, "Crowds: Anonymity for web transactions", *ACM Transactions on Information and System Security* vol. 1-1, pp. 66-92, 1998.

[87] A. Riera, and J. Borrell, "Practical approach to anonymity in large scale electronic voting schemes", *In Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, 1999.

[88] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21-2, pp. 120–126, 1978.

[89] R. L. Rivest, and W. D. Smith, "ThreeVotingProtocols: ThreeBallot, VAV, and Twin", *Electronic Voting Technology Workshop*, Boston, MA, 2007.

[90] Safevote, "Voting system requirements", *The Bell Newsletter*, ISSN 1530-048X, 2001.

[91] K. Sako, and J. Kilian, "Secure voting using partially compatible homomorphisms", *In Advances in Cryptology – CRYPTO'94*, Santa Barbara, CA, pp. 411-424, 1994.

[92] K. Sako, and J. Kilian, "Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth", *EUROCRYPT'95*, Malo, France, pp. 393-403, 1995.

[93] R. Sampigethaya, and R. Poovendran, "A framework and taxonomy for comparison of electronic voting schemes", *Elsevier Computers & Security*, vol. 25- 2, pp. 137-153, 2006.

[94] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its applications to electronic voting", *In Advances in Cryptology – CRYPTO'99*, Springer-Verlag, pp. 148-164, 1999.

[95] A. Shamir, "How to share a secret", *Communications of the ACM*, vol. 22-11, 1979.

[96] A. T. Sherman, A.Gangopadhyay, S. H. Holden, G. Karabatis, A. G. Koru, C. M. Law, D. F. Norris, J. Pinkston, A. Sears, and D. Zhang, "An examination of vote verification technologies: findings and experiences from the Maryland study", *In Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop*, Canada, 2006.

[97] V. Shoup, "Practical threshold signatures", *EUROCRYPT'00*, Springer Verlag, 2000.

[98] ThreshSig, "SourceForge.net: Java Threshold Signature Package", https://sourceforge.net/projects/threshsig, last accessed 15.12.2007.

[99] United States presidential election controversy and irregularities, http://en.wikipedia.org/wiki/2004_U.S._presidential_election_controversy_and_irregularities, last accessed 15.12.2007.

[100] VoteHere VHTi, http://www.votehere.com, last accessed 15.12.2007.

[101] X Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD", *Cryptology ePrint Archive*, 2004

[102] C. C. Yang, C. Y. Lin, and H. W. Yang, "Improved anonymous secure e-voting over a network", *Information & Security*, vol. 15-2, pp.181-194, 2004.

[103] F. Zhang and K. Kim, "ID-based Blind Signature and Ring Signature from Pairings", ASIACRYPT'02, Springer Verlag, pp. 533-547, 2002.

APPENDIX A

SUPPLEMENTARY CRYPTOGRAPHIC PRIMITIVES

RSA public key cryptosystem and threshold cryptography are briefly explained for unfamiliar readers.

A.1 RSA Public Key Cryptosystem

Public key cryptography, also known as asymmetric cryptography, is a form of cryptography in which a user has a pair of cryptographic keys namely public key and private key. The private key is kept secret, while the public key may be widely distributed. The task of computing private key for a given public key is computationally infeasible.

Separate keys offer a significant advantage over secret key algorithms, because the private key does not need to be shared at all, significantly reducing the chance the key will be compromised. Moreover, the same key pair can be used for communication with many parties, who would otherwise require many different secret keys, posing a difficult key management challenge.

The main objective of public-key encryption is to provide confidentiality and authenticity. So, public key cryptography provides two main functionalities as follows.

- *Public key encryption*: A message encrypted with a recipient's public key cannot be decrypted by anyone except the recipient possessing the corresponding private key. This is used to ensure confidentiality.

- *Digital signatures*: A message signed with a sender's private key can be verified by anyone who has access to the sender's public key, thereby proving that the sender signed it and that the message has not been tampered with. This is used to ensure data integrity and authenticity.

The RSA and ElGamal public key encryption schemes are the well known ones. There are also other schemes such as Elliptic Curve public key encryption based on the algebraic structure of elliptic curves over finite fields; Rabin's public-key encryption scheme which is provably as secure as factoring; McEliece public-key encryption scheme based on error-correcting codes; Chor-Rivest public-key encryption scheme based on the subset sum (knapsack) problem; and probabilistic public-key encryption schemes which are not very popular [71].

RSA is a public key algorithm that can be used for both encryption and digital signing. A message encrypted with a private key constitutes a digital signature because only the holder of that private key could have produced that encrypted message, provided the key has been kept secure. The corresponding public key is used to verify the signature and since the key is public, anyone is able to perform this test. The RSA public key cryptosystem [88] relies on the difficulty of factoring large numbers to provide its security. The keys for the RSA algorithm are generated in the following way:

- Choose two distinct large random prime numbers and p and q
- Compute $n = pq$, where n is the modulus for both the public and private keys and made public; while the two primes p and q are kept secret.
- Compute the $\Phi(n) = (p-1)(q-1)$
- Choose an integer e such that $1 < e < \Phi(n)$, such that $gcd(e, \Phi(n)) = 1$.
- Compute the unique integer d, $1 < d < \Phi(n)$, such that $ed = 1 \bmod \Phi(n)$.
- The public key is (e, n) and the private key is (d, n).

In order to encrypt a message m for the public key (e, n) one simply performs the following exponentiation:

$c = m^e \bmod n$, where c is the cipher

In order to decrypt cipher c, one should do the following:

$m = c^d \bmod n$, since $m^{ed} \bmod n = m \bmod n$.

RSA is suitable for signing as well as encryption. Suppose Alice wishes to send a signed message to Bob. She produces a hash value of the message, raises it to the *power of d mod n* (as she does when decrypting a message), and attaches it as a "*signature*" to the message. When Bob receives the signed message, he raises the signature to the *power of e mod n* (as he does when encrypting a message), and compares the resulting hash value with the message's actual hash value. If they agree with each other, he knows that the author of the message was in possession of Alice's secret key and that the message has not been tampered with since.

A.2 Threshold Cryptography

The *(t, n)* threshold cryptography [95], [35] is used to distribute highly sensitive secret information (i.e. a secret key) and computation (i.e. decryption or signing operations) between *n* participants in order to remove single point of failure so that only when more than *t* participants come together, the secret can be reconstructed and the computation can be performed. The required trust in the cryptographic service is distributed among the group of authorities in such a way that:

- Any *t-1* or fewer participants cannot figure out the secret and perform operation
- Only *t* or more participants can reconstruct the secret information and perform operation

One of the key features of threshold cryptography is robustness since even *t-1* corrupt participants cannot learn any information about the secret key or cannot forge a valid signature.

