NATIONAL KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

GRADUATE STUDY PROGRAM
COMPUTATIONAL SCIENCE

DIPLOMA THESIS

Voter Verifiable Internet Voting Protocols

Anthi A. Orfanou

Supervisor: **Aggelos Kiayias,** Associate Professor

ATHENS

OCTOBER 2012

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΥΠΟΛΟΓΙΣΤΙΚΗ ΕΠΙΣΤΗΜΗ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πρωτόκολλα Ηλεκτρονικών Εκλογών Με Επαλήθευση Ψήφου

Ανθή Α. Ορφανού

Επιβλέπων: **Άγγελος Κιαγιάς,** Αναπληρωτής Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2012

DIPLOMA THESIS

Voter Verifiable Internet Voting Protocols

Anthi A. Orfanou

RN: M1044

SUPERVISOR: **Aggelos Kiayias**, Associate Professor

COMMITTEE: **Aggelos Kiayias**, Associate Professor
Alexis Delis, Associate Professor

OCTOBER 2012

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πρωτόκολλα Ηλεκτρονικών Εκλογών Με Επαλήθευση Ψήφου

Ανθή Α. Ορφανού

ΑΜ: M1044

ΕΠΙΒΛΕΠΩΝ: **Άγγελος Κιαγιάς**, Αναπληρωτής Καθηγητής ΕΚΠΑ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Άγγελος Κιαγιάς**, Αναπληρωτής Καθηγητής ΕΚΠΑ
Αλέξης Δελής, Αναπληρωτής Καθηγητής ΕΚΠΑ

ΟΚΤΩΒΡΙΟΣ 2012

Abstract

In this Master thesis we study the topic of remote Internet voting, which suffers inherently from two main drawbacks: the untrusted voting clients and the vote-coercion problem. We focus on the code verification voting protocols, an approach to deal with integrity issues due to untrusted voting platforms in remote Internet voting protocols. Code verification protocols use voter-dependent return codes to allow the voters verify that their ballots were cast as intended. In this thesis we study the existing code verification protocols and discuss their features and security guarantees. Moreover we propose a new protocol that achieves vote verification, i.e. it provides the voter with a receipt of her vote without the use of security codes, assuming the existence of a real-time untappable channel between the voting servers and the voter. Our protocol is flexible, allowing us to include an arbitrary number of voting servers to enhance voter privacy, overcoming the main privacy drawback of the previous approaches. Furthermore we present a natural extension of our protocol to a code verification protocol, relaxing the security requirements of the secure channel, based on the existence of two out-of-band channels that do not depend on the voter computers, established in all previously proposed protocols. In addition we adapt visual cryptography techniques to transform our construction into a visual vote verification protocol that allows the voters verify their votes visually, by overlaying seemingly random black and white images that reveal a receipt of the vote when combined.

SUBJECT AREA: Cryptography
KEYWORDS: Electronic Voting, Voter Verifiability, Voter Privacy, Vote Integrity, Untrusted Voting Platforms

Περίληψη

Σε αυτή τη διπλωματική εργασία μελετούμε το θέμα των ηλετρονικών εκλογών εξ΄ αποστάσεως μέσω Διαδυκτίου, οι οποίες είναι από τη φύση τους εκτεθειμένες σε δύο σημαντικές απειλές: τις μη αξιόπιστες πλατφόρμες υποβολής ψήφου και το πρόβλημα του εξαναγκασμού ψήφου. Εστιάζουμε στα πρωτόκολλα κωδικού επαλήθευσης, μια τεχνική αντιμετώπισης των ζητημάτων ακεραιότητας ψήφου που προκύπτουν από τη χρήση των μη αξιόπιστων προσωπικών υπολογιστών. Τα πρωτόκολλα κωδικού επαλήθευσης χρησιμοποιούν εξατομικευμένους κωδικούς επιβεβαίωσης για κάθε ψηφοφόρο, ώστε να οι χρήστες να μπορούν να επαληθεύσουν ότι η ψηφος τους καταγράφηκε αναλλοίωτη. Σε αυτη την εργασία μελετούμε τα υπάρχονται πρωτόκολλα κωδικού επαλήθευσης και αναλύουμε τα χαρακτηριστικά και τις εγγυήσεις ασφαλείας που παρέχουν. Επίσης προτείνουμε ένα νέο πρωτόκολλο επαλήθευσης ψήφου, το οποίο χρησιμοποιεί την ίδια την ψήφο ως απόδειξη καταχώρησης, χωρίς τη χρήση ενδιάμεσων κωδικών, βασιζόμενοι στην υπάρξη ενός ασφαλούς καναλιού πραγματικού χρόνου μεταξύ των διακομιστών εκλογών και του χρήστη. Η κατασκευή μας είναι ευέλικτη καθώς μας επιτρέπει να χρησιμοποιήσουμε οποιοδήποτε πλήθος διακομιστών ώστε να ενδυναμώσουμε την εγγύηση μυστικότητας ψήφου, ξεπερνώντας ένα σημαντικό πρόβλημα των υπάρχοντων λύσεων. Επιπλέον παρουσιάζουμε μια φυσική επέκταση του πρωτοκόλλου μας με χρήση κωδικών επαλήθευσης, χαλαρώνοντας τις απαιτήσεις μας για το ασφαλές κανάλι, στηριζόμενοι στην υπάρξη δύο καναλιών με ασθενέστερες εγγυήσεις ασφαλείας, στα οποία βασίζονται όλες οι προηγούμενες λύσεις. Τέλος, υιοθετούμε τεχνικές οπτικής κρυπτογραφίας για να μετασχηματίσουμε την κατασκευή μας σε ένα πρωτόκολλο οπτικής επαλήθευσης ψήφου, το οποίο επιτρέπει στο χρήστη να επιβεβαιώσει την ψήφο του συνδιάζοντας ασπρόμαυρες, φαινομενικά τυχαίες, εικόνες οι οποίες αποκαλύπτουν την απόδειξη ψήφου όταν συνδιάζονται.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Κρυπτογραφία
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Ηλεκτρονικές Εκλογές, Επιβεβαίωση Ψήφου, Μυστικότητα Ψήφου, Ακαιρεότητα Ψήφου, Μη Αξιόπιστες Πλατφόρμες Υποβολής Ψήφου

Στη μνήμη της γιαγιάς Καλλιόπης

Ευχαριστίες

Κατ΄ αρχήν θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, κύριο Άγγελο Κιαγιά για τη βοήθεια, την εμπιστοσύνη και την καθοδήγησή του κατά τη διάρκεια της διπλωματικής μου και του μεταπτυχιακού προγράμματος, καθώς και για την υποστήριξή του για τα μελλοντικά μου βήματα. Το μάθημα της κρυπτογραφίας και η οργάνωση του crypto.sec group του τμήματος έγιναν η αφορμή να γνωρίσω το πολύ ενδιαφέρον αντικείμενο της κρυπτογραφίας στο οποίο επιθυμώ να δουλέψω ερευνητικά στο μέλλον. Ευχαριστώ ιδιαιτέρως τον κύριο καθηγητή Αλέξη Δελή που συμφώνησε να είναι ο εξεταστής αυτής της εργασίας καθώς τον κύριο καθηγητή Ηλία Κουτσουπιά, που με υποστήριξαν στη συνέχιση των σπουδών μου. Πάνω απ'ολα ευχαριστώ τη μητέρα μου, Μαρία, που είναι πάντα δίπλα μου καθώς και το φίλο μου, Σταύρο Γερακάρη, για την υποστήριξη του και τη συμβολή του στην ολοκλήρωση αυτής της εργασίας.

Contents

1 Introduction to Electronic Voting **16**

2 Cryptographic Background and Tools **21**

2.1 Hash Functions and Random Oracles 22

2.1.1 Hash functions 22

2.1.2 Pseudo-random functions 23

2.1.3 Random oracles 23

2.2 Commitments 24

2.2.1 Pedersen's commitment scheme 25

2.3 Signatures 26

2.4 Zero-Knowledge Proofs of Knowledge 26

2.4.1 The Schnorr protocol 28

2.4.2 The Chaum-Pedersen protocol 28

2.4.3 The disjunction of zero-knowledge proofs 28

2.4.4 The conjunction of zero-knowledge proofs 29

2.4.5 Range and set membership proofs 30

2.4.6 Non-interactive zero-knowledge proofs 30

2.5 Public Key Encryption 31

2.5.1 The ElGamal crypto-system 32

2.6 Oblivious Transfer 33

2.6.1 The AIR 1-out-of-N oblivious transfer 34
2.6.2 The proxy oblivious transfer 35
2.7 Secret Sharing 36
2.7.1 A n-out-of-n scheme 36
2.7.2 Visual cryptography 37
2.8 The Communication channels 39

3 The Untrusted Platform Problem 40
3.1 Code Voting 41
3.2 Code Verification Voting 43
3.3 The Proxy Oblivious Transfer Approach 44
3.3.1 POT E-Voting 44
3.3.2 Security guarantees and weaknesses 48
3.4 The Pseudo-random Composition Approach 49
3.4.1 The shared-key E-voting 50
3.4.2 Vote encoding and tallying improvements 54
3.4.3 Security guarantees and weaknesses 56
3.4.4 Avoiding Coalitions 57

4 A New Vote Verification Protocol 62
4.1 The vote verification protocol 64
4.1.1 The main idea 66
4.1.2 Commitments' announcement 67
4.1.3 Range proof 69
4.1.4 Adding more voting servers 70
4.1.5 Security guarantees and performance 70

4.1.6 Complexity analysis 76
4.2 Extension to code verification 77
4.2.1 The trusted channels and the security codes 77
4.2.2 First code verification protocol 78
4.2.2.1 Security guarantees 79
4.2.3 Second code verification protocol 79
4.2.3.1 Security guarantees and overhead 80
4.3 Extension to visual vote verification 81
4.3.1 Previous work 81
4.3.2 The visual vote encoding 84
4.3.3 The ideal security model and our guarantees 86
4.3.4 Our visual sharing shape descriptor construction 86
4.3.5 Vote submission and verification 87
4.3.6 Security and complexity properties 89

Conclusion **95**

Abbreviations **96**

Bibliography **97**

List of Algorithms

1 Pedersen's Commitment 25

2 The Schnorr Protocol 28

3 The Chaum-Pedersen Protocol 29

4 ZKP Disjunction 29

5 ZKP Conjunction 30

6 The ElGamal Crypto-system 32

7 The AIR $(1, N)$-OT 34

8 (Weak) Proxy Oblivious Transfer 45

9 ZKP encryptions with equal plain-texts 46

10 POT Code-Verification E-Voting: Setup 46

11 POT Code-Verification E-Voting: Vote Submission 47

12 Shared-Key Code Verification E-Voting - Setup Phase 51

13 Shared-Key Code Verification E-Voting - Vote Submission 52

14 Shared-Key Code Verification E-Voting - Tallying 56

15 PRF-Composition Enhanced Code Verification 59

16 ZKP of consistent encryptions and commitments 60

17 The full ZKP for the vote-verification protocol 72

18 The Splitting Vote-Verification Protocol 73

19 Proof of valid visual votes 91

20 Visual Vote Verification 92

List of Figures

1.1 The structure of an Internet Voting scheme 18

2.1 Proxy Oblivious Transfer 36

2.2 2-out-of-2 visual secret sharing 38

3.1 Comparison of different code sheet's types 43

3.2 The POT E-voting 48

3.3 Pseudo-random composition shared-key E-Voting 53

4.1 ZKP Equality of committed values 67

4.2 Proof $\pi_{sum} = \text{ZPK}(x, y, r_x, r_y : \ C_x = g^x h_1^{r_x} \wedge C_y = g^y h_2^{r_y} \wedge (y = x \vee y = x + m))$ 68

4.3 Range proof in Pedersen commitments 71

4.4 Overview of the splitting-vote verification protocol 74

4.5 Overview of the code verification protocol 82

4.6 Chaum's visual receipts 83

4.7 The visual building block 85

4.8 8-bit and 16-bit subdivisions of the building block 85

4.9 Example of encoding of 5 candidates using the same fixed share v_1. 87

4.10 All valid shares of Λ_8 that are sufficient for encoding up to 5 candidates. . . . 88

4.11 Schnorr OR ZK Proof 90

4.12 Bitwise OR combinations 90

List of Tables

3.1 The PRF composition: Knowledge required by the entities 53

3.2 The PRF composition: Proof of messenger's security 57

3.3 Comparison of the code verification protocols 61

Chapter 1

Introduction to Electronic Voting

During the past decades with the massive explosion of electronic devices the vast majority of everyday actions have been improved and made simpler through their electronic equivalent. The Internet, playing the most significant role towards this direction, provided us with numerous easily accessible and user-friendly services such as electronic communication, electronic commerce, electronic banking and many more. Consequently, electronic voting consists one of the most important electronic services, making people able to express their preferences in an easy and confidential way, encouraging them to participate actively and more massively in the elections procedure.

In conventional elections a voting protocol requires the collaboration of voters, that submit paper votes in sealed envelopes in a ballot box, and voting authorities which are responsible to collect and open the submitted ballots after the end of the elections, count the votes and specify the outcome. Similarly, in an electronic voting protocol all the above parts remain essential, but the authorities are no longer human beings. Instead, computers are assigned those tasks and we expect them to complete them in an effective and trustworthy manner. Electronic voting consists of two different main approaches: Internet Voting and Kiosk Voting [12]. The first lets the voters participate in the elections remotely, through the Internet, by using their personal computers or mobile devices, while the second requires them to cast their votes in dedicated fixed locations, like public libraries or schools, by using specially designed voting machines.

Internet voting has the advantage of making the voting process more convenient, as it supports mobility, allowing people to vote remotely and thus encouraging their participation in

the elections and resulting in a more representative and fair democratic outcome. Nonetheless, while Internet voting has several usability advantages, it is also prone to several attacks regarding vote integrity, due to the vulnerability of personal computers, which may be affected by malicious software, like trojans, worms and viruses, as well as to the possible threat of voters' coercion within their home environment. On the other hand, Kiosk Voting, overcomes those two undesired situations, but lacks in convenience and usability, requiring the voters to act in a similar way to conventional elections. Both cases share similar advantages, as electronically submitted votes support quick tallying, counting the votes almost instantly, avoid human mistakes and thus build trust among the voters and the elections' authorities.