There are two feasible approaches for generating the secret shares. The simpler approach is for a dealer to generate the secret normally and split it into shares; then distribute these shares to the appropriate participants. A more secure approach is to have the participants generate the secret together, with no single party ever learning the complete secret in the process.

A.2.1 Secret Sharing

The concept of secret sharing based on Lagrange interpolation was first introduced by Shamir in 1979 [95]. In this scheme, a shared secret *s* is an element of a finite field,

where any *k*-sized subset of *n* shares reveals *s* ($k \leq n$), but any subset of size smaller than *k* reveals nothing about *s*.

Distinct elements $x_1 \ldots x_n$ are assigned to the *n* participants in the finite field. The dealer selects a random polynomial *P* of degree at most *k-1* over the finite field, such that *P(0) = s*. Then he computes the secret shares $y_i = P(x_i)$ for $i = 1 \ldots n$ *where* $x \neq 0$ and communicates y_i to the participant assigned x_i, for all *i*.

Note that each share is the pair *(x, y)*. The finite field, its elements $x_1 \ldots x_n$ and their assignments to the participants are public information, the shares $y_1 \ldots y_n$ and the polynomial *P* are secret.

Since *P* has degree at most *k-1*, it is uniquely determined by values at any *k* points. Therefore *k* points are sufficient to recover the polynomial *P* and in particular they can compute *s = P(0)*.

Lagrange interpolation gives the following formula:

$$P(x) = \sum_{i=1}^{k} y_i \left(\prod_{j=1,\, j \neq i}^{k} \frac{(x - x_j)}{(x_i - x_j)} \right)$$

Since we seek only the secret *P(0)*, we can skip the computation of the actual polynomial coefficients and go straight to:

$$s = P(0) = \sum_{i=1}^{k} y_i \left(\prod_{j=1,\, j \neq i}^{k} \frac{-x_j}{(x_i - x_j)} \right)$$

A.2.2 Threshold RSA Public Key Cryptosystem

In a *(t, n)* threshold cryptosystem the private key is *(t, n)* secret shared among the authorities, while one public key is published. Any group of at least *t* authorities can jointly decrypt messages encrypted under this public key using a distributed decryption protocol. An adversary thus needs to compromise at least *t* of the authorities to decrypt messages or to mount a denial of service attack against the system's cryptographic service. A minority of compromised authorities can be tolerated.

Threshold cryptography can also be used to distribute signature operations among several participants. In order to sign a message *m* more than *t* participants execute an interactive signature generation protocol by using their secret shared keys and obtain the signature of *m* that can be verified by anybody using the public key.

Threshold schemes based on the discrete log problem are relatively easy to build. On the other hand, there are some technical difficulties in RSA, in particular, key generation which requires that the product of two primes be obtained without any single party knowing these two primes. Desmedt and Frankel [35] briefly address RSA threshold signature scheme issues.

The general idea of RSA threshold decryption is to share a secret exponent *d*, then use Lagrange interpolation to recombine *k* out of *n* of these [36], [44], [34], [97]. Although earlier works assumed that it is difficult to carry out this interpolation over Z^*_n, Shoup [97] showed that, if the primes factors of the RSA modulus are "safe primes" *(p = 2q+1)* then polynomial interpolation is possible over Z^*_n.

Distributed efficient RSA key generation is first performed by Boneh and Franklin [12] such that no one party learns the factorization. Each participant obtains a share of the secret exponent *d*. Frankel *et al*. [41] did some improvements by making the generation process resistant against active attack. Damgard and Koprowski [30] combined the efficiency of the Shoup scheme and the distributed nature of the Frankel *et al*. scheme.

APPENDIX B

IMPLEMENTATION DETAILS

This chapter provides prototype implementation details and gives brief information on installation and prototype usage of DynaVote protocol and PVID scheme.

B.1 Software Packages

The package hierarchy used in the implementation is listed below and general overview is given in Figure B.1:

- `evoting.authorities.Ballot_Generator`
- `evoting.authorities.Collector`
- `evoting.authorities.Counter`
- `evoting.authorities.Key_Generator`
- `evoting.database`
- `evoting.PVID_Authority`
- `evoting.utils`
- `evoting.voter`

The packages in `evoting.authorities` are used for authorities whereas `evoting.voter` package contains the classes for voter. `evoting.utils` package includes classes for cryptographic functions, mathematical operations, file processing and some other supplementary functions. `evoting.database` package provides database operations.

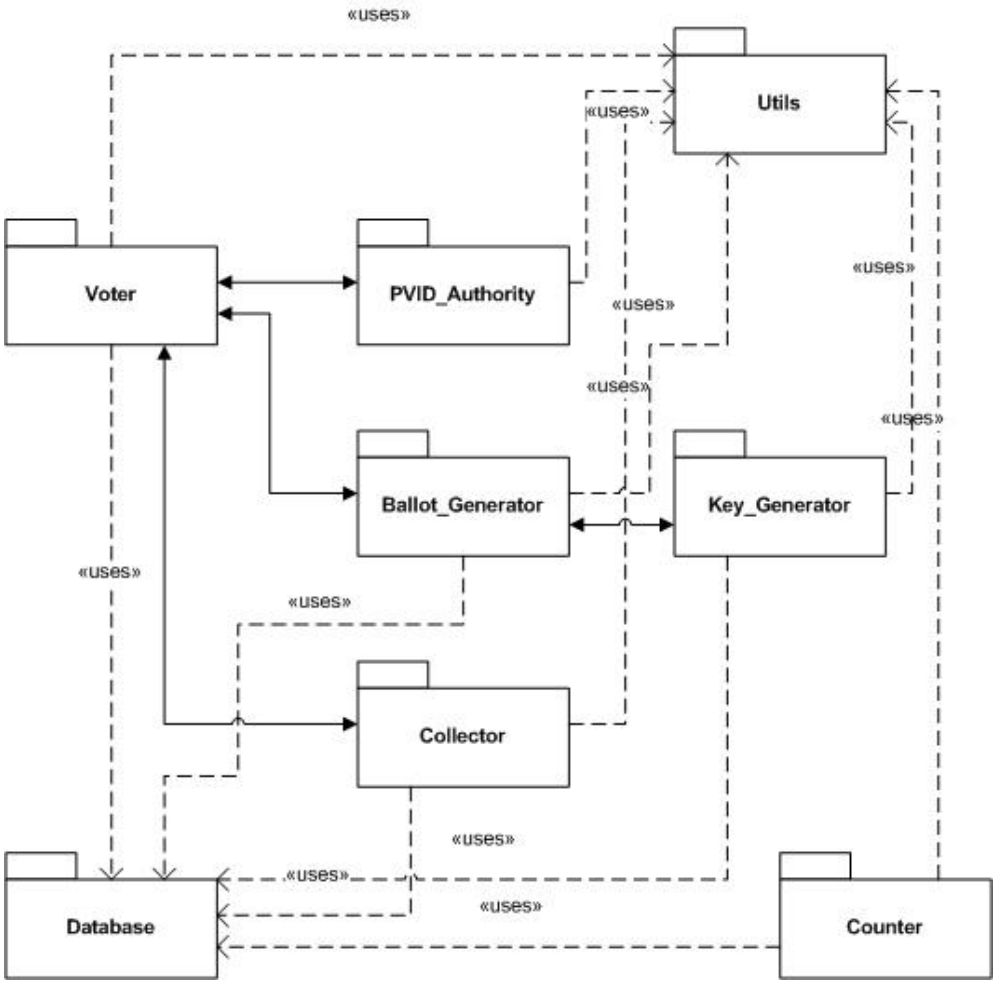

Figure B.1: Package hierarchy.