In this thesis we are going to focus our attention in Internet Voting, which significantly supports usability and easiness from the user's perspective. However as stated before, Internet Voting should fulfill additional requirements in order to be secure enough to be implemented. These concerns regarding Internet Voting have been the main reason that it has not seen wide deployment despite its advantages. Thus apart from the established properties of an electronic voting scheme, Internet Voting should meet additional criteria to guarantee security against untrusted voting platforms and coercers. Hence, regarding untrusted platforms, a reliable Internet Voting system should allow the voters to cast secret ballots and verify their recording by the system, even in the case their devices leak information or do not function appropriately. Regarding the coercion problem, no coercer or vote buyer that takes advantage of the open environment of Internet voting should be able to manipulate the elections.

Of course, none of the previous approaches would be implemented without the essential contribution of cryptography. Thus cryptography provides us the necessary primitives and tools to built robust, efficient and secure voting protocols that guarantee correct outcomes, preserving anonymity and integrity.

In Internet voting, the voters are able to submit their votes through their personal computers or other personal electronic devices, without using any additional specialized hardware. Votes should be encrypted with an efficient encryption scheme in order to be transmitted and submitted to the electronic ballot box, without revealing information about the vote. When the election period is over the valid ballots are forwarded to the server that is responsible for decrypting and tallying them, defining the election's outcome.

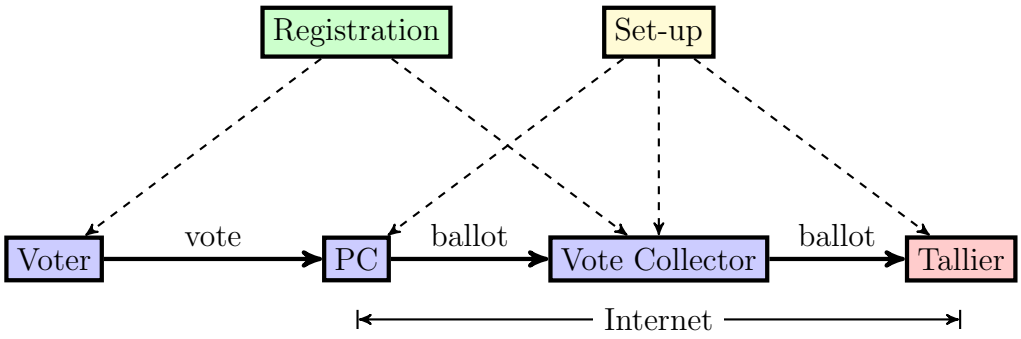

Figure 1.1: The structure of an Internet Voting scheme

Definition 1.1. Internet Voting Protocol. An Internet Voting Protocol is a communication protocol between a set of voters and the authorities running the elections. The main participating entities of an Internet voting protocol are:

1. Voters, people, that cast their votes.
2. Personal computers that encrypt and forward the votes submitted by the users.
3. Vote Collectors that receive and store the submitted ciphertext.
4. Talliers that decrypt the ciphertexts of the votes and specify the outcome.

A typical Internet voting protocol consists of the the following phases:

1. Elections Set-up: The initial phase where the system parameters are chosen and key generation takes place.
2. Voter Registration: The phase where eligible citizens receive their voting credentials, taking any particular actions needed.
3. Vote Submission: The main voting phase where eligible voters cast their votes.
4. Vote Tallying: The last part of the elections where votes are gathered and counted.

In order to create a trustworthy electronic voting system it is essential to clarify the properties it should have. Following, we briefly discuss the desired properties of a reliable voting protocol, which have been discussed in [23]. Although we would require an ideal voting protocol which would fulfill all the criteria presented bellow, in practice it is hard to create such a protocol, as some of the targets conflict.

Authentication and Eligibility. Only authorized voters should be allowed to submit votes, prohibiting unauthorized voters to cast ballots and allowing all eligible voters to participate. Authentication is usually achieved by using certain voting credentials to identify the voters.

Integrity. No one should be able to alter the submitted votes, and if this is violated it should be detected by the voters or the system.

Privacy. No one should be able to determined how any voter voted. Privacy is usually achieved by distributing a task to a number of entities with conflicted interests that are not likely to collaborate.

Vote Encoding Verifiability. The voters should be able to determine that their ballots were cast as intended. The voting system should be able to prove to the voters that their votes were successfully received and recorded.

Vote Tallying Verifiability. The voters should be able to determine that their ballots were counted as cast. The voting system should be able to prove to the voters that their votes were successfully included in the final outcome.

Universal Verifiability. Anyone, including passive observers, should be able to verify that all valid cast votes have been included in the final tally. This is the strongest verifiability requirement were anyone can test the validity of the elections outcome. A system that supports voter verifiability and universal verifiability is an End-to-End verifiable voting system.

Uniqueness. No voter should be allowed to submit more than one valid votes. This may be achieved by allowing a voter to vote once or by ensuring that in case of re-voting only the last vote will be included in the final tally. The last approach is usually employed as a means against coercion.

Coercion-resistance and Receipt-Freeness. The voters should not be able to prove that they voted for a particular candidate and if they voted at all and they cannot reveal their voting credentials to an attacker who successfully submits ballots on behalf of them. Receipt-freeness is a weaker requirement for coercion-resistance stating that no voter should be able to prove that she voted in a particular manner, even if she wishes to do so.

Fairness. No one should be able to learn partial results before the end of the elections.

Robustness. The voting system should be able to recover from the faulty behavior of any reasonably sized coalition of its entities. Robustness is typically achieved by distributing a task over several entities and ensure that if the well-functioning entities exceed a threshold the scheme will not fail.

Organization of this thesis

In the following chapters we discuss Internet voting protocols and how the address the requirements we ask for. In chapter 2 we present the essential cryptographic background and tools that will be exploited in building and analyzing voting protocols and studying their security. Chapter 3 focuses on the untrusted platform problem, summarizing the existing solutions that preserve vote integrity, paying particular attention in the category of code verification protocols. Chapter 4 presents our contribution against untrusted platforms, where we built a vote-verification internet voting protocol that guarantees integrity and enhanced vote secrecy, along with its adaptations in the code verification setting and in visual cryptography setting.

This thesis focuses on the vote submission phase, the online phase of the elections and considers the key generation and the tallying phases separate offline phases, which are solved by using existing protocols that we do not analyze.

Chapter 2

Cryptographic Background and Tools

In this chapter we present the necessary background and the basic cryptographic tools that are used to design secure e-voting protocols. We begin our discussion by introducing the most import hard problems, on which the security of various cryptographic protocols is relying and proceed with essential cryptographic constructions that will be used in several voting protocols.

Definition 2.1. The Discrete Logarithm Problem (DL). Let G be a cyclic group of order q and $\langle g \rangle$ be the group generator. The discrete logarithm problem is to find an integer $x \in Z_q$ such that $g^x = y \ mod \ p$, for an element $y \in G$.

Although we have no proof that the above problem is computationally hard, if the group is chosen carefully and appropriately, the solution requires a large number of steps. Thus many cryptographic protocols base and prove their security on the assumption of the hardness of the DL problem. However, many protocols are based on a related problem, which is proved to be no harder than the DL problem: the Decisional Diffie-Hellman Problem which is present below.

Definition 2.2. The Decisional Diffie-Hellman Problem (DDH) Let G be a cyclic group of prime order q and $\langle g \rangle$ the group generator. Given the values g^a, g^b, g^c, with $a, b, c \leftarrow Z_q$, decide if $c = ab$ or $c \leftarrow Z_q$.

Intuitively the DDH problem states that we cannot distinguish between tuples of the form (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c). Thus the formal security proofs of various protocols consist of reductions to DDH problem, showing that the failure of the protocol would imply the existence of a polynomial time algorithm for the DDH problem.

There are also other alternative formulations of the DDH problem which are proved to be equivalent. Thus we may encounter the DDH problem with the following forms.

Definition 2.3. The Decisional Diffie-Hellman Problem (DDH). Alternative Definitions:

(1a) Given $(g_1, g_2) \in G \times G$ decide if (x_1, x_2) was sampled uniformly from the powers of (g_1, g_2), i.e. there is an $s \in Z_q$ $(0 \leq s \leq q)$ such that $(x_1, x_2) = (g_1^s, g_2^s)$, or uniformly from $G \times G$.

(1b) Given $(g_1, ..., g_n) \in G^n$ decide if $(x_1, ..., x_n)$ was sampled uniformly from the powers of $(g_1, ..., g_n)$ or uniformly from G^n.

2.1 Hash Functions and Random Oracles

2.1.1 Hash functions

A key element in cryptographic protocols are the hash functions, which have numerous applications in commitment schemes and guaranteeing data integrity. A hash function is a mapping that compress an input, i.e. takes as input a message of arbitrary length and outputs an element of bounded-size.

Definition 2.4. Hash Function. A hash function is a pair of probabilistic polynomial time algorithms (Gen, H) such that on security parameter k:

1. Gen is a probabilistic algorithms which takes as input k and outputs a secret key s.

2. There is a polynomial l such that when H takes as input the secret key s and a string $x \in \{0,1\}^*$ it outputs a string $H_s(x) \in \{0,1\}^{l(k)}$

If H_s is defined only for inputs $x \in \{0,1\}^{l'(n)}$ with $l'(n) > l(n)$, then (Gen, H) is a fixed length hash function for inputs of length $l'(n)$.

An ideal hash function should fulfill the following requirements, which are ordered according to the level of the achieved security.

- Non-invertibility (First Pre-mage resistance): Given the secret key s and a hash $y = H_s(x)$ and a hash function H, it is infeasible for any probabilistic polynomial time algorithm to find a message m, such that $y = H_s(m)$.

- Weak collision-resistance (Second Pre-image resistance): Given the secret key s and a message m_1, it should be infeasible for any probabilistic polynomial algorithm to find another message $m_2 \neq m_1$ such that $H_s(m_1) = H_s(m_2)$.
- Strong Collision-resistance: Given the secret key s it should be infeasible for any probabilistic polynomial algorithm to find two messages $m_1 \neq m_2$, such that $H_s(m_1) = H_s(m_2)$

2.1.2 Pseudo-random functions

Pseudo-random functions are functions which cannot be distinguished from truly random functions by any efficient procedure which can get the value of the function at arguments of its choice, in a black box manner.

Pseudo-random functions can be described in terms of keyed functions. A keyed function is a two-input function $F : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ with the first input being a secret key and the second being the normal input. As long as a random key is chosen and fixed we may denote the function as F_k mapping $\{0,1\}^* \rightarrow \{0,1\}^*$. Hence pseudo-randomness is expressed in terms of indistinguishability between F_k from the 2^n functions generated for $k \leftarrow \{0,1\}^n$, and f being randomly chosen from the set of 2^{n2^n} mappings from $\{0,1\}^n$ to $\{0,1\}^n$. Pseudo-randomness clearly depends of the secrecy of the selected k, stating that once the key is revealed to an adversary then he can easily distinguish between a random and pseudo-random function.

Definition 2.5. Pseudo-random Function. Let $F : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ be an efficient, keyed function. We define F to be a pseudo-random function if for all probabilistic polynomial time algorithms A having oracle access to the functions F_k, f it holds that $|Pr[A^{F_k}(1^n) = 1] - Pr[A^f(1^n) = 1]| \leq negl(n)$, where $k \leftarrow \{0,1\}^k$ and f is chosen uniformly at random from the set of functions mapping n-bit strings to n-bit strings.

2.1.3 Random oracles

The random oracle model is based on the existence of a public, randomly chosen function H that can be evaluated when querying an oracle. The oracle is treated as a black box, which should be consistent in the sense that when the oracle outputs a value y for a query x, then it should always give the same answer when x is asked. Moreover, given the input x, the value of $H(x)$ should be completely unpredictable, unless someone has already seen $H(x)$.

More precisely a random oracle is defined as a mapping from $\{0,1\}^*$ to $\{0,1\}^\infty$, chosen by selecting each bit of $H(x)$ independently and uniformly. As they lack any concrete structure, they are very useful in security proofs, helping as view hash functions in an abstract way. Thus they provide the background to prove various cryptographic protocols secure, under the so-called *random oracle model*, where the adversary views some functions as random oracles. However, as there is no indication that random oracles can truly exist, for practical implementations they are instantiated by an appropriate hash function.

2.2 Commitments

A commitment scheme enables an entity to commit to a secret value so that he can no longer change his mind. This is done by publicly announcing a commitment value which can be later verified. There are two basic properties that should be satisfied by commitment schemes, called the hiding property and the binding property.

- Hiding: This property states that it should be hard for anyone to gain any knowledge about the secret value, unless the committer opens the commitment.
  - Unconditional hiding: The commitment values should be statistically indistinguishable, that is for any two values s_0, s_1 the statical distance between corresponding commitments c_0, c_1 should be negligible.
  - Computational hiding: No polynomial time algorithm A can distinguish among the commitments c_0, c_1 corresponding to two different secret values s_0, s_1, that is $|prob[A(c_0) = 1] - prob[A(c_1) = 1]$ is negligible.
- Binding: This property states that it should be hard for the committer to alter the secret value when the commitment is made public.
  - Unconditional binding: No computationally unbounded committer can open a commitment for a different value than the one for which he committed.
  - Computational binding: No computationally bounded committer can open a commitment for a different value than the one for which he committed.

No commitment scheme can achieve simultaneously unconditional hiding and binding.

Definition 2.6. Commitment Scheme A commitment scheme consists of a triple of algorithms $\langle Gen, Com, Open \rangle$ where:

1. The randomized $Gen(1^k)$ algorithm takes as input the security parameter k and outputs public key pk.

2. The randomized $Com_{pk}(v, r)$ algorithm takes as input the secrete value v and a randomly generated value r and outputs the commitment/de-commitment value pair (c, d).

3. The deterministic $Open(c, d, v)$ algorithm takes as input the commitment value c and the de-commitment value d and the secret v and checks that $Open(c, d) = v$.

2.2.1 Pedersen's commitment scheme

The commitment scheme described in this section is due to T. Pedersen and is based on the hardness of the discrete logarithm [15]. The scheme works on a finite cyclic subgroup G of prime order q, generated by $\langle g \rangle$.

Scheme 1 Pedersen's Commitment

1. $Gen(1^k)$: $a \leftarrow Z_q$
   Set $pk = g^a$

2. $Com^r_{pk}(s)$: $r \leftarrow Z_q$
   Set $c = g^s \cdot pk^r$
   Output (c, r)

3. $Open_{pk}(s, c, r)$: Verify that $c = Com_{pk}(s, r)$

The above scheme reveals no information about the secret value, being unconditionally hiding, since given an announced value c every secret value s is equally likely to be the commitment's value. Moreover the scheme prevents the committer from lying, being computationally binding, by revealing a value different from s, unless he is capable of computing discrete logarithms. Specifically, if the committer can find values $s \neq s' \in Z_q$ such that $Com^r(s) = Com^{r'}(s')$, then it should hold that $r \neq r'$ and thus he can efficiently compute $a = log_g pk = \frac{s-s'}{r-r'}$ mod q.

Pedersen commitments are linearly homomorphic satisfying the following properties:

- $Com_h^{r \leftarrow Z_q}(x)^a = Com_h^{ar \leftarrow Z_q}(ax)$
- $Com_h^{r \leftarrow Z_q}(x) Com_h^{r' \leftarrow Z_q}(x') = Com_h^{r+r' \leftarrow Z_q}(x + x')$

2.3 Signatures

Digital signatures are used for public key message authentication. A signature scheme is a triple of algorithms $\langle Gen, Sign, Ver \rangle$, where Gen is a randomized key-generator algorithm, $Sign$ is a signing algorithm and Ver is a verification algorithm.

- $Gen(1^k)$: On input the security parameter k the generation algorithm outputs a public/secret key pair (pk, sk) with the public key set as a signature verification key and the secret key working as a signing key.
- $Sign_{sk}(m)$: On input a signing key sk and a message m the signing algorithm outputs a digital signature σ of m.
- $Ver_{pk}(m, \sigma)$: On input the verification key pk, a message m and a signature σ the verification algorithm accepts or rejects the signature as valid.

The security model of digital signatures is the *unforgeability against chosen message attack* (CMA). In this setting we face an active adversary who is allowed to ask for signatures of multiple messages to the signing oracle. The CMA security requires a probabilistic polynomial adversary to be unable to produce a valid signature on message of his chose that he has not queried to the oracle.

Definition 2.7. Existential Unforgeability against Chosen Message Attack (EUF-CMA). On a security parameter k, for every probabilistic polynomial time adversary A it holds that $Prob[Ver(m^*, \sigma^*) = \text{accept} \mid (sk, vk) \leftarrow Gen(1^k), (m^*, \sigma^*) \leftarrow A^{Sign_{sk}}(vk, m)] \leq negl(k)$ where A cannot ask the oracle to sign m^*.

2.4 Zero-Knowledge Proofs of Knowledge

A zero-knowledge argument is an argument used to convince a player about the validity of an argument without leaking any information out of the conversation.

Zero-knowledge proofs are essential cryptographic tools having many application in cryptographic protocols. A zero-knowledge proof is a communication protocol between two players which enables one party to convince the other party of the validity of a statement without revealing any information. We call the entities that participate in the protocol prover P and verifier V. The prover possesses some knowledge about a statement x and wants to prove this to the verifier. Let w be a witness, which indicates that the prover indeed possesses the knowledge he claims to have, but is not willing to reveal. Both parties should know a polynomial-time predicate R which can efficiently compute $R(x, y)$ and test that (x, w) is a valid pair, i.e. that w is a valid witness for the statement x. We present bellow the formal definition of the protocol.

Definition 2.8. Zero-Knowledge Proof of Knowledge (ZKP). Let P and V be a pair of interactive programs and L be a binary language. A ZKP interactive protocol, is a three-message communication protocol, executed on common input x and private inputs w for P and z for V, if it satisfies the three following properties:

1. **Completeness:** If $x \in L$ and $R(x, w) = 1$ for some witness w then the verifier accepts with overwhelming probability, for every challenge z he choses.

2. **Soundness:** For any polynomial-time program P^*, let $p = Prob[(P^*, V) - accepts]$. Then there exists an efficient knowledge extractor K, which is able to produce a valid witness w' for the statement x with probability $p' = Prob[K(x, z) = w' | R(x, w') = 1]$. It must hold that if p is non-negligible, then so is p'.

3. **Statistical Zero-Knowledge:** For any polynomial-time program V^* there is an efficient simulator S, such that for all valid pairs (x, w), the output of the simulator on input x, z is statistically indistinguishable from the output of the protocol run by P and V^*, for all strings z.

Informally, completeness guarantees that if both the prover and the verifier execute the protocol honestly the verifier must always accept. The soundness property states that if the prover is cheating then the extractor can gain some knowledge from him. Finally, the zero-knowledge property states that if a dishonest verifier can extract useful information from an honest prover, then he must be able to do the same with the simulator. Thus, as the simulator does not know anything about the witness, it is implied that the prover does not reveal any knowledge.

Usually we consider 3-move zero knowledge protocols with the prover initiating the communication by announcing a committing value and terminating the communication by sending

the reply (step 1 and 3), and a single random challenge value from the verifier (step 2). We call these protocols Σ protocols. The most important Σ protocol is the Schnorr Protocol of knowledge of a discrete logarithm, presented in the following section.

2.4.1 The Schnorr protocol

This ZKP protocol forks over a cyclic group G, with group generator $\langle g \rangle$, of order q. The prover P wants to prove the knowledge of a solution to the discrete logarithm problem, knowing a value $w \in Z_q$ such that $h = g^w$, for some $h \in G$, and the verifier should confirm that $w = log_g h$, knowing p, q, g and h.

Protocol 2 The Schnorr Protocol

- **Commitment:** P chooses $t \leftarrow Z_q$ and sends $y = g^t$ to V
- **Challenge:** V chooses challenge $c \leftarrow Z_q$ and sends c to P
- **Reply:** P computes $s = t + wc$ mod q and sends s to V. V accepts if $g^s = yh^c$ mod p.

The Schnorr Protocol satisfies completeness as well as special soundness, which holds if we are able to compute the secret information from two valid runs with identical commitment phases. In addition the protocol satisfies honest-verifier zero-knowledge, that is, zero-knowledge as long as we require the verifier to execute the protocol faithfully, while he is allowed to make additional computations.

2.4.2 The Chaum-Pedersen protocol

The Chaum-Pedersen protocol is a proof of equation of two discrete logarithms, i.e. proving the relation $log_g x = log_h y$. In fact the protocol is a conjunctive version of the Schnorr protocol, where we prove both knowledge of the discrete logarithms as well as the equality relation they satisfy. In this protocol the prover wishes to show the possession of an element $a \in Z_q$ such that $x = g^a$ and $y = h^a$. Similar to the Schnorr protocol, the protocol satisfies completeness, special soundness and honest-verifier zero-knowledge.

2.4.3 The disjunction of zero-knowledge proofs

There are various scenarios where a prover is required to prove the knowledge of a valid witness for one out of two or more statements, without disclosing which of them he possesses. In this

Protocol 3 The Chaum-Pedersen Protocol

- P chooses $w \leftarrow Z_q$, set $(A, B) \leftarrow (g^w, h^w)$ and sends (A, B) to V
- V chooses challenge $c \leftarrow Z_q$ and sends c to P
- P computes $r = w + ac$ mod q and sends r to V. V accepts if $g^r = Ax^c$ mod p and $h^r = By^c$ mod p.

case, the prover has to reply to the verifier's challenge by modifying his messages accordingly. Combining zero knowledge proofs in an "OR" manner can be achieved by using a single challenge, as shown in [17]. Below we present the disjunction of two zero-knowledge proofs for the discrete logarithm problem. We assume that the prover knows a valid witness for h_1 and wants to prove the knowledge of one of the discrete logarithms of $h_1 = g^{x_1}, h_2 = g^{x_2}$. The trick used for disjunction of proofs is to produce two accepting arguments, one for the real witness and a simulated proof for the other part. For the simulated part, the prover pre-selects a challenge and modifies the real part's challenge appropriately.

Protocol 4 ZKP Disjunction

1. The prover randomly selects $s_2, c_2 \leftarrow Z_q$ and sets $y_1 = g^{t_1}$ and $y_2 = g^{s_2} h_2^{-c_2}$. Then he sends to the verifier the values y_1, y_2.
2. The verifier randomly selects $c \leftarrow Z_q$ and sends it to the prover.
3. The prover computes $c_1 = c - c_2$ mod q, $s_1 = t_1 + c_1 x_1$ mod q and sends c_1, c_1, s_1, s_2 to the verifier.
4. The verifier tests whether $c = c_1 + c_2$ mod q, $g^{s_1} = y_1 h_1^{c_1}$ and $g^{s_2} = y_2 h_2^{c_2}$ and accepts/rejects accordingly.

2.4.4 The conjunction of zero-knowledge proofs

Similarly to the previous section, there are cases where the prover needs to prove knowledge of multiple statements simultaneously. Thus there is need to design a zero-knowledge protocol that proves knowledge of all statements, while responding to a single challenge of the verifier. We present the conjunction of two proofs of knowledge of two discrete logarithms x_1, x_2 such that $h_1 = g^{x_1}$ and $h_2 = g^{x_2}$.

Protocol 5 ZKP Conjunction

1. The prover randomly selects $t_1, t_2 \leftarrow Z_q$ and sets $y_1 = g^{t_1}$ and $y_2 = g^{t_2}$. Then he sends the values y_1, y_2 to the verifier.
2. The verifier randomly selects $c \leftarrow Z_q$ and sends it to the prover.
3. The prover computes $s_1 = t_1 + cx_1 \bmod q$, $s_2 = t_2 + cx_2 \bmod q$ and sends them to the verifier.
4. The verifier checks whether $g^{s_1} = y_1 h_1^c$ and $g^{s_2} = y_2 h_2^c$ and accepts/rejects accordingly.

2.4.5 Range and set membership proofs

A range proof is a proof of knowledge that a committed number lies in an arbitrary integer interval $[a, b]$. Range proofs are special case of set membership proofs where the prover shows that a committed number belongs in any set of values Φ.

Definition 2.9. Proof of Membership Let $\langle Gen, Com, Open \rangle$ be a string commitment scheme. Given a commitment c a set membership proof is a proof of knowledge for the statement $PK(\mu, \rho \mid c = Com(\mu, \rho) \wedge \mu \in \Phi)$.

The most known range proof for a range of the form $[0, 2^k - 1]$ has the prover commit to all k-bits of the committed number μ and then prove that each commitment hides either a 0 or an 1 and that the committed bits are indeed the bits of μ. The proof requires $\Theta(k)$ single bit proofs. Using a homomorphic commitment scheme generalizes the proof in a range of the form $[a, 2^k - 1 + a]$ by substituting the commitment c with the value c/g^a. The proof can be extended to an arbitrary range $[0, b]$ by proving the AND composition of the statements $\mu \in [0, 2^k - 1]$ and $\mu \in [b - (2^k - 1), b]$.

2.4.6 Non-interactive zero-knowledge proofs

An arbitrary Σ-protocol can be made non-interactive in the random oracle model, by using the Fiat-Shamir heuristic [16]. Σ protocols are public coin protocols, i.e. the verifier sends a single random value. In order to make zero knowledge proofs non interactive the prover should be apple to run the whole protocol himself, producing randomness, and then let anyone verify its validity. Thus the verifier in a non interactive zero knowledge proof not only needs to check the validity tests, but also needs to make sure that the prover made the initial commitment before

the challenge value was selected.

The Fiat-Shamir heuristic uses a cryptographic hash function H, $H : G \times G \rightarrow Z_q$, which takes as input the public keys of the system as well as the commitment value announced by the prover to produce the challenge value as $c = H(Com, Keys)$. Hence the non-invertibility property of a secure hash function implies the validity of the proof. Usually H is treated as a random oracle, guaranteeing invertibility.

2.5 Public Key Encryption

A public key crypto-system consists of three algorithms $\Pi = \langle Gen, Enc, Dec \rangle$.

- The algorithm Gen is a randomized key generation algorithm, which on input 1^k, k being the security parameter, outputs a secret key/public key pair (s_k, p_k)
- The algorithm Enc is a randomized encryption algorithm which given the input message m and the public key p_k outputs a ciphertext c, $c = Enc_{p_k}(m)$
- The algorithm Dec is a deterministic decryption algorithm that, owing the secret key s_k, on input c outputes a message m', $Dec_{s_k}(C) = m'$

For every valid triple of public key crypto-system algorithms it must hold that $Dec_{sk}(Enc_{p_k}(m)) = m$.

The security model for public key encryption is the *security against chosen plaintext attacks* (CPA security). In this setting we face an active adversary who is allowed to ask for encryptions of multiple messages to the encryption oracle. The CPA security requires the adversary to be unable to distinguish with non-negligible probability among the encryptions of two arbitrary messages m_0, m_1, even when he is given access to the encryption oracle. We note that only randomized crypto-systems can be be proved secure under this model, as otherwise it would be trivial for the adversary to decide which message is encrypted.

Definition 2.10. IND-CPA Security

Consider the following game:

1. $(s_k, p_k) \leftarrow Gen(1^k)$.

2. A is given p_k and access to the encryption algorithm Enc_{p_k}, viewed as an encryption oracle. Then he outputs two plaintexts $m_0 \neq m_1$.

3. A random bit $b \in \{0, 1\}$ is chosen and message m_b is encrypted. Let $c = Enc_{p_k}(m_b)$ which is given to the adversary.

4. A accesses the encryption oracle and outputs a bit b'. He wins if $b = b'$.

A public-key crypto-system is IND-CPA secure if for all probabilistic polynomial times adversaries A it holds that $Prob[A \text{ wins}] \leq \frac{1}{2} + negl(k)$.