B.1.1 evoting.authorities.Ballot_Generator

This package contains `BallotServer`, `BallotServerThread` and `BallotServerProtocol` classes. `BallotServer` is the main class for Ballot Generator. The application listens on a dedicated port for voter connections and runs until the end of the election. `BallotServer` class uses multi threading. If a voter connects to the server, then an instance of `BallotServerThread` class is created.

Initially, server backs up data of old election, and then truncates the database for new election. At the end of the election it exports data on its own database and data in BGBB (Ballot Generator Bulletin Board) table of `BulletinBoards` database. Hence it

creates a <BallotGenerator.sql> data file in order to send Counter authority and it announces dynamic ballots in <BallotGenerator_Result.html>.

`BallotServerThread` is a class where messages are received from voter, checked and processed by calling a method of `BallotServerProtocol` class and sent back to the voter. The thread runs until the processed messages are not equal to terminating messages. `BallotServerProtocol` class defines the communication protocol between the voter and Ballot Generator during ballot obtaining phase. Dynamic ballot is prepared in this class with SHA1 PRNG algorithm. All the transactions are written to databases.

B.1.2 evoting.authorities.Collector

This package contains `CollectorServer`, `CollectorServerThread` and `CollectorServerProtocol` classes. `CollectorServer` is the main class for Collector. The relations between classes and general working scheme of this package are similar to `evoting.authorities.Ballot_Generator` package. The application listens on a dedicated port for voter connections and runs until the end of the election. Dynamic votes are collected with associated PVIDs in `CollectorServerProtocol` class.

B.1.3 evoting.authorities.Counter

This package contains `Counter`, `CounterGUI` and `Information` classes. `Counter` is the main class for counting and tallying operations. The application provides a user interface to process all data sent by Ballot Generator, Key Generator and Collector and to check consistency of internal lists, published lists and bulletin boards. `Counter` class includes some methods to check whether a vote will be counted in the final tally or it will be discarded as well. `CounterGUI` class is used to provide the application user interface. `Information` class is a small class which prints the election result and gives information about winner candidate.

B.1.4 evoting.authorities.Key_Generator

This package contains `KGServer`, `KGServerThread` and `KGServerProtocol` classes. The relations between classes and general working scheme of this package are

similar to evoting.authorities.Ballot_Generator package. KGServer is the main class for Key Generator. The application listens on a dedicated port for Ballot Generator connections and runs until the end of the election. It does not communicate with voter. Voter's public-private voting key pair for casting his dynamic vote is generated in KGServerProtocol class. At the end of the election, it creates two files, namely <KeyGenerator.sql> and <KeyGenerator_Result.html>.

B.1.5 evoting.database

This package contains four classes: VoterDatabase, BGDatabase, KGDatabase and CollectorDatabase. VoterDatabase class is implemented for voter to check consistency of data came from servers with data in BulletinBoards. Others are used for servers and they include internal lists of authorities. The classes are used to perform database operations for both authorities and voter.

B.1.6 evoting.PVID_Authority

This package contains PVIDServer, PVIDServerThread and PVIDServerProtocol classes. PVIDServer is the main class for PVID Authority. The relations between classes and general working scheme of this package are similar to evoting.authorities.Ballot_Generator package. The application listens on a dedicated port for voter connections and runs until the end of the election. Signed PVIDs are generated in PVIDServerProtocol class.

B.1.7 evoting.utils

This package contains Constants, CryptoUtil, FileUtil, GUIUtil and Keys_Construction classes. The methods, variables and constants in these classes are static and they are used by almost all packages.

Constants class includes configuration data. FileUtil class performs file I/O operations. Keys_Construction class includes methods to generate RSA public-private key pairs and to reconstruct RSA keys. GUIUtil class helps to produce user friendly results. It provides some conversion methods between different types and some concatenation and split operations for byte arrays.

`CryptoUtil` is one of the most significant classes in the prototype, since all cryptographic functions are implemented in this class. Five frequently used functions of this class are: sign, unsign, encrypt, decrypt and hashSha256 methods. Encryption function is implementation of $\check{\mathbf{E}}_b(m) = DES_{dk}(m) \parallel \check{\mathbf{E}}_b(dk)$; and sign function is implementation of $\check{\mathbf{S}}_b(m) = \check{\mathbf{D}}_b(\mathbf{H}(m)) \parallel m$.

B.1.8 evoting.voter

This package contains `Voter` and `VoterGUI` classes. `Voter` is the main class for the election web application. The web page provides voters to cast their votes. Communication between voter and authorities is carried out by this class. Voter requests dynamic ballot from Ballot Generator and casts his dynamic vote to Collector. Besides, voter verifies hashed values related to his vote against KGBB and BGBB databases. `VoterGUI` class is used to provide web user interface.

B.2 Prototype Usage

In this section, the software usage of the prototype is described. In order to perform an election, firstly PVID Authority, Ballot Generator, Key Generator and Collector servers should be started with the same election termination time parameter on command prompt as Java applications. Afterwards any voter can access to PVID application web page on Election web site over Internet to obtain signed PVIDs.

Election web site uses signed applets embedded in HTML files, so while using the system voters are notified about it and the system requests permission to read the files in flash memory to be able to reach voter's private key or his PVIDs. Web browser prompts standard signed applet warning and confirmation screens as shown in Figure B.2 and Figure B.3. Voter can examine the owner of the certificate and if he finds the certificate suitable, then he confirms it.