2.5.1 The ElGamal crypto-system

Probably the most studied public key crypto-system, that has numerous applications in modern cryptographic protocols, is the ElGamal crypto-system presented in the this section.

The ElGamal crypto-system is an CPA-secure crypto-system. In the setting we discuss the ElGamal crypto-system is based on the hardness of the DDH problem and works over a cyclic group G of prime order q of Z_p, p and q being large primes with $q|p-1$, with $\langle g \rangle$ being the group generator. The triple of algorithms of the encryption scheme follows.

Protocol 6 The ElGamal Crypto-system

- **Key Generation:** $Gen(1^k)$: On input the security parameter k choose the Group $\langle G, q, g \rangle$. Then select the public/secret key pair by choosing $s_k \leftarrow Z_q$ and letting $p_k = g^{s_k} mod\ p$. The public key is a tuple $\langle G, q, g, p_k \rangle$ and the secret a tuple $\langle G, q, g, s_k \rangle$.

- **Encryption of plain text** m**:** $Enc_{p_k}^{r \leftarrow Z_q}(m)$: On input the public key tuple and a message $m \in G$ choose $r \leftarrow Z_q$ and output the cipher text $Enc_{p_k}(m) = \langle m \cdot p_k^r, g^r \rangle$.

- **Decryption of ciphertext:** $Dec_{s_k}(\langle C_1, C_2 \rangle)$: On input the private key tuple and the cipher text pair $\langle C_1, C_2 \rangle$ output $C_1 / C_2^{s_k} mod\ p$.

To show the correctness of the above protocol, we see that on a valid encrypted cipher text input $\langle C_1, C_2 \rangle = \langle m \cdot {p_k}^r, g^r \rangle$ to the decryption algorithm, the initial message is computed correctly by $\frac{C_1}{C_2^{s_k}} = \frac{{p_k}^r \cdot m}{(g^r)^{s_k}} = m$.

The basic property of the ElGamal crypto-system is the multiplicative homomorphism which is defined below.

Definition 2.11. Homomorphic Encryption. Let E be a probabilistic encryption scheme, M the message space and C the cipher text space, such that M is a group under operation $\oplus$ and C is a group under the $\otimes$. E is called a $(\oplus, \otimes)$-homomorphic encryption scheme is for any instance E of the scheme, given $c_1 = Enc_k^{r_1}(m_1)$ and $c_2 = Enc_k^{r_2}(m_2)$ there exists an r such that $c_1 \otimes c_2 = Enc_k^r(m_1 \oplus m_2)$.

ElGamal is multiplicatively homomorphic as the product (modulo p) of two cipher texts equals the cipher text of their product having $\langle m_1 p_k^{r_1}, g_1^r \rangle \cdot \langle m_2 p_k^{r_2}, g^{r_2} \rangle = \langle m_1 m_2 p_k^{r_1+r_2}, g^{r_1+r_2} \rangle$. An additive variant of the ElGamal crypto-system can be achieved by raising the message to be encrypted in the exponent of the group generator, encrypting g^m instead of m.

Notation. *From now on we will use the notation $Enc_k^r(m)$ to denote the randomized encryption of message m with the public key k using randomness r. We may write $Enc_k^{r \leftarrow Z_q}(m)$ to mention explicitly the choice of a random value in the encryption algorithm or abbreviate to $Enc_k^r(m)$ either for simplicity or to state that we use a specific random value r, which has be selected in advance. The same notation will be used for randomized commitment schemes like Pedersen commitments.*

2.6 Oblivious Transfer

Oblivious transfer protocols are two-party communication protocols for sharing information [9]. In an oblivious transfer protocol one entity, called the sender, stores a database of N elements and the other entity, called the receiver, wishes to retrieve one or more particular elements of the database, without revealing which values he wants. We symbolize the aforementioned cases as 1-out-of-N oblivious transfer protocol and k-out-of-N oblivious transfer protocol respectively.

Definition 2.12. Oblivious Transfer (OT). Let R be the receiver, choosing an index x and S be the sender storing a database $f = \{f_0, ..., f_{N-1}\}$. An 1-out-of-N oblivious transfer is a protocol that satisfies the following requirements.

1. Correctness: If both R and S execute the protocol faithfully, R gets f_x after executing the protocol with S, where x is its choice.

2. Receiver's privacy: After the execution of the protocol S shall not get information about R's choice x.

3. Sender's privacy: After the execution of the protocol R gets no information about any other database element f_i for $i \neq x$.

Regarding the protocol's security modol, we describe the receiver's security in terms of indistinguishability. More precisely for any elements f_i, f_j, $0 \leq i, j \leq N-1$, and any polynomial probabilistic and possibly malicious sender S^*, S^* cannot distinguish between the distribution of the receiver's requests for f_i and f_j with non-negligible probability. In the case of sender's security we require that the receiver gets no more information than those he is allowed to retrieve. We consider the ideal scenario where a trusted third party, given the sender's database as input along with the receiver's request i, outputs the correct element f_i. We require that for every polynomial possibly malicious receiver R^* in the real model there is a polynomial simulator S' in the ideal model, such that their outputs are computationally indistinguishable.

2.6.1 The AIR 1-out-of-N oblivious transfer

The protocol discussed in this section was designed by Aiello, Ishai and Reingold [8] and is based on the ElGamal crypto-system. The protocol, that works over a finite cyclic group G of prime order q, lets the receiver retrieve the element f_i he wishes, for the specified index x, while preventing him from gaining any knowledge about the rest of the database. This corresponds to the notion of conditional disclosure in a computational setting, that is to enable the sender to disclose the value f_i conditioned on $x = i$.

Protocol 7 The AIR $(1, N)$-OT

- The receiver R generates the public/secret key pair (pk, sk) and sends pk to the sender S, along with the Query $c = Enc_{pk}(x)$ of the element x.

- The sender verifies that pk is a valid public key and c a valid encryption. Then for every element of the database $(f_0, ..., f_{N-1})$, uniformly selects $a_j \leftarrow G$ and computes the message $m_j = a_j(x - j) + f_j$ and encrypts them under the receiver's key. Finally he sends the set of the encrypted elements $\{Enc_{pk}(m_0), ..., Enc_{pk}(m_{N-1})\}$ to the receiver.

- The receiver decrypts $m_i = Enc_{pk}(f_i)$ and outputs f_i.

In the above protocol, the decrypted message m_i equals f_x for $i = x$, otherwise it is a

completely random element in G, containing no significant information for the an information-theoretic perspective. Clearly the AIR OT protocol has linear communication and computational complexity in the size of the database.

Moreover we should note that the security conditions are relaxed. In case of a malicious sender, the simulator is allowed produce the sender's view without taking into consideration the receiver's output, that is the malicious sender is not required to know any information about the database. Since the simulator only needs to create a public-key and a random element's encryption, this can be done easily. In case of a a malicious receiver, we allow the sender's simulator to be computationally unbounded and output a statistically indistinguishable output from the original sender's output. Unboundedness is required in order for the simulator to be able to compute the secret key sk for the specific public key. Then, on a query for element indexed by x, the simulator lets m_i be a random encryption of element f_i, along with letting m_j for all $i \neq j$ to be random encryptions of random elements.

2.6.2 The proxy oblivious transfer

A useful extension of oblivious transfer is the so-called, proxy oblivious transfer. The proxy oblivious transfer protocol is a three-party communication protocol where the role of the receiver is divided among two separate entities: a chooser, that selects the elements to be retrieved from the database, and a proxy, that gets the selected elements, without knowing their identities.

Definition 2.13. Proxy Oblivious Transfer Protocol (POT). A proxy oblivious transfer protocol consists of 4 polynomial algorithms $\langle Gen_k, Query, Reply, Answer \rangle$.

1. The Gen_k algorithm generates the key pair $(p_k, s_k) \leftarrow Gen(1^k)$.
2. The randomized $Query_{p_k}(x)$ is the algorithm run among the chooser and the sender on input the index x, selected by the chooser.
3. The $Reply_{pk}(f, Query_{p_k}(x))$ is the randomized algorithm run among the sender and the proxy, having the query and the sender's database $\langle f \rangle$ as input.
4. The $Answer_{s_k}(Reply_{pk}(f, Query_{p_k}(x)))$ algorithm is executed by the proxy, on the data received from the sender, in order to obtain f_x.

Regarding the privacy guarantees of a proxy oblivious transfer we require CPA-security for the chooser, that is, no malicious polynomial sender S^* and no malicious polynomial proxy

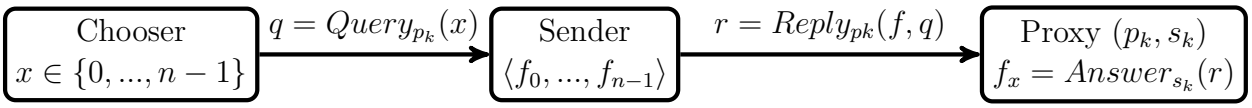

Figure 2.1: Proxy Oblivious Transfer

P^* should be able to distinguished between the Query/Reply messages respectively, for two plaintexts x_0, x_1 chosen by the sender, with non-negligible probability. Regarding the sender's security we require the existence of an unbounded simulator which, given some Query message and an honest's proxy output, should be able to output a Reply message, statistically indistinguishable from the honest sender's output.

2.7 Secret Sharing

In various situations certain information needs to be distributed among several entities which need to collaborate in order to reconstruct the initial value, otherwise no information can be extracted. This is known as the secret sharing problem where a dealer needs to distribute a secret to a group of n share-holders, requiring a threshold k of them to collaborate in order to reveal the secret. This is called a (k, n)-threshold secret sharing protocol, which is robust against coalitions of $n - k$ malicious entities. A (k, n)-secret sharing scheme should meet the following two requirements:

1. Knowledge of any k or more pieces makes the secret easily computable.
2. Knowledge of any $k - 1$ or fewer pieces leaves the secret completely undetermined, i.e. all its possible values are equally likely.

2.7.1 A n-out-of-n scheme

The simplest secret sharing scheme is the n-out-of-n scheme that distributes a secret at n different shares and requires that all shares are gathered to reveal it. To share a secret $s \in G$, where G denotes any group the scheme works, the dealer picks uniformly at random $n-1$ values $s_1, \ldots, s_{n-1}$ from G and sets $s_n = s - \sum_{i=1}^{n-1} s_i$. The shares s_i are given to n share-holders who need to combine all of them to reconstruct s as stated before. Any combination of less than n shares, that is the value $\sum_{i \neq i^*} s_i = s - s_{i^*}$ is a random value in G that contains no information about the secret.

2.7.2 Visual cryptography

Visual cryptography was introduced by Naor and Shamir [27] who designed a protocol for a k-out-of-n visual secret sharing. Their scheme provides a way so that people, what do not need to do any cryptographic or computational operations, are able to decode concealed images visually. In this approach printed text or images can be hidden in a perfectly secure way by using multiple different layers that reveal no information on their own, but reconstruct the initial image when k of them are combined and aligned.

Visual secret sharing works with each pixel of the original image separately, by preparing n different shares or modified variations of a certain pixel. Each share consists of m black and white sub-pixels so that the original image is represented by a $n \times m$ binary matrix $S = [s_{ij}]$, where s_{ij} is 1 if the j-th pixel of the i-th share is black, otherwise it is 0. The binary OR operation of all elements in a column represents the visual overlaying of the black and white sub-pixels, with the final picture having the j-th black sub-pixel if there is at least one 1 in column j. In order to reconstruct the initial share one has to overlay k rows of the matrix S, yielding a vector v, with the result being interpreted as black if the number of the black sub-pixels, denoted by the Hamming weight (the number of non-zero elements) $H(v)$, exceeds a pre-defined threshold $H(v) \geq d$, $(1 \leq d \leq m)$, or white if the number of black sub-pixels is bellow a certain threshold $H(v) \leq d - \alpha m$, for $\alpha > 0$, so that we can easily distinguish black from white pixels.

A k-out-of-n visual secret sharing scheme is constructed by two collections of $n \times m$ binary matrices C_0, C_1 where in order to share a white pixel or black pixel a dealer picks a matrix in C_0 or C_1 respectively. The 3 following conditions need to be met:

1. $H(v) \leq d - \alpha m$ for any $S \in C_0$ and any vector v constructed as the OR of any k rows of S.

2. $H(v) \geq d$ for any $S \in C_1$ and any vector v constructed as the OR of any k rows of S.

3. For any set of $\lambda < k$ rows of any $S \in C_0$ should be indistinguishable from a set of λ rows of any matrix $S' \in C_1$, in the sense that they contain the same vectors with the same frequencies

A simple 2-out-of-n visual secret sharing can be built by the collection of matrices C_0, C_1 defined

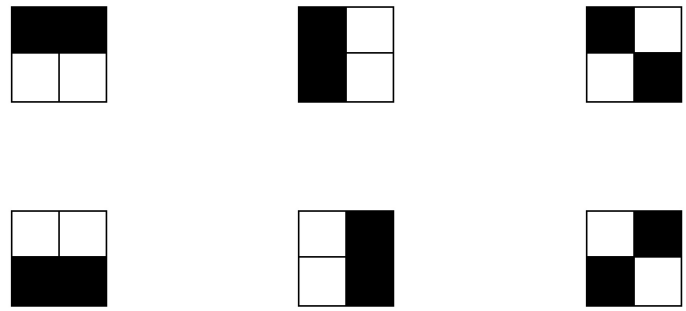

Figure 2.2: 2-out-of-2 visual secret sharing

bellow and the special case of the 2-out-of-2, with $d = 4$ and $\alpha = 1/2$ is depicted in figure 2.2.

$$C_0 = \{\text{all matrices by permuted columns of } \begin{bmatrix} 100\ldots0 \\ 100\ldots0 \\ \ldots \\ 100\ldots0 \end{bmatrix}\}$$

$$C_1 = \{\text{all matrices by permuted columns of } \begin{bmatrix} 100\ldots0 \\ 010\ldots0 \\ \ldots \\ 000\ldots1 \end{bmatrix}\}.$$

Similarly a k-out-of-k secrets sharing is created, with $m = 2^{k-1}, \alpha = 1/2^{k-1}$, having $r = 2^{k-1}!$ matrices in collections C_0, C_1. Each collection contains $k \times 2^{k-1}$ matrices, with C_0 constructed by all column permutations of row vectors with even number of 1 and C_1 containing all column permutations of row vectors with odd number of 1. Based on a k-out-of-k scheme with $C_0 = \{T_1^0, T_2^0, \ldots, T_r^0\}, C_1 = \{T_1^1, T_2^1, \ldots, T_r^1\}$ a k-out-of-n scheme is derived, by considering a collection H of l functions h from $\{1, \ldots, n\}$ to {1,...,k}. Let $B \subset \{1, \ldots, n\}$, with $|B| = k$ and p_q be the probability that a randomly chosen function $h \in H$ yields q different values on B, with $1 \leq q \leq k$. Then the new scheme's parameters are $m' = ml, \alpha = p_k\alpha$ and $r' = r^l$ with each collection of matrices containing the matrices S_t^b for $1 \leq t \leq r^l$ and $b = 1, 2$, where $t = (t_1, \ldots, t_l)$ and $1 \leq t_i \leq r$. The new matrices are composed as $S_t^b[i, (j, h)] = T_{t_j}^b[h(i), j]$. The new scheme corresponds in repeating l independent times the k-out-of-k scheme inheriting

its security.

2.8 The Communication channels

Throughout our discussion regarding electronic voting we will use several types of communication channels offering different security guarantees. Below we summarize the categories of communication channels.

- Voting booths: A voting booth is a physical apparatus used in kiosk voting to cast a vote. It guarantees the secrecy of the communication between the voter and the voting servers and provides vote integrity since it is considered a trusted voting platform, ensuring also coercion-resistance as no other can be present during vote submission.
- Untappable channels: An untappable channel is the stronger physical assumption we may ask for in remote voting. An untappable channel is a private channel that prevents a adversary from intercepting sent messages, as all information sent through an untappable channel established between two entities remains perfectly secret to all other parties.
- Anonymous channels: An anonymous channel is an secure channel that prevents an adversary from identifying the sender of a message. Anonymous channels can be designed by special constructions like mixing networks.
- Broadcast channels: A broadcast channel is secure channel, often with memory, where multiple senders can post publicly available information, accessed by multiple receivers.
- Encrypted channels: A secure channel can be implemented by using cryptographic techniques like encryption. A secure channel protects the transmitted data from a computationally bounded, passive attacker.
- Out-of-band channels: A channel that is referred as out-of-band communication channel denotes a private physical channel that by-passes certain entities that are considered weak in terms of security.

Chapter 3

The Untrusted Platform Problem

Personal computers that will be used in Internet voting can be easily compromised with a dramatic effect on the elections outcome. Malicious voting clients may try to learn or even alter that vote that the user submits. Thus special measures should be taken in order to hide the voting values from malicious computers and prevent them from altering the elections' outcome without being detected from the voters. As personal computers are the vulnerable part of the procedure the vote verification step requires immediate communication with the voters, through out-of-band communication channels, without relying on the computers. We only discuss solutions that do not depend on any specialized hardware in vote verification phase, through code verification devices, or vote submission phase, through trusted smart-cards.

Receipts are a highly controversial issue in electronic voting, as although they increase the credibility of the system, they contradict by default the property of coercion-resistance/receipt-freeness. In order to deal with this issue, many e-voting system allow the voters to re-vote as long as the election is running, taking into account only the last submitted votes. In addition, paper voting is also available after the end of the Internet voting period, which requires the system's authorities to be able to cancel electronically submitted votes. Receipts are used to ensure the integrity of a submitted vote throughout an untrusted platform.

We begin our discussion presenting the entities involved in voter verifiable protocols along with their role.

- **Voters** (V): Voters are people that participate in the elections casting ballots that represent their preferences. Throughout our discussion we will consider ballots consisting

of a single option chosen by the voter, that is a 1-out-of-N candidates ballot format for simplicity, unless otherwise stated. All the schemes we review can be extended to support k-out-of-N ballot format, without taking the order into account, either trivially by including multiple chiphertexts in the vote or by employing new vote encoding that allow this option. The voter verify their votes by receipts provided by the voting system.

- **Personal Computers** (PC): Internet voting uses the personal computers of the voter's for the submission of the votes. Thus many important security problems arise from this option, as voter's computers are vulnerable to several attacks that may alter the elections' outcome.

- **Vote collectors** (VC) or **Ballot Boxes** (B): Vote collectors are the entities that store the submitted encrypted ballots during the election period and forward them to the tallier after the procedure is over. Additionally, the vote collectors cooperates with the messengers to reconstruct the vote receipts.

- One or more **Messengers** (MS) or **Receipt Code Generators** (RCG) : Receipt code generator severs are responsible to cooperate with the vote collectors to produce the receipt codes for vote verification.

- **Talliers** (T): Talliers are the entities that are responsible to decrypt and tally the submitted votes, defining the outcome when the election period has expired.

- **Out-of-Band Communication Channels**: Voters should be given receiptsfor each candidate and those receipts should be sent back to the voters. Thus we need a means to provide this information, avoiding contact with the personal computers

- **Auditor**: An authority that supervises the entire process, sees the contents of each involved entity and checks that everyone executes the protocol faithfully.

3.1 Code Voting

Code voting protocols require a set of online voting server that act as receipt code generators. SureVote, presented by D. Chaum [7] was the first Internet voting protocol that allowed vote verification through untrusted voting clients. In this approach the voters receive before the elections a voting card, through an out-of-band communication channel. Every voting card,

which is identified by a unique number (ID), has different, personalized voting codes and receipt codes for each voter and candidate. A voter casts a ballot by submitting the voting card's identification number for authentication purposes and the voting code that corresponds to the candidate of her choice. The voting code is a pre-encrypted ballot that together with the voting card's identification number enables the receipt code generators to extract the suitable voting receipt. Finally the during the tallying process voting codes need to be translated into correct candidates, using a suitable zero-knowledge proof to show the validity of the mapping.

The fact that pre-encrypted ballots are pseudo-random random numbers, unknown by the PC, guarantees privacy against malicious voting clients that cannot extract any information about the submitted vote. Moreover there is no means for a malicious computer to cast a vote for a different valid candidate, guaranteeing vote integrity. Finally the voter is assured that her ballot was recorded as cast upon receiving the correct receipt code. The receipt code is generated by the identification number and the voting code by a set of online receipt generators and can be returned to the voter using the PC as the communication channel, since the PC is unable to vote for a different candidate after receiving the correct receipt code.

An alternative scheme called Pretty Good Democracy (PGD) was described by Ryan and Teague in [5], which uses a single receipt code for each voting card, to achieve a degree of receipt-freeness. The voters are given voting cards with a single receipt code and a set of voting codes, such that all voting codes are distinct. The protocol requires a set of online receipt code generator servers that store for each voting card a permuted list of encryptions of all voting codes, an encryption of the permutation used for the particular voting card and an encryption of the receipt code. Upon submission of a candidate code and an voting card identification number the servers perform a plaintext equality test to find a matching encrypted voting code and jointly decrypt the receipt code of the voting sheet, which is given to the voter and can be safely posted on the bulletin board. By this trick the voter is assured that a ballot is counted on her behalf, although she cannot determine which option was cast. Vote tallying requires combining the matching encrypted voting code with the encrypted stored permutation to extract the correct vote.

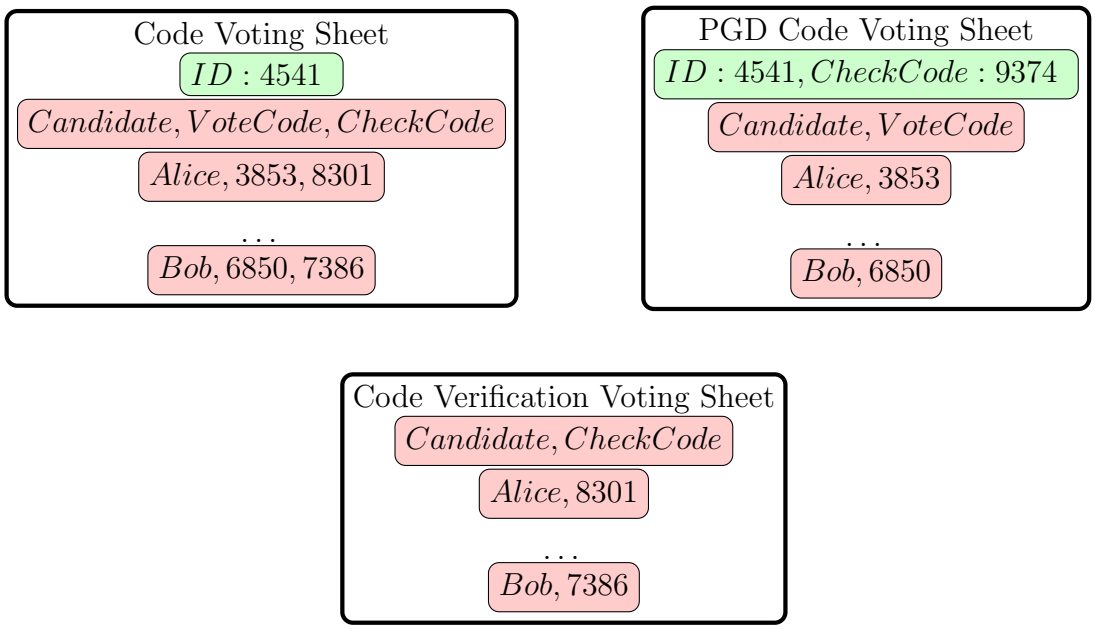

Figure 3.1: Comparison of different code sheet's types

3.2 Code Verification Voting

In this part of the chapter we discuss the electronic voting system that Norway plans to apply to conduct elections until 2017. A trial of protocol has already been implemented for the local elections in Norway in September 2011. We will focus on all the aspects of this protocol, discussing the role and properties of each entity and presenting the relevant security guarantees. The main attribute that makes this protocol attractive and unusual is the code verification property that allows the voters to check whether their votes reached the protocol's infrastructure entities.

In code verification voting a single messenger server is sufficient to reconstruct the security code from the encrypted, submitted ballot and notify the voter. Code verification voting uses a globally known correspondence of candidates and voting codes, in contrast to code voting, but it employs voter-dependent security codes that are associated with each candidate. These candidate-code pairs are generated by trusted servers prior to the election and need to be delivered in time to the voters, using a secure pre-channel, such as regular mail, to prevent the computers from learning the codes. When the election period starts, voters cast ballots to their computers that encrypt and forward them to the vote collector. Immediately the vote collector should send the necessary information to the messenger, who provides the voter the security

code, through a secure post-channel, such as an SMS.

In the following sections we study the proposed code-verification protocols that guarantee vote integrity in case of malicious computers and discuss their efficiency and the new threats that arise.

3.3 The Proxy Oblivious Transfer Approach

This protocol was proposed in [1] by Heiberg et al. to be adapted to the Norwegian E-voting system in order to guarantee vote integrity against malicious personal computers. The protocol is based on completely random security codes, which are generated by an ideal hash function, viewed as a random oracle, and randomly selected pre-codes. In order to let the messenger reconstruct the security code and notify to the voter, a proxy oblivious transfer protocol is used. The protocol allows re-voting as a means against coercion.

A 1-out-of-n proxy oblivious transfer is a two-message communication protocol between 3 entities: a chooser, a sender and a proxy. The chooser chooses an index $x \in \{0, ..., N-1\}$, which is obliviously transferred to the messenger, through the sender. The chosen index is sent encrypted to sender, who stores an ordered database $f = (f_0, ..., f_{N-1})$ that should be provided to the proxy, conditionally encrypted on the selected index x. Then the proxy, who has the decryption key, obtains f_x at the end of the protocol without getting any knowledge about the index.

3.3.1 POT E-Voting

We present the proposed e-voting protocol, which employs the ElGamal crypto-system, over a finite cyclic subgroup $G \subset Z_p$ of prime order q, generated by g, where $q|p-1$ and p, q are large primes. In this setting the chooser corresponds to the voter's computer that casts the vote, the sender to the vote collector, who should store the database of the pre-codes, and the proxy to the messenger, who needs to obtain the correct pre-code and reconstruct the human-readable security code. The POT protocol described in this section is based on the AIR oblivious transfer protocol presented in section 2.6.1. There are two different versions of the protocol, depending

on the size of the database's elements. If these elements are big, so that it is inefficient to solve the discrete logarithm problem, the proxy is allowed to store an unordered, sorted version of the database, denoted by $F = \{g^{f_0}, ..., g^{f_{N-1}}\}$, otherwise the proxy stores no additional information. We call this version a weak proxy oblivious transfer.

Protocol 8 (Weak) Proxy Oblivious Transfer

1. Gen: The proxy runs the key generation algorithm $(sk, pk) \leftarrow Gen(1^k)$. The proxy's public key is given to the chooser and the sender.
2. Query: The chooser chooses x and sends $e = Enc_{pk}^{r \leftarrow Z_q}(g^x)$ to the sender.
3. Reply: For every element f_i of its database, the sender selects $r_i \leftarrow Z_q$ and computes $e_i = (\frac{Enc_{pk}^{\rho=1}(g^i)}{e})^{r_i} \cdot Enc_{pk}^{r'_i \leftarrow Z_q}(g^{f_i})$. Then he sends the set $\{e_0, \ldots, e_{N-1}\}$ to the proxy.
4. Answer: The proxy decrypts each element a in the reply set obtaining $y = Dec_{sk}(a) = g^z$.
   - Simple POT: The proxy outputs the smallest result $z_{min} = dlog_g y$.
   - Weak POT: The proxy outputs the decrypted element that belongs to F.