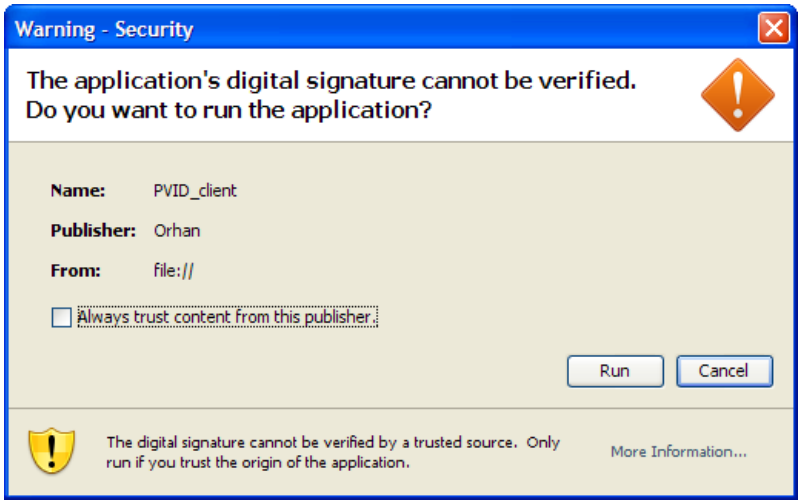

Figure B.2: Signed applet warning.

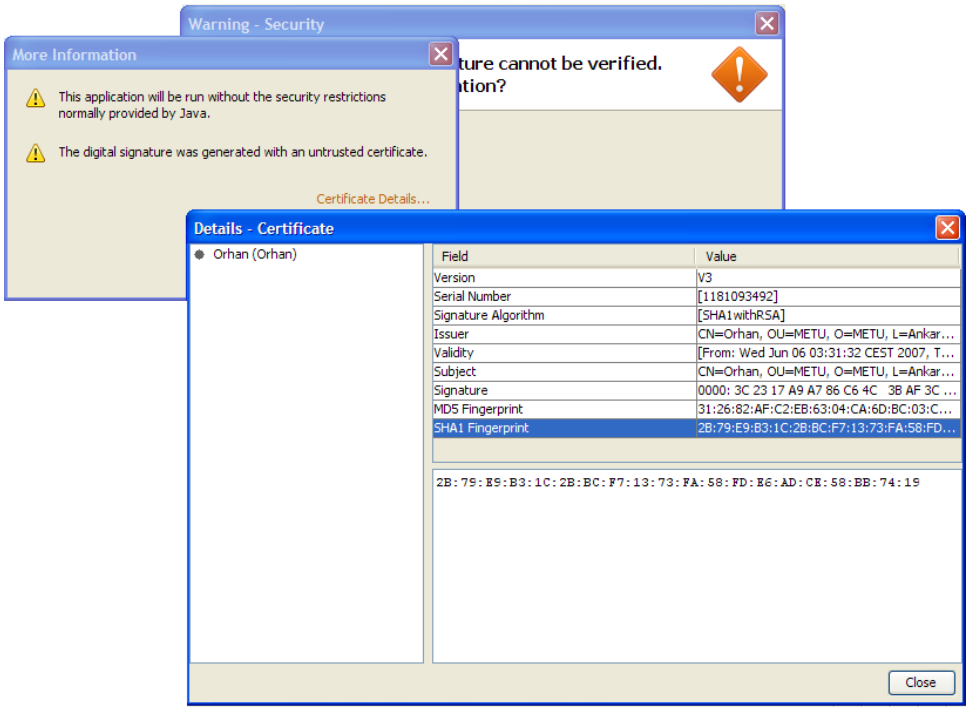

Figure B.3: Details of signed applet certificate.

PVID application web page has a simple user interface, which asks voter his registration ID and his private key. The private key is just used in client-side to encrypt voter messages.

Figure B.4 shows a screen shot of the PVID application web page which has printed after PVIDs are obtained.

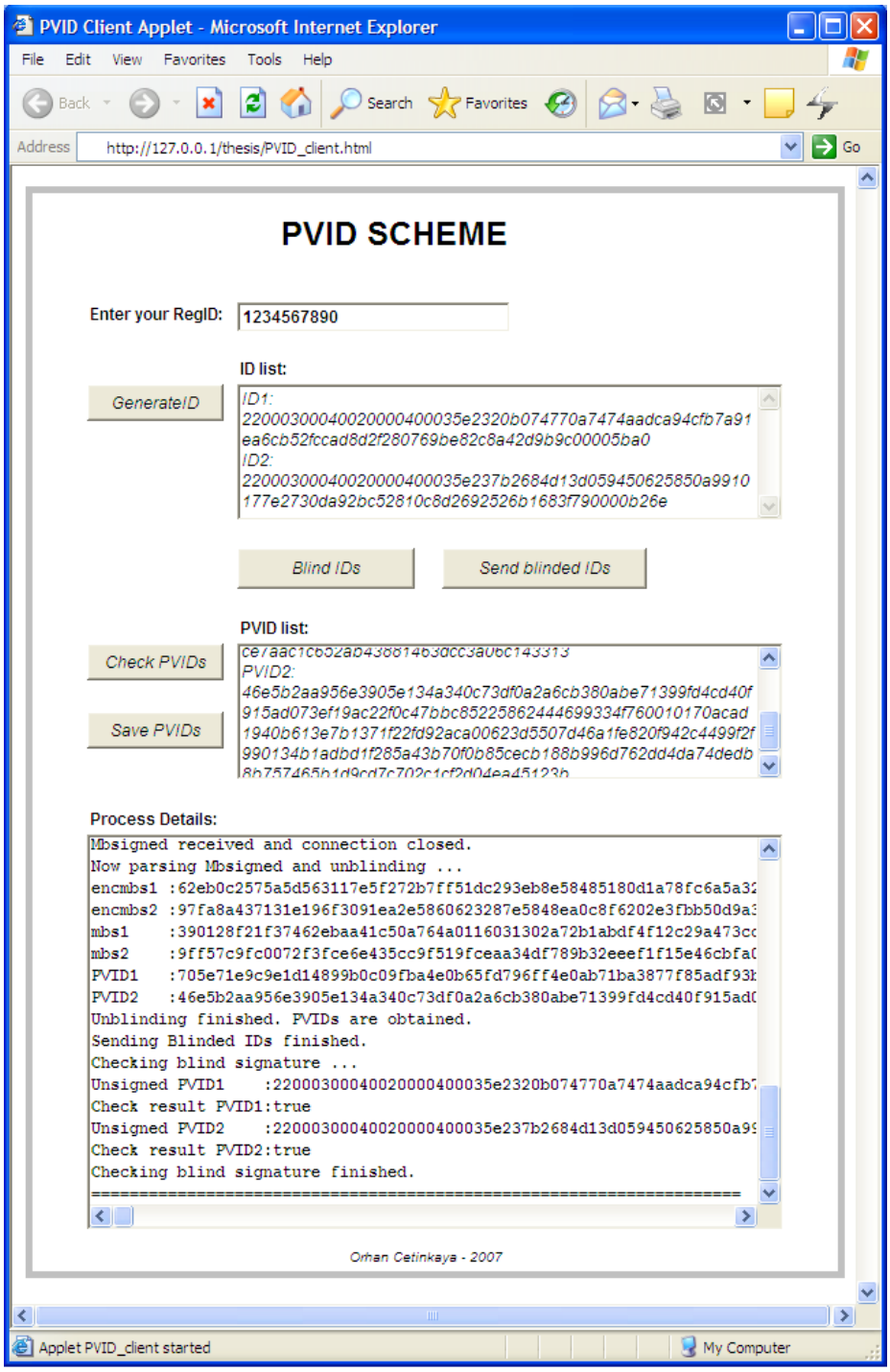

Figure B.4: PVID scheme prototype.

Then any voter can access to Voting web page on Election web site and he can perform voting process if he has valid PVIDs. Then, he votes the desired party by selecting from radio buttons, then click “Vote” button and then sees a popup window that warns voting process is completed. Figure B.5 shows a screen shot of the Voting web page which has printed after the voting process. If PVIDs are not valid, voter sees an error message and Voting web page is closed.

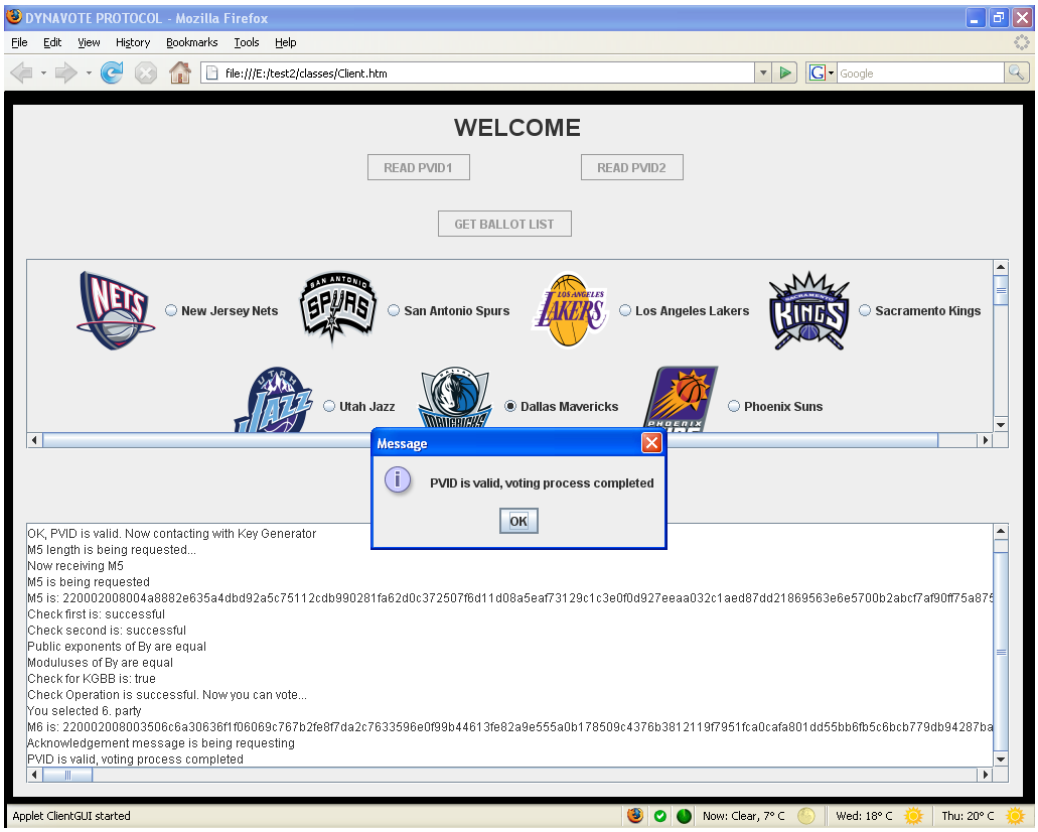

Figure B.5: Voting web page.

After election times out, all election data in server databases are exported by authorities. These exported data are sent to Counter server offline. Counter server application can be run after this point. Counter application imports all election data and then starts counting process. During the counting, it announces dynamic votes; and after tabulation, it opens a popup window that shows the winner. The number of cast votes for

each candidate and their percentage are also published. Figure B.6 shows a screen shot of Counter application.

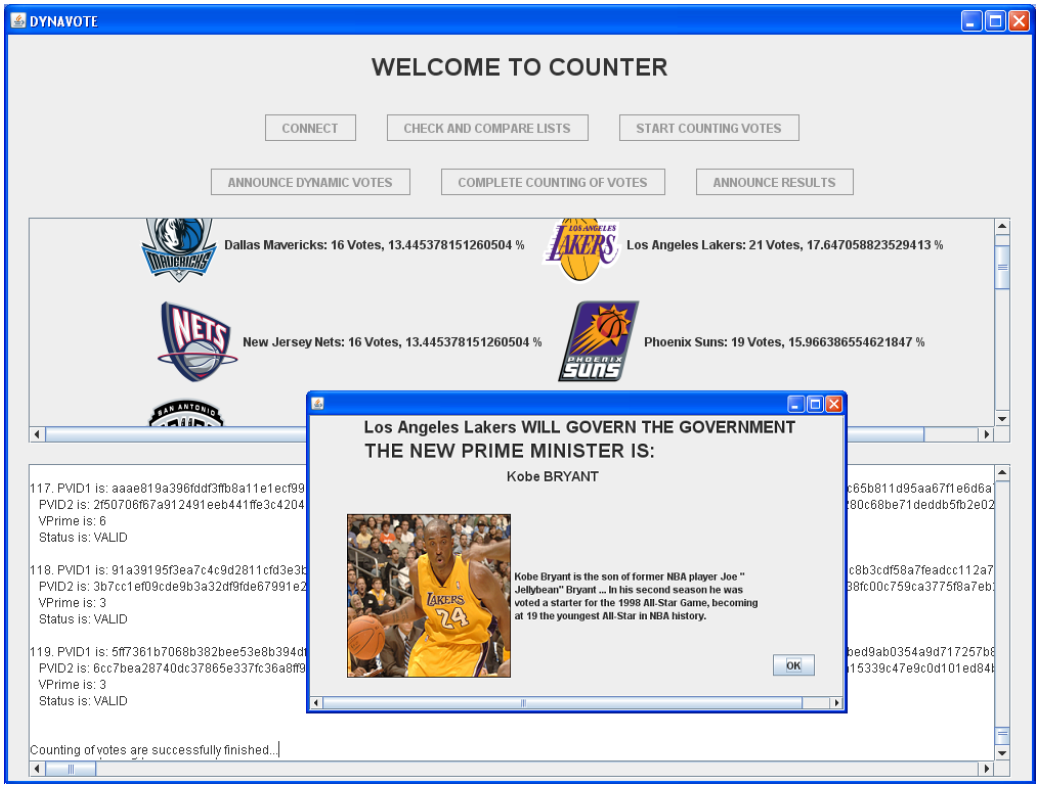

Figure B.6: Counter application.