Completeness and security. In the above protocol the Reply set is created in the desired format due to the homomorphic properties of ElGamal and the fact that exponentiation can be done efficiently over already encrypted elements. Thus each element is encrypted as $Enc_{pk}^{r'_i=(1-r)r_i+r'_i}(g^{(i-x)r_i+f_i})$. The election scheme we discuss uses the weak POT protocol, where the proxy decrypts the elements and looks for them in F. Let x be the submitted vote. Then for $i = x$ the proxy obtains $g^{f_x} \in F$, the only valid value he wants to retrieve. For all other elements ($i \neq x$), the decrypted value $g^{(i-x)r_i+f_i}$ is a completely random group element which does not reveal any information about the database, guaranteeing the sender's privacy. Concerning the chooser's privacy, the sender sees only an encrypted value that hides x and the proxy, that retrieves g^{f_x}, learns nothing about the index x. The proxy oblivious transfer is the bottleneck of the protocol as in requires linear number of exponentiations in the number of the candidates. As security codes are completely random in this approach this overhead is inevitable.

From the discussion above it becomes clear that the voter's PC will have to encrypt each submitted vote with the messenger's public key, in order to run the oblivious transfer protocol and return the correct security codes. Thus, the voter's PC is required to create a different

encryption of the vote, under the tallier's public key, in order for the vote to be successfully counted. Although this process has the advantage that the tallier's vote decryption is completely independent from the messenger's pre-code decryption, it has an additional overhead as the PC should prove in zero knowledge that both the ciphertexts encrypt the same value $\pi = PK(\mu, r_1, r_2 \mid e_1 = Enc^{r_1}_{pk_1}(g^\mu) \wedge e_2 = Enc^{r_2}_{pk_2}(g^\mu))$. In order to prove that two ciphertexts of this format correspond to the same plaintexts encrypted under different keys the authors use an adaptation of Schnorr's protocol. In addition the voter should be able to authenticate his identity to vote collector so that we achieve voter's eligibility. Thus the voter should be able to sign their ballots using a double envelope scheme, so that the vote collector can verify the signatures. We assume that there is a public-key infrastructure that enables the voter's to sign messages and that can be verified by the vote collector. Any EUF-CMA secure signature scheme suffices for this purpose.

Protocol 9 ZKP encryptions with equal plain-texts

Let $e_1 = Enc^{r_1}_{pk_1}(g^m), e_2 = Enc^{r_2}_{pk_2}(g^m)$.

1. The prover chooses $m' \leftarrow Z_q$ and sets $i_1 = Enc^{r'_1 \leftarrow Z_q}_{pk_1}(g^{m'})$ and $i_2 = Enc^{r'_2 \leftarrow Z_q}_{pk_2}(g^{m'})$.
2. The verifier chooses the challenge $c \leftarrow Z_q$ and sends it to the prover.
3. The prover computes and sends $m^* \leftarrow m' + m \cdot c$ mod q, $r^*_1 \leftarrow r_1 + r'_1 \cdot c$ mod q and $r^*_2 \leftarrow r_2 + r'_2 \cdot c$ mod q.
4. The verifier accepts if $Enc^{r^*_1}_{pk_1}(g^{m^*}) = i_1 \cdot e^c_1$ and $Enc^{r^*_2}_{pk_2}(g^{m^*}) = i_2 \cdot e^c_2$

Protocol 10 POT Code-Verification E-Voting: Setup

1. Select the system parameters $\langle G, g, q\rangle$ and the hash function $H : G \rightarrow Codes$.
2. For every voter V a trusted server creates voter-dependent pairs of candidates and security codes, selecting $R_V[cnd] \leftarrow Z_q$ and setting $Code_V[cnd] = H(g^{R_V[cnd]})$.
3. The pairs $(cnd, Code_V[cnd])$ are sent by a secure pre-channel to the voters. The pair $(cnd, R_V[cnd])$ are signed and sent to the vote collector creating its database and the values $g^{R_V[cnd]}$, after being sorted and signed, are given to the messenger.

Protocol 11 POT Code-Verification E-Voting: Vote Submission

Let $(pkt, skt), (pkm, skm)$ be the public key pair of the tallier and messenger respectively and $(sk_V, vk_V), (sk_{vc}, vk_{vc})$ be the signing key pair for the voter and the vote collector.

- The voter: submits to his computer a valid candidate number *cnd* and waits for the security code.
- The voter's PC:
  Knows $pkt, pkm, (sk_V, vk_V)$.
  1. Creates the message $E_m = Query_{pkm}(cnd)$ of the POT protocol 8, encrypted with the messengers public key pkm.
  2. Creates the encrypted ballot $E_t = Enc_{pkt}(g^{cnd})$, under the tallier's public key pkt along with a NIZKP $\pi = (\mu, r, r' \mid E_t = Enc^{r}_{pkt}(g^{\mu}) \wedge E_m = Enc^{r'}_{pkm}(g^{\mu}))$.
  3. Sings $\sigma = Sign_{sk_V}(\pi, E_m, E_t)$ and sends $(VoterID, \sigma, \pi, E_t, E_m)$ to the vote collector.
  4. Waits for an accept/reject message from the vote collector.
- The vote collector:
  Knows $pkm, vk_V, (sk_{vc}, vk_{vc}), \forall\ V\ R_V = \langle R_V[0], \ldots, R_V[N-1] \rangle$.
  1. Verifies π, σ and accepts/rejects accordingly.
  2. Stores the ballot replacing previously submitted ballots from the same voter.
  3. Creates the set *Reply*=$\{e_0, ..., e_{N-1}\}$, e_i being $(\frac{Enc^{\rho=1}_{pk}(g^i)}{E_m})^{r_i} \cdot Enc^{r'_i \leftarrow Z_q}_{pk}(g^{R_v[i]})$ of the POT protocol (protocol 8).
  4. Signs $\sigma' = Sign_{sk_{vc}}(Reply)$ and sends $(VoterID, Reply, \sigma')$ to the messenger.
  5. When the election ends signs each submitted ballot $\tilde{\sigma} = Sign_{sk_{vc}}(E_t)$ and forwards $(\tilde{\sigma}, E_t)$ to the tallier.
- The messenger:
  Knows $vk_{vc}, (skm, pkm), \forall\ V\ \{g^{R_V[0]}, \ldots, g^{R_V[N-1]}\}$.
  1. Verifies σ'.
  2. Decrypts the elements of the *Reply* set and outputs the Answer $z = g^{R_v[cnd]}$ of the POT protocol 8. Then he obtains the security code $Code = H(z)$ and notifies the voter $VoterID$ by sending him the code through the post-channel.

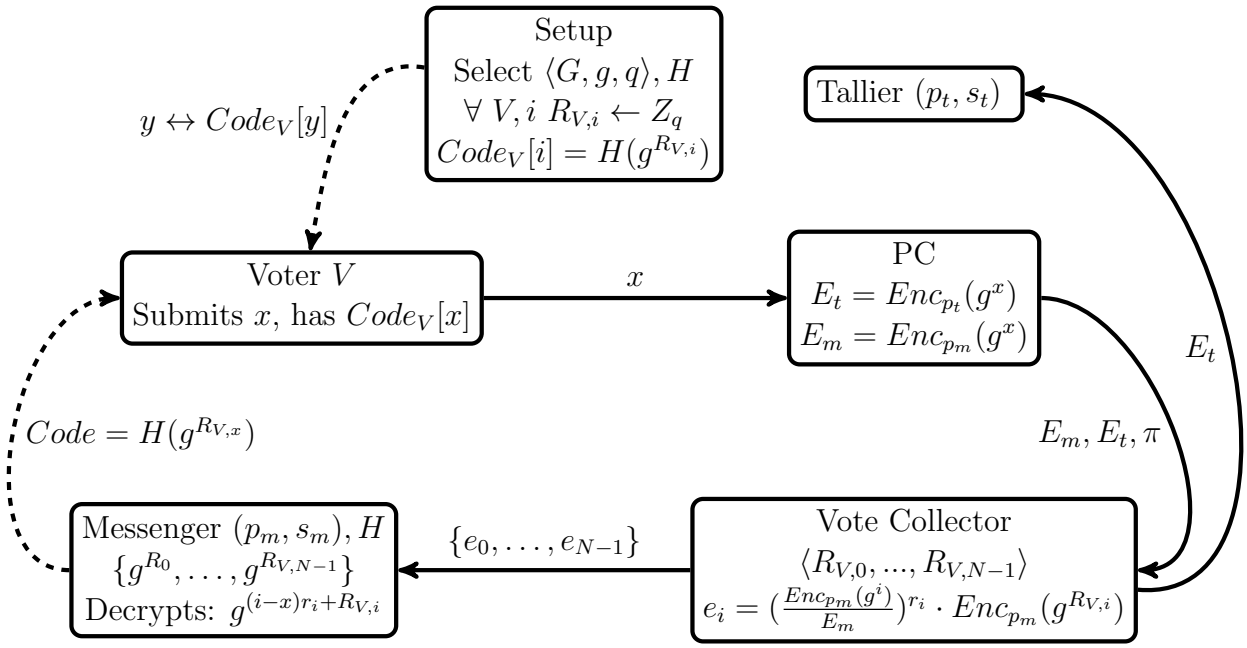

Figure 3.2: The POT E-voting

3.3.2 Security guarantees and weaknesses

Regarding the security of the proxy oblivious transfer e-voting protocol guarantees can be provided only in the case where one infrastructure player is corrupted while the other entities are honest. More specifically, regarding privacy, a malicious vote collector gains no information about the submitted votes, as he sees only the encrypted cipher texts and ElGamal is semantically secure. Regarding vote integrity, since submitted ballots are signed, a malicious vote collector cannot alter the votes. Moreover, concerning the privacy aspect, the messenger sees only the vote-independent pre-codes and the verification code of the submitted vote which contain no information about the vote itself. The only information a messenger can extract is to decide if the voter votes for a new candidate in case of re-voting. The voter's computer sees the vote itself, having no privacy guarantees, however it cannot alter it and submit a different ballot without being detected by the voter who receives a wrong verification code. Moreover, anytime a malicious computer tries to submitted a fake vote on behalf of an honest voter, the voter will be notified due to an unexpected verification code and complain about a forgery.

On the other side, code-verification protocols pose certain threats regarding vote's privacy and security in case of coalition of two (or more) entities. Probably the most important coalition

which cannot be prevented from the existing protocols is that of a malicious PC with a malicious messenger. In this scenario, the malicious PC initially submits the valid candidate x which is entered by the voter and the messenger generates the appropriate verification code $Code_V[x]$. However the messenger may delay sending back the verification code, until the malicious PC creates a forged ballot $x' \neq x$ and re-votes on behalf of the unaware voter. Then the messenger discards the new security code and sends back the old-one instead. Thus the voter receives the correct verification code while a forged ballot has been submitted undetected. Otherwise the PC itself may submit a forged ballot without the voter's participation as long as the messenger is willing to drop the security code. Both scenarios imply that the messenger should be able to deviate from its standard function, sending no codes at all.

Last but not least, probably the most important drawback of the protocol is that the coalition of the vote collector and the messenger can break the voter's privacy, even if they exchange partial information, i.e. they do not decrypt the votes directly. Thus in the case they execute the protocol correctly but share their stored information, the vote collector's database and the messenger's output, privacy will be lost by comparing the value $g^{R_V[i]}$ obtained by the messenger with the pre-code database stored by the vote collector, which provides the correspondence between each candidate i and the value $R_V[i]$. Furthermore, the correct index, i.e. the corresponding candidate may be identified by the collaboration of those entities if the vote collector sends the messenger an ordered tuple instead of a set. This can be prevented by running a mixing network between the two two entities at the expense of the additional computational cost.

3.4 The Pseudo-random Composition Approach

The protocol discussed in this section is the one that Norway decided to adapt for generating the security codes and running its elections. Several papers analyze and discuss the features of the Norwegian voting protocol which aims to become a transparent and well-studied protocol in order to ensure voter's trust [2] [4] [11] [3]. This method is considerably faster compared to the proxy oblivious transfer approach, as there is no need to send a whole database in order to retrieve a particular random security code. In order to achieve this property, the protocol uses pseudo-random instead of completely random codes, which are constructed as the superposition of three pseudo-random functions on the submitted vote, that is an encoding function, a blind-

ing factor and a pseudo-random function selected from an appropriate family. As trade-off this approach requires a more complex setup phase, where in order to generate the security codes, all three pseudo-random functions should be evaluated by trusted servers. The system uses he ElGamal crypto-system and works over a finite cyclic group $G \subset Z_q$ of prime order q, such that $q|p-1$. It takes advantage of the multiplicativelt homomorphic property of ElGamal that allows exponentiation to be done inside a cipher text.

Another advantage compared to the proxy oblivious transfer approach is that due to a smart trick, there is no the need to encrypt the submitted vote with two different keys and prove equality of plaintexts in zero knowledge. In order to achieve this property, the protocol shares the tallier's secret key among the personal computers and the vote collectors in order to make one encrypted vote sufficient both for the tallying and code reconstruction processes. However this fact poses a new threat as the coalition of a dishonest vote collector and messenger can reconstruct the tallier's secret key and break protocol's privacy.

3.4.1 The shared-key E-voting

We now proceed in describing the voting protocol. For our discussion we assume that a voter submits a single option each time, so that tallying can be done effectively. Later we will further analyze and remove this assumption, generalizing to an efficient k-out-of-N ballot format. We separately present the key and code generation phase, which we assume to be done by trusted servers prior to the election, the election's phase and we briefly describe the counting phase for the extended ballot format.

During vote submission the computer submits a signed ballot and proves knowledge of it contests in order to prevent the corruption scenario where a malicious vote collector along with a corrupt voter could submit honest voter's ciphertexts as its own and then learn the vote from the receipt codes. Given a ciphertext $E_t = \langle x_t, w_t \rangle = \langle g^r, pk_t^r f(cnd) \rangle$ the computer can prove knowledge of the plaintext $f(cnd)$ by proving knowledge of r as the voter could obtain $f(cnd)$ from E_t using r. Thus proof is a proof of discrete logarithm knowledge $\pi_v = PK(r \mid x_t = g^r\ )$ using the Schnorr protocol.

The public keys of the shared-key protocol are created so that they satisfy the relation

Protocol 12 Shared-Key Code Verification E-Voting - Setup Phase

Let $\mathcal{F} : G \rightarrow Codes$ be a pseudo-random function family.