B.3 Development Details

In this section some important source code is given.

```
-----------------------------------------
-----Signing the client applet------------
-----------------------------------------
keytool -genkey -keyalg rsa -alias keyOC
keytool -export -alias keyOC -file keyOC.crt

javac ClientGUI.java mainClient.java client.java
jar    cvf    ClientGUI.jar    ClientGUI.class    mainClient.class
client.class
client$buttonListener.class
jarsigner ClientGUI.jar keyOC
pause
```

```
-----------------------------------------
-----Database initialization--------------
-----------------------------------------
drop database if exists BallotGenerator;
drop database if exists Collector;
drop database if exists BulletinBoards;
drop database if exists KeyGenerator;
drop database if exists COUNTER;
drop database if exists VOTERS;

create database BallotGenerator;
create database Collector;
create database BulletinBoards;
create database KeyGenerator;
create database COUNTER;
create database VOTERS;

USE BallotGenerator;
CREATE TABLE AnnouncedListBG (
   B BLOB,
   Q INT,
   EBX VARBINARY(100),
   NBX BLOB);
CREATE TABLE BallotList (
   PVID1 BLOB,
   M3 BLOB,
   B BLOB,
   Q INT,
   EBX VARBINARY(100),
   NBX BLOB);
CREATE TABLE Parties (
   ELECTION_DATA INT,
   NAME VARCHAR(30),
   LOGOPATH VARCHAR(100),
   DESCRIPTION VARCHAR(1000));

USE Collector;
CREATE TABLE AnnouncedListCollector (
   ENCSECOND BLOB,
   DATE_TIME VARBINARY(100));
CREATE TABLE VoteList (
   PVID1 BLOB,
   EBY VARBINARY(100),
   NBY BLOB,
   ENCFIRST BLOB,
   ENCSECOND BLOB,
   DATE_TIME BLOB);

USE KeyGenerator;
CREATE TABLE VotingKeyList (
   EBY VARBINARY(100),
   NBY BLOB,
   EBZ VARBINARY(100),
   NBZ BLOB,
   DOZ BLOB,
   NOZ BLOB);
CREATE TABLE AnnouncedListKG (
```

```
   EBY VARBINARY(100),
   NBY BLOB,
   EBZ VARBINARY(100),
   NBZ BLOB);

USE BulletinBoards;
CREATE TABLE BGBB (DATA BLOB);
CREATE TABLE CBB (DATA BLOB, DATE BLOB);
CREATE TABLE KGBB (DATA BLOB);

USE Counter;
CREATE TABLE DynamicVote (
   PVID1 BLOB,
   PVID2 BLOB,
   EBX VARBINARY(100),
   NBX BLOB,
   EBY VARBINARY(100),
   NBY BLOB,
   EBZ VARBINARY(100),
   NBZ BLOB,
   DOZ BLOB,
   NOZ BLOB,
   VPRIME INT,
   Q INT,
   ENCFIRST BLOB,
   ENCSECOND BLOB,
   DATE_TIME BLOB,
   STATUS VARCHAR(50));


-----------------------------------------
-----Key parts of the source code---------
-----------------------------------------

-----------------------------------------
-----Using threads------------------------
-----------------------------------------
try {
  ServerSocket serverSocket = new ServerSocket(4444);

  //Reconstruct the RSA Private Key for Server
  ...
}
catch (IOException e) { e.printStackTrace(); }

while (listening) {
  new PVIDSignServerThread(serverSocket.accept(),
                                          rsa_priv_key).start();
}
serverSocket.close();


-----------------------------------------
-----How thread works---------------------
-----------------------------------------
public class PVIDSignServerThread extends Thread {
  private Socket socket = null;
  private RSAPrivateKey priKey = null;
```

```
  public PVIDSignServerThread(Socket socket,
                                          RSAPrivateKey priKey) {
    super("PVIDSignServerThread");
    this.socket = socket;
    this.priKey = priKey;
  }

  public void run() {
    byte [] input = new byte[512];
    byte [] output = new byte[512];

    try {
      OutputStream out = socket.getOutputStream();
      InputStream in = socket.getInputStream();

      PVIDSignProtocol pvidp = new PVIDSignProtocol();
      output = pvidp.processInput(null, priKey);
      out.write(output);
      in.read(input);

      while (input != null && doProcess) {
        output = pvidp.processInput(input, priKey);
        out.write(output);
        in.read(input);
      }
      out.close();
      in.close();
      socket.close();
    }
    catch (IOException e) { e.printStackTrace(); }
  }
}

-----------------------------------------
-----RSA Key Generation-------------------
-----------------------------------------
// RSA Key Generator Application KeyGenerator.java
package evoting.keyGenerator;

import java.security.KeyFactory;
import java.security.KeyPairGenerator;
import java.security.KeyPair;
import java.security.PublicKey;
import java.security.PrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.interfaces.RSAPrivateKey;
import java.security.NoSuchAlgorithmException;
import java.security.spec.RSAPublicKeySpec;
import java.security.spec.RSAPrivateKeySpec;
import java.math.BigInteger;
import java.io.*;

public class KeyGenerator {
  private RSAPublicKey dpublicKey = null;
  private RSAPrivateKey dprivateKey = null;
  public KeyGenerator() { }
```

```
public void func_KeyPair() {
  try {
    KeyPairGenerator keyGen =
                          KeyPairGenerator.getInstance("RSA");
    keyGen.initialize(1024);
    KeyPair keypair = keyGen.genKeyPair();

    dpublicKey = (RSAPublicKey)keypair.getPublic();
    dprivateKey = (RSAPrivateKey)keypair.getPrivate();
    //e: pubKey.getModulus());
    //n: pubKey.getPublicExponent());
    //d: dprivateKey.getPrivateExponent());
  }
  catch (NoSuchAlgorithmException nsae) {
    JOptionPane.showMessageDialog(new Frame(), e.toString());
  }
}

public RSAPublicKey getPubKey() {
  return dpublicKey;
}
public RSAPrivateKey getPriKey() {
  return dprivateKey;
}

//This method takes public exponent and modulus
//as byte arrays, reconstructs RSAPublicKey from them,
//then returns the key.
public static RSAPublicKey reconstructPubKey
                                          (byte[] e, byte[] n) {
  RSAPublicKey rsa_pub_key = null;

  try {
    KeyFactory rsa_key_fac = KeyFactory.getInstance("RSA");
    BigInteger n_num = new BigInteger (1, n);
    BigInteger e_num = new BigInteger (1, e);
    RSAPublicKeySpec rsa_keyspec =
                           new RSAPublicKeySpec(n_num, e_num);
    rsa_pub_key = (RSAPublicKey)
                       rsa_key_fac.generatePublic(rsa_keyspec);
  }
  catch (Exception ex) { ex.printStackTrace(); }

  return rsa_pub_key;
}


//this method takes private exponent and modulus
//as byte arrays, reconstructs RSAPrivateKey from them,
//then returns the key.
public static RSAPrivateKey reconstructPriKey
                                          (byte[] d, byte[] n) {
  RSAPrivateKey rsa_pri_key = null;

  try {
    KeyFactory rsa_key_fac = KeyFactory.getInstance ("RSA");
    BigInteger n_num = new BigInteger (1, n);
    BigInteger d_num = new BigInteger (1, d);
```

```
      RSAPrivateKeySpec rsa_keyspec =
                              new RSAPrivateKeySpec (n_num, d_num);
      rsa_pri_key = (RSAPrivateKey)
                        rsa_key_fac.generatePrivate (rsa_keyspec);
    }
    catch (Exception ex) { ex.printStackTrace(); }

    return rsa_pri_key;
  }
}
```

```
----------------------------------------
-----CryptoUtil class--------------------
----------------------------------------
//class CryptoUtil.java
package evoting.utils;

import java.io.IOException;
import java.math.BigInteger;
import java.security.*;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import javax.crypto.*;
import javax.crypto.spec.*;

public class CryptoUtil {

  public static byte[] RSAsign(RSAPrivateKey privateKey,
                                                  byte[] document) {
    BigInteger exponent = privateKey.getPrivateExponent();
    BigInteger modulus = privateKey.getModulus();
    BigInteger message = new BigInteger(1, document);
    BigInteger sign = message.modPow(exponent, modulus);
    return remove00(sign.toByteArray());
  }

  public static byte[] RSAunSign(RSAPublicKey publicKey,
                                                    byte[] sign) {
    BigInteger pub_EXP = publicKey.getPublicExponent();
    BigInteger pub_MOD = publicKey.getModulus();
    BigInteger sM = new BigInteger(1, sign);
    BigInteger uM = sM.modPow(pub_EXP, pub_MOD);
    return remove00(uM.toByteArray());
  }


  //This Sign method, first of all hashes the msg with
  //hashSha256 method, then sign it with the algorithm
  //SHA256withRSA. After signing, concatenate raw msg with
  //signed msg, then returns it.
  public static byte[] Sign (RSAPrivateKey pri, byte[] msg) {
    byte[] rawMsg = msg;
    byte[] hashedMsg = hashSha256 (msg);
    byte[] signedMsg = null;

    try {
      Signature sig = Signature.getInstance ("SHA256withRSA");
```

```
    sig.initSign (pri);
    sig.update (hashedMsg);
    signedMsg = sig.sign();
  }
  catch (Exception e) { e.printStackTrace(); }

  byte[] sigNedMsg = GUIUtil.concat
                            (new byte[][] {rawMsg, signedMsg});
  return sigNedMsg;
}


//This Unsign method first hashes rawMsg with hashSha256
//method. After that it verifies calculated hashed msg
//with hashed msg extracted. If verification is
//successful, returns raw msg, else returns null.
public static byte[] Unsign (RSAPublicKey pub, byte[] msg) {
  byte[] rawMsg = GUIUtil.getByteArray (msg, 0);
  byte[] hashedMsg = hashSha256 (rawMsg);
  byte[] signedMsg = GUIUtil.getByteArray (msg, 1);
  boolean verify = false;
  byte[] unsignedMsg = null;

  try {
    Signature sig = Signature.getInstance ("SHA256withRSA");
    sig.initVerify (pub);
    sig.update (hashedMsg);
    verify = sig.verify (signedMsg);
  }
  catch (Exception e) { e.printStackTrace(); }

  if (verify)
    unsignedMsg = rawMsg;
    return unsignedMsg;
}


public static byte[] randomNumber(int bytes) {
  byte [] result = new byte[bytes];
  SecureRandom SRNG = null;
  try {
    SRNG = SecureRandom.getInstance("SHA1PRNG", "SUN");
  }
  catch (Exception e) { e.printStackTrace(); }

  SRNG.setSeed(SRNG.generateSeed(bytes));
  SRNG.nextBytes(result);
  BigInteger rN = new BigInteger(1, result);
  return remove00(rN.toByteArray());
}


public static byte[] getRandomBlindingFactor() {
  return randomNumber(128);
}
```

```
public static byte[] blindMsg(byte[] msg,
                  RSAPublicKey publicKey, byte[] blindFactor) {
  BigInteger exponent = publicKey.getPublicExponent();
  BigInteger modulus = publicKey.getModulus();
  BigInteger r = new BigInteger(1, blindFactor);
  BigInteger rE = r.modPow(exponent, modulus);
  BigInteger m = new BigInteger(1, msg);
  BigInteger mrE = m.multiply(rE).mod(modulus);
  return remove00(mrE.toByteArray());
}

public static byte[] deblindMsg(byte[] blindedSign,
                  byte[] blindFactor, RSAPublicKey publicKey) {
  BigInteger modulus = publicKey.getModulus();
  BigInteger message = new BigInteger(1, blindedSign);
  BigInteger r = new BigInteger(1, blindFactor);
  BigInteger r_1 = r.modInverse(modulus);
  BigInteger m = message.multiply(r_1).mod(modulus);
  return remove00(m.toByteArray());
}

public static byte[] RSAencrypt(RSAPublicKey publicKey,
                                                byte[] ptext) {
  BigInteger pub_EXP = publicKey.getPublicExponent();
  BigInteger pub_MOD = publicKey.getModulus();
  BigInteger sM = new BigInteger(1, ptext);
  BigInteger eM = sM.modPow(pub_EXP, pub_MOD);
  return remove00(eM.toByteArray());
}

public static byte[] RSAdecrypt(
                     RSAPrivateKey privateKey, byte[] ctext) {
  BigInteger exponent = privateKey.getPrivateExponent();
  BigInteger modulus = privateKey.getModulus();
  BigInteger message = new BigInteger(1, ctext);
  BigInteger dM = message.modPow(exponent, modulus);
  return remove00(dM.toByteArray());
}


//In Encrypt method, since msg length is generally too
//long, this function first encrypts msg with DES key.
//Then it encrypts that DES key with RSAPublicKey.
//After that it concatenates those two byte[] arrays and
//returns the result.
public static byte[] Encrypt (RSAPublicKey key, byte[] msg) {
  byte[] onlyEncMsg = null;
  byte[] encKey = null;
  byte[] encMsg = null;

  try {
    KeyGenerator desGen = KeyGenerator.getInstance ("DES");
    SecretKey des = desGen.generateKey();
    SecretKeyFactory fac = SecretKeyFactory.getInstance("DES");
    DESKeySpec spec = (DESKeySpec) fac.getKeySpec (des,
                          javax.crypto.spec.DESKeySpec.class);
    byte[] rawDesKey = spec.getKey();
```

```
      Cipher c = Cipher.getInstance ("DES");
      c.init (Cipher.ENCRYPT_MODE, des);
      onlyEncMsg = c.doFinal (msg);
      Cipher d = Cipher.getInstance ("RSA");
      d.init (Cipher.ENCRYPT_MODE, key);
      encKey = d.doFinal (rawDesKey);
    }
    catch (Exception e) { e.printStackTrace(); }

    encMsg = GUIUtil.concat (new byte[][] {encKey, onlyEncMsg});

    return encMsg;
  }

  //Decrypt method decrypts encrypted byte array
  //representation of DES key (encrypted with
  //RSAPublicKey) with RSAPrivateKey and then constructs
  //a DES key. Then decrypts the encrypted msg with
  //constructed DES key.
  public static byte[] Decrypt (RSAPrivateKey key, byte[] msg) {
    byte[] rawKey = GUIUtil.getByteArray (msg, 0);
    byte[] encMsg = GUIUtil.getByteArray (msg, 1);
    byte[] decMsg = null;

    try {
      Cipher c = Cipher.getInstance ("RSA");
      c.init (Cipher.DECRYPT_MODE, key);
      byte[] rawDesKey = c.doFinal (rawKey);

      SecretKeyFactory fac = SecretKeyFactory.getInstance("DES");
      DESKeySpec newKeySpec = new DESKeySpec (rawDesKey);
      SecretKey newKey = fac.generateSecret (newKeySpec);

      Cipher d = Cipher.getInstance ("DES");
      d.init (Cipher.DECRYPT_MODE, newKey);
      decMsg = d.doFinal (encMsg);
    }
    catch (Exception e) { e.printStackTrace(); }
    return decMsg;
  }

  //hash method which hashes msg with SHA-256
  public static byte[] hashSha256 (byte[] msg) {
    byte[] hashedMsg = null;
    try {
      MessageDigest digest = MessageDigest.getInstance
                                                ("SHA-256");
      digest.reset();
      hashedMsg = digest.digest (msg);
    }
    catch (Exception e) { e.printStackTrace(); }
    return hashedMsg;
  }
}//end of source code
```