1. Choose values $a_2, a_3 \leftarrow Z_q$ and set $a_1 = a_1 + a_3$ mod q. Let $sk_{vc} = a_2$, $sk_m = a_3$ and $sk_t = a_1$ be the secret keys of the vote collector, the messenger and the tallier respectively, while $pk_{vc} = g^{a_2}$, $pk_m = g^{a_3}$ and $pk_t = g^{a_1}$ denote their public keys.
2. Select a global encoding function $f : C \rightarrow G$, where C is the set of candidates
3. For every voter V select a secret exponent $s_V \leftarrow Z_q$, compute the commitment $\gamma = g^s$ and choose a pseudo-random function d_V from $\{\mathcal{F}\}$. Send the corresponding (V, s) pairs to the vote collector and the corresponding tuples (V, γ, d_V) to the messenger.
4. For every voter-candidate pair compute the security codes $Code_V(cnd) = d_V(f(cnd)^{s_V})$ and send the pairs $(cnd, Code_V(cnd))$ to the voter V through the secure pre-channel.

$pk_t = pk_{vc} \cdot pk_m$, as $g^{a_1} = g^{a_2} g^{a_3}$ which is shown in protocol 12. This property gives the vote collector and the messenger the opportunity to partially decrypt cipher-texts encrypted with the tallier's public key. Thus, in the second step of the submission protocol (protocol 13) the vote collector blinds the input ciphertext $E_t = \langle x_t, w_t \rangle = \langle g^r, pk_t^r f(cnd) \rangle$ with the secret exponent s_V obtaining $E_{blind} = E_t^{s_V} = \langle x_{blind}, w_{blind} \rangle = \langle x_t^{s_V}, w_t^{s_V} \rangle = \langle g^{s_V r}, pk_t^{s_V r} \cdot f(cnd)^{s_V} \rangle$. The corresponding proof of correct computation π_{blind} consists of proving knowledge of discrete logarithm in G^3 such that $\pi =_{blind} PK(s \mid \gamma = g^s \wedge x_{blind} = x_t^s \wedge w_{blind} = w_t^s)$.

Then the vote collector uses his secret key $sk_{vc} = a_2$ on the ciphertext E_{blind} obtaining $\langle x_m, w_m \rangle = \langle x_{blind}, \frac{w_{blind}}{x_{blind}^{a_2}} \rangle = \langle x_{blind}, w_{blind}(x_{blind}^{-a_2}) \rangle = \langle g^{s_V r}, (g^{a_1 - a_2})^{s_V r} f(cnd)^{s_V} \rangle = \langle g^{s_V r}, pk_m^{s_V r} f(cnd)^{s_V} \rangle$ which is can be decrypted by the messenger that owns the secret key $sk_m = a_3$. This is accompanied by a proof of knowledge of correct partial decryption $\pi_{pdecr} = PK(a_2 \mid pk_{vc} = g^{a_2} \wedge w_{pdecr} = x_{blind}^{-a_2})$, which is a dlog proof in $G \times G$, with $w_{pdecr} = x_{blind}^{-a_2}$. The final proof consists of $\pi_{correct} = (w_{blind}, w_{pdecr}, \pi_{blind} \wedge \pi_{pdecr})$, with its steps and verification depicted in table 3.1.

As soon as the messenger receives the values from the vote collector it checks the validity of the proofs. Upon success it applies the function d_V he owns on the decrypted value $f(cnd)^{s_V}$ obtaining the corresponding security code.

Protocol 13 Shared-Key Code Verification E-Voting - Vote Submission

Let $(pkt, skt), (pkm, skm)$ the public key pair of the tallier and messenger respectively and $(sg_V, vk_V), (sg_{vc}, vk_{vc}), (sg_m, vk_m)$ be the signing key pair for the voter, the vote collector and the messenger.

- The voter V: Submits a vote for candidate x to his computer and waits for the integrity cod.
- The voter's PC:
  Knows $f, (sg_V, vk_V), pk_t$.
  1. Computes the encoding $f(x)$ and encrypts $E_t = Enc_{pk_t}^{r \leftarrow Z_q}(f(x)) = \langle x_t, w_t \rangle = \langle g^r, pk_t^r f(x) \rangle$.
  2. Proves knowledge of the encrypted vote π_v.
  3. Computes the signatures the ballot $\sigma = Sign_{sk_V}(E_t)$ and sends (V, E_t, σ) to the vote collector.
  4. Waits for a signature σ' from the vote collector, verifies it and accepts/rejects accordingly.
- The vote collector:
  Knows $s_V, (pk_{vc}, sk_{vc}), (sg_{vc}, vk_{vc})$.
  1. Verifies σ, π_v on the encrypted ballot, signed it $\sigma^* = Sign_{sg_{vc}}(E_t)$ and stores it, replacing any previously submitted ballots.
  2. Blinds E_t computing $E_t^* = {E_t}^{s_V}$.
  3. Partially decrypts E_t^* computing $E_m = Decr_{sk_{vc}}(E_t^*) = Enc_{pk_m}(f(x)^{s_V})$.
  4. Creates a ZK proof π of correct computation $\pi_{correct}$.
  5. Sends $(V, E_t, E_m, \sigma, \pi_{correct}, \pi_v)$ to the messenger and E_t to the tallier when the election period ends.
  6. Waits for σ' from the messenger and forwards it to the voter's PC.
- The messenger:
  Knows $d_V, \gamma, (pk_m, sk_m), (sg_m, vk_m)$.
  1. Verifies proofs $\pi_{correct}, \pi_v$.
  2. Decrypts the received values with sk_t obtaining $z = f(x)^{s_V}$, and computes the security code $Code = d_V(z)$ which sends directly to the voter using a secure post-channel.
  3. Computes the signature $\sigma' = Sign_{sg_m}(Code, Voter)$ and sends it to the voter V, through the vote collector.

Proof	Prover	Required knowledge
π_{blind}	$(g, x_t, w_t)(g^{s_V}, x_{blind}, w_{blind}) =$ $(g, x_t, w_t)(g^{s_V}, x_t^{s_V}, w_t^{s_V})$	DLOG s_V
π_{pdecr}	$(g, x_{blind})(pk_{vc}, w_{pdecr}) =$ $(g, x_{blind})(g^{a_2}, x_{blind}^{a_2})$	DLOG $sk_{vc} = a_2$
$\pi_{correct} =$ $(w_{blind}, w_{pdecr}, \pi_{blind}, \pi_{pdecr})$		$w_{blind} = w_t^{s_V}, w_{pdecr} = x_t^{s_V a_2}$
	Verifier	Required knowledge
$\pi_{correct}$	$w_m = w_{blind} \cdot w_{pdecr}$	Messenger's input $\langle x_m, w_m \rangle$
π_{blind}	Check on $(g, x_t, w_t)(\gamma, x_{blind}, w_{blind})$	$x_{blind} = x_m, \gamma$
π_{pdecr}	Check on $(g, x_t)(pk_{vc}, w_{pdecr})$	$pk_{vc} = g^{a_2}$

Table 3.1: The PRF composition: Knowledge required by the entities

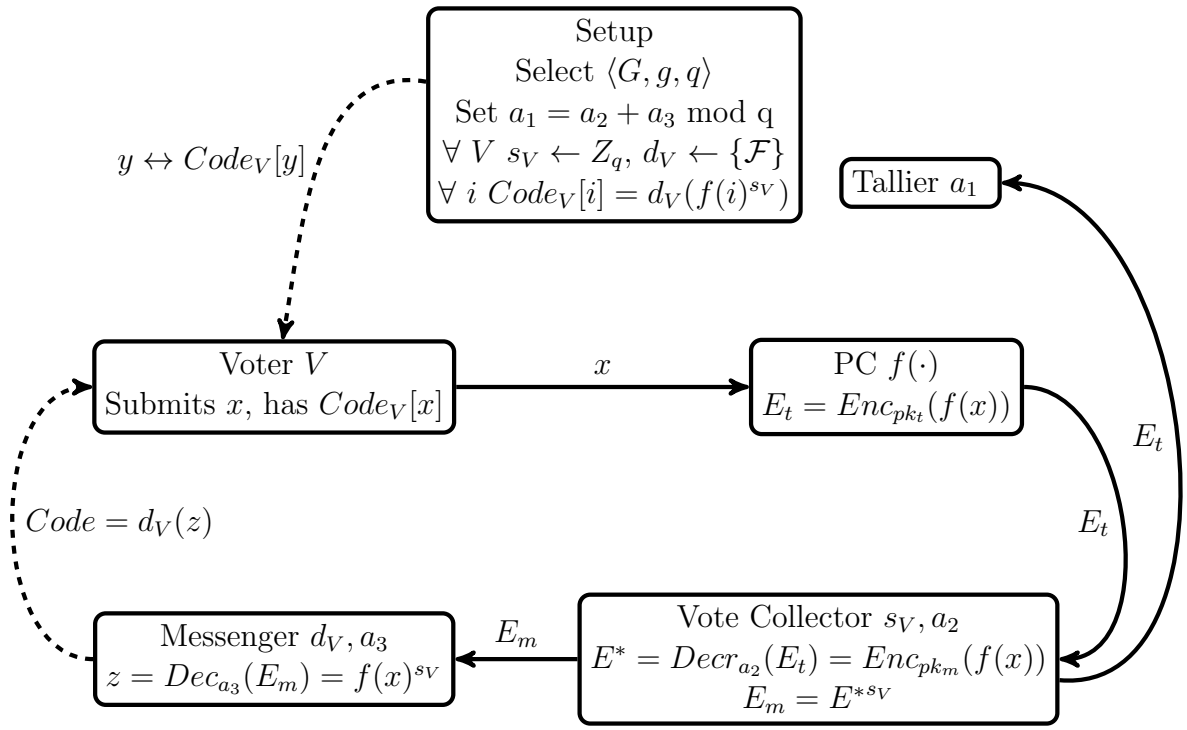

Figure 3.3: Pseudo-random composition shared-key E-Voting

3.4.2 Vote encoding and tallying improvements

In the previous discussion for simplicity we assumed that each ballot consists of a single candidate. However this can be extended to a ballot format that allows voting for multiple candidates at the same time, without taking the order into account. In the case of k-out-of-N ballot format the voter votes for k or less options while the rest are padded to the fix length. The security codes are generated for each submitted candidate individually in order to maintain the protocol's usability while tallying can be computed either for each ciphertext separately or can take advantage of a new proposed model that allows multiple ciphertext compression into one and full recovery.

The vote encoding function has a significant impact on the protocol's efficiency regarding tallying, as well as its security guarantees. If we choose the encoding function to be a random injection from the candidate set to G then tallying has a significant overhead, having to mix and decrypt multiple ciphertexts per candidate. In this section we discuss a novelty which takes advantage of a special group structure in order for the protocol to be able to compress and recover ElGamal ciphertexts. In order to achieve this functionality, the encoding function should be of a special form. As the encoding function changes, the security of the protocol is based on a new problem, related to the DDH, which is is believed to be hard on this specific group structure. However, as a trade-off for the efficiency increase, we simply rely on this conjecture, as there is no proof to support this statement.

Let q, p be primes such that $p = 2q + 1$ and let G be the group of the quadratic residues of Z_p. We denote by L the set of the smallest primes $\{l_1, l_2, ..., l_L\}$, $l_i \in G$, such that $l_i \leq \sqrt[k]{p}$, and we define the encoding function to be a random injection from the set of candidates to the set L. Although factoring in the general case is considered a computationally hard problem, its variant that deals with small primes can be efficiently solved. Thus if we are given the product of k small primes then we can efficiently recover the primes involved. In this way, we define a map $\phi : G \to G^k$ that assigns each product $a = \prod_{k=i}^{j} l_k$ of the primes of L to an ordered tuple $(l_i, ..., l_j)$ of the involved primes and any other group element x to the tuple $(1, ..., 1, x)$. We are now ready to present the problem that forms the bases of the improved protocol.

Definition 3.1. Prime DDH. Given $(l_1, ..., l_L) \in G^n$ decide if $(x_0, x_1..., x_L) \in G^{L+1}$ was sampled uniformly from the set $\{g^s, l_1^s, , ..., l_L^s\}$ (with $0 \leq s < q$) or uniformly from G^{L+1}.

The above problem is very similar to the DDH problem and although Prime DDH is hard only if DDH is hard, it seems difficult to prove its hardness from the general DDH. For the purposes of the voting protocol we discuss it suffices to rely on the alleged hardness of a simplified version of this problem. The new problem asks, given a permutation of a subset of the powers of the elements in L $\{l_i^s, ..., l_j^s\}$, to deduce information about the primes and permutation that were used.

We consider the simple case of elections with two options where two primes l_0, l_1 are involved and prove it equivalent to the DDH problem. The same argumentation holds for deciding whether one given element belongs to a set of multiple primes $(l_0, l_1, \ldots, l_k)$, but breaks down if we wish to identify multiple primes raised to the same power. The first case corresponds to a "yes/no" voting scheme with two possible options while the other corresponds to an one-out-of-L election scheme with one ciphertext per ballot. Both cases are common elections' scenarios for which the proposed scheme's security guarantees are based on the well known DDH problem. However we should mention that both cases do not use the new problem's advantage, the compression of multiple ciphertexts.

Proof. Let A be an adversary against the above simplified problem that given (l_0, l_1, g, g^s, a) tries to decide which prime (if one) was used in value a, i.e. if $a = l_0^s$, $a = l_1^s$ or $a = g^t$, $r \leftarrow Z_q$.

- If A can distinguish (wlog) between l_0^s and g^t with non-negligible probability, then we have an adversary against DDH (see definition 1b 2.3, that is given $(g_1, g_2) = (g, l_0) \in G \times G$ we can decide if $(x_1, x_2) = (g^s, a)$ is sampled from (g^s, l_0^s) or from $G \times G$). Let p_{00}, p_{11} denote the probability that A identifies the l_i correctly and p_{0r}, p_{1r} the probability that A identifies the random element g^t as a power of l_0, l_1 respectively (where $p_{0r} = 1 - p_{1r}$). We have that $|p_{00} - p_{0,r}| = \mu$ is negligible (similarly for p_{11}, p_{1r}).
- Let us consider $|(p_{00} - \frac{1}{2}) + (p_{11} - \frac{1}{2})|$ as the adversary's advantage. If $p_{00} + p_{11} - 1 = 2\epsilon$, ϵ non-negligible, then he have that at least one of p_{00}, p_{11} must be larger than $1/2 + \epsilon$. Wlog let it be p_{00}. As it must hold that $\mu = |p_{00} - p_{0,r}| < \epsilon$, we get $p_{11} - p_{1r} = (1 + 2\epsilon - p_{00}) - (1 - p_{0r}) = 2\epsilon - \mu \geq \epsilon$ which implies an adversary for DDH.