CURRICULUM VITAE

Orhan Çetinkaya received his B.S. degree in Computer Engineering and Information Science from Bilkent University, in 1997 in Turkey; he received his M.S. degree in Computer Engineering from Middle East Technical University, in 1999 in Turkey. He worked as a teaching assistant in METU (1997 - 1998) and as a researcher in Software Research & Development Center, METU (1998 - 1999). Then he worked in Havelsan Corp., Ankara, Turkey for six years as a software engineer (1999 - 2005). He has been working as a senior scientist in an international organisation in the Netherlands since 2005. His current research interests are security in electronic/Internet voting protocols, public key cryptosystems, cryptographic protocols and applied cryptography.

Refereed Journal/Conference Papers:

- O. Cetinkaya and D. Cetinkaya, "*Verification and Validation Issues in Electronic Voting", The Electronic Journal of e-Government (EJEG), Volume 5 Issue 2,* pp 117-126, Academic Conferences Limited, United Kingdom, 2007.

- O. Cetinkaya, *"Analysis of Security Requirements for Cryptographic Voting Protocols (Extended Abstract)", 4th Symposium on Requirements Engineering for Information Security (SREIS'08)*, Barcelona, Spain, 4-7 March 2008. ©IEEE

- O. Cetinkaya and A. Doganaksoy, "*A Practical Verifiable E-Voting Protocol for Large Scale Elections over a Network", In Proceedings of the 2nd International Conference on Availability, Reliability and Security (ARES'07),* Vienna, Austria, pp. 432-442, 10-13 April 2007. ©IEEE

- O. Cetinkaya and A. Doganaksoy, "*Pseudo-Voter Identity (PVID) Scheme for E-Voting Protocols*", *In Proceedings of the International Workshop on Advances in Information Security (WAIS'07) in conjunction with ARES'07,* Vienna, Austria, pp. 1190-1196, 10-13 April 2007. ©IEEE

- O. Cetinkaya and D. Cetinkaya, "*Validation and Verification Issues in E-Voting*", *In Proceedings of the 7th European Conference on E-Government (ECEG'07),* The Hague, Netherlands, pp. 63-70, 21-22 June 2007.

- O. Cetinkaya and D. Cetinkaya, "*Anonymity in E-Voting Protocols", In Proceedings of the 3rd International Conference on Global E-Security (ICGeS'07),* London, United Kingdom, pp. 137-143, 18-20 April 2007.

- O. Cetinkaya and D. Cetinkaya, "*Towards Secure E-Elections in Turkey: Requirements and Principles*", *In Proceedings of the 2nd International Workshop on Dependability and Security in e-Government (DeSeGov'07) in conjunction with ARES'07,* Vienna, Austria, pp. 903-907, 10-13 April 2007. ©IEEE

- A. Dogac, C. Beeri, A. Tumer, M. Ezbiderli, N. Tatbul, C. Icdem, G. Erus, O. Cetinkaya, N. Hamali, *"MARIFlow: A Workflow Management System for Maritime Industry", Chapter in Application of Information Technologies to the Maritime Industry*, C. Guedes Soares, J. Brodda (editors), EU/ESPRIT Program, MAREXPO Consortium, June 1999.

National Publications:

- D. Cetinkaya and O. Cetinkaya, "*E-Seçim Uygulamaları için Gereksinimler ve Tasarım İlkeleri", XI. 'Türkiye'de Internet' Konferansı (inet-tr'06),* Ankara, Turkey, 21-23 December 2006.

- O. Cetinkaya and A. Doganaksoy, "*A Practical Privacy Preserving E-Voting Protocol Using Dynamic Ballots", National Cryptology Symposium II*, Ankara, Turkey, 15-17 December 2006.

- O. Cetinkaya and A. Doganaksoy, "*Electronic Voting Protocols Based on Blind Signatures", National Cryptology Symposium I,* Ankara, Turkey, 18-20 November 2005.

M.Sc. Thesis:

- O. Cetinkaya, "A GUI and Guard Generation for Agent-based Workflow Enactment over the Internet", M.Sc. Thesis, Dept. of Computer Engineering, Middle East Technical University, Ankara, Turkey, 1999.