□

In the above protocol the compression procedure done by the vote collector combines the encryptions of all options in a voter's ballot in a single cipher-text $c^* = (X^*, Y^*) = (X_1 X_2 \ldots X_k,$

Protocol 14 Shared-Key Code Verification E-Voting - Tallying

Let m be the number of submitted votes, k the maximum numbers of candidates per ballot and $\phi : C \rightarrow G^k$.

The Vote Collector:

- Takes as input tuples (V, e, σ) from each voter's PC, where the ballot $e = (c_1, ..., c_k)$ contains ElGamal ciphertexts.
- Extracts the votes computing $X^* = \prod_{i=1}^{k} X_i$ and $W^* = \prod_{i=1}^{k} W_i$, where $(X_i, W_i) = c_i$ and sends $c^* = (X^*, Y^*)$ to the tallier.

The Tallier:

- Selects a random permutation π on set $\{1, \ldots, m\}$ and computes $m^*_{\pi(i)} = Dec_{skt}(c^*_i)$ (implements a verifiable suffling protocol).
- For each element m^*_i computes $m_i = \phi(m^*_i)$ and publishes the outcome.

$Y_1 Y_2 \ldots Y_k) = (g^{\sum_{i=1}^{k} r_i}, pk_t^{\sum_{i=1}^{k} r_i} f(cnd_1) f(cnd_2) \ldots f(cnd_k))$. As the product of the encoded values is a product of small primes in L, the individual values can be recovered successfully using the mapping ϕ.

3.4.3 Security guarantees and weaknesses

The shared-key approach has significantly better online computational and communication complexity compared to the POT approach, as it does not depend on the number of candidates. However, both protocols have similar security guarantees in privacy and integrity aspects. They also suffer from the same drawbacks in the case of active attacks by coalitions of malicious servers. Again, the collaboration of the vote collector and the tallier breaches privacy either directly by decrypting votes due to the additive relation among the secret keys, or indirectly be revealing partial information. In the case that the messenger reveals its outcome $f(x)^{s_V}$ and the vote collector the secret exponent s_V then by comparing the outcome with all the possible candidates privacy is lost.

Unlike the POT approach, the return codes are computed based on the vote and the protocol should guarantee that the security codes computed by the messenger leak no information about the vote, i.e. an adversary cannot decide if the pre-codes contain $f(x)^s$ for any valid option

Game	(x_t, w_t)	(x_m, w_m)
Game 0	$(g^r, pk_t^r v)$	$(x_t^s, (w_t x_t^{a_2})^s) = (g^{rs}, pk_m^{rs} v^s)$
Game 1	$(g^r, x_t^{a_1} v) = (g^r, pk_t^r v)$	$(\gamma^r, x_m^{a_3} p_v) = (g^{rs}, pk_m^{rs} p_v)$
Game 2	$(g^r v^{r'}, x_t^{a_1} v) = (g^r v^{r'}, pk_t^r v)$	$(\gamma^r p_v^{r'}, x_m^{a_3} p_v) = (g^s v^{st'}, pk_m^{rs} v^{st' a_3} p_v)$
Game 3	$(g^r v^{r'}, x_t^{a_1} v) = (g^r, pk_t^r v)$	$(\gamma^r p_v^{r'}, x_m^{a_3} p_v) = (g^s v^{st'}, pk_m^{rs} v^{st' a_3} p_v) \mid p_v \leftarrow G$
Game 4	$(g^r, x_t^{a_1} v) = (g^r, pk_t^r v)$	$(\gamma^{r'}, x_m^{a_3} p_v) = (g^{st'}, pk_m^{st'} p_v)$
Game 5	$(g^r, pkt_t^r v)$	$(g^{r'}, pk_m^{t'} p_v)$
Game 6	(g^r, pkt_t^r)	$(g^{r'}, pk_m^{t'} p_v)$

Table 3.2: The PRF composition: Proof of messenger's security

$f(x)$ or completely random elements $t^s \in G$.

Proof. The proof goes through a series of indistinguishable games between an adversary that runs the messenger and a simulator that runs the entire protocol, having full knowledge of all public and secret keys. Indistinguishability is derived from the fact that all values, depicted in the table below, follow the same distribution. For the reduction we denote $\gamma = g^s$ and $p_v = v^s$ with $v = f(c)$ which we compute for all candidates c, s being the voter's exponent.

We begin with Game 0 which follows the protocol. In Game 1 we take advantage of the values γ, p_v (no longer using s) and the knowledge of the secret keys we posses to maintain the distribution. In Game 2 we maintain the (x, w)-relation (so does the distribution) but modify the x values exploiting the properties of the cyclic group. Game 3 is based of the alleged hardness of Prime DDH substituting $p_v = v^s$ with a random $p_v \leftarrow G$. Game 4 modifies the used randomness so that Game 5 ends up with a pre-code value $w_m = pk_m^{r'} r_v$ which is an independent encryption of a random value. In Game 6 the value p_v becomes unrelated to the value v from which it should be derived, unnoticed from the adversary. As p_v is random and independent of v, no knowledge about the vote is revealed to the adversary.

□

3.4.4 Avoiding Coalitions

In order to overcome the shortcomings of the previously discussed protocols Lipmaa [6] proposed a combination of the approaches in order to build a protocol with better security guarantees. The new protocol uses the pseudo-random composition approach, but separates the keys of the

entities to avoid the coalition privacy issue. As a step further towards this direction the new protocol moves the blinding step from the vote collector to the voter's PC. Lipmaa's protocol is an adaptation of Gjosteen's protocol with more details regarding the protocol's instantiation.

In this approach the security codes are constructed as compositions of three pseudo-random functions, similarly to section 3.4.1. The public encoding function $f : Cand \rightarrow G$ is instantiated as the keyed function $f_{k_1}(x) = g^{AES_{k_1}(x)}$, k_1 being its secret key, which should be given to each PC. A secret exponent s_V is chosen for each voter and is applied to the encoding in order to produce the security code. This is done by the voter's PC as well, which having s_V, computes and encrypts $f_{k_1}(x)^{s_V} = h_V^{AES_{k_1}(x)}$, where $h_V = g^{s_V}$. Finally the security code is computed and sent back to the voter by the messenger, who applies the final pseudo-random function $d_{k_2} : G \rightarrow Codes$, having its corresponding key k_2. Only the online phase of the protocol is discussed, as tallying is performed offline using a suitable mixing and decryption scheme. The full protocol is given in protocol 15.

The protocol requires a setup phase, where we need to rely on trusted servers who know the secret keys k_1, k_2 and the commitment $h_V = g^{s_V}$ for each voter, so that they can compute for each voter-candidate pair $\langle x, V\rangle$ the corresponding code: $Code_V[x] = d_{k_2}(h_V^{AES_{k_1}(x)}) = d_{k_2}((g^{s_V})^{AES_{k_1}(x)}) = d_{k_2}(f_{k_1}(x)^{s_V})$.

During vote submission phase, the vote encryption performed by the PC needs to prove that the corresponding plaintexts satisfy the correct blinding relation $g^{\mu}, (g^{\mu})^s$ being consistent with the announced commitment $C_V = g^{s_V}h^r$ that hides the blinding factor. Let the encryptions be $E_t = (e_{t1}, e_{t2}) = (g^x pk_t^{r_1}, g^{r_1})$ and $E_m = (e_{m1}, e_{m_2}) = (g^{x\cdot s_V} pk_m^{r_2}, g^{r_2})$. Then the latter can be written as $E_m = ((g^x pk_t^{r_1})^{s_V} \cdot pk_m^{r_2} \cdot pk_t^{-r_1\cdot s_V},\ (g^{r_1})^{s_V} \cdot g^{r_2 - s_V r_1}) = ((e_{t1}^{s_V})(pk_m^{r_2})(pk_t^{-r_1\cdot s_V}), (e_{t2}^{s_V})(g^{r_2})(g^{-r_1\cdot s_V})) = E_t^{s_V} \cdot Enc_{pk_t}^{-r_1\cdot s_V}(1) \cdot Enc_{pk_m}^{r_2}(1)$. Hence it suffices to create a proof $\pi = \mathrm{ZKP}(x, s_V, r_1, r_2, r_3, r_4 : E_t = Enc_{pk_t}^{r_1}(g^x) \wedge C_V = Com_h^{r_2}(g^{s_V}) \wedge E_m = E_t^{s_V} \cdot Enc_{pk_t}^{r_3}(1) \cdot Enc_{pk_m}^{r_4}(1))$. The proof is a genralization of the Schnorr protocol presented in protocol 16.

The protocol has same security guarantees to that of the previous code verification protocols regarding individual entities. The main advantage of the protocol is that it deals with the most serious threat of the previous protocols, the coalition between malicious vote collector and messenger. In this setting the two malicious entities would require the messenger's output $h_V^{AES_{k_1}(x)}$,

Protocol 15 PRF-Composition Enhanced Code Verification

- Voter V: Submits a vote for candidate x.
- The Voter's PC:
  Knows $s_V, f, k_1, sk_V, pk_m, pk_t$.
  1. Computes $E_t = Enc_{pk_t}(f_{k_1}(x))$, $E_m = Enc_{pk_m}(f_{k_1}(x)^{s_V})$
  2. Publishes Pedersen Commitment $C_V = g^{s_V} h^{r_v}$, where $h \in G$ is a public key and $r_V \leftarrow Z_q$.
  3. Creates an non interactive proof $\pi = PK(m_1, m_2, r_1, r_2, r_3 : E_t = Enc_{pk_t}^{r_1 \leftarrow Z_q}(g^{m_1}) \wedge E_m = Enc_{pk_m}^{r_2 \leftarrow Z_q}(g^{m_1 m_2}) \wedge C_V = Com_h^{r_3 \leftarrow Z_q}(g^{m_2}))$.
  4. Signs the contents $\sigma = Sign_{sk_V}(E_t, E_m, \pi)$ and sends (E_t, E_m, σ, π) to the vote collector.
- The Vote Collector:
  Knows $pk_t, pk_m, vk_V, (sk_{VC}, vk_{VC})$.
  1. Verifies π, σ and accordingly stores the vote, replacing previously submitted ballots from the same voter.
  2. Signs $\sigma' = Sign_{sk_{VC}}(E_m, \pi)$ and sends $(E_t, E_m, \pi, \sigma, \sigma')$ to the messenger.
  3. Waits until the election is over to forward E_t to the tallier.
- The Messenger:
  Knows $d, k_2, vk_{VC}, vk_V, vk_{VC}$.
  1. Verifies the signatures σ, σ' and the proof π.
  2. Decrypts $z = Dec_{sk_m}(E_m) = h_V^{AES_{k_1}(x)}$
  3. Computes Code$= d_{k_2}(z)$ and sends it through the secure post-channel to the voter.

Protocol 16 ZKP of consistent encryptions and commitments

$\pi = (\mu_1, \mu_2, r_1, r_2, r_3, r_4 : E_t = Enc^{r_1}_{pk_t}(g^{\mu_1}) \wedge C_V = Com^{r_2}_h(g^{\mu_2}) \wedge E_m = E_t^{\mu_2} \cdot Enc^{r_3}_{pk_t}(1) \cdot Enc^{r_4}_{pk_m}(1))$

1. The prover choses $m_1, m_2, \rho_1, \rho_2, \rho_3, \rho_4 \leftarrow Z_q$ and computes $i_1 = Enc^{\rho_1}_{pk_t}(g^{m_1})$, $i_2 = Com^{\rho_2}_h(g^{m_2})$, $i_3 = E_t^{m_2} \cdot Enc^{\rho_3}_{pk_t}(1) \cdot Enc^{\rho_4}_{pk_m}(1)$.
2. The verifier choses $c \leftarrow Z_q$.
3. The prover computes and sends $m'_1 = m_1 + c\mu_1$, $r'_1 = \rho_1 + cr_1$, $m'_2 = m_2 + c\mu_2$, $r'_2 = \rho_2 + cr_2$, $r'_3 = \rho_3 + cr_3$, $r'_4 = \rho_4 + cr_4$.
4. The verifier accepts only if $i_1 \cdot E_t^c = Enc^{r'_1}_{pk_t}(g^{m'_1})$, $i_2 \cdot C_V^c = Com^{r'_2}_h(g^{m'_2})$ and $i_3 \cdot E_m^c = E_t^{m'_2} \cdot Enc^{r'_3}_{pk_t}(1) \cdot Enc^{r'_4}_{pk_m}(1)$

the secret key k_1 and either the secret exponent s_V or the commitment h_V in order to violate privacy by comparisons for every possible candidate. The key k_1 is highly possible to leak, as it is common among all PCs. However, unlike the previous protocols, the vote collectors no longer possesses s_V and the proof of knowledge does not reveal h_V, which is perfectly hided due to the Pedersen commitment, making the coalition incapable of breaching privacy. In addition, as the protocols uses independent keys, the online entities cannot any longer decrypt votes directly.

Although this modification avoids the coalition attack, we should note that in this setting the vote collector has no active role in the code verification procedure, which has is moved to the voter's computer. The vote collector simply verifies proofs and signatures and then passes all its information to the messenger. Thus, there is no reason to have two entities if their combined knowledge does not affect the protocol's privacy, so we can merge them in a single entity, without undermining the protocol.

We conclude our discussion about code verification protocols by comparing the approaches proposed so far, summarised in table 3.3 in terms of building blocks, efficiency and security guarantees for N candidates.

	Proxy Oblivious Transfer	Shared Key Approach	Lipmaa's Protocol
Code generation	random elements	prf composition	prf composisiton
Components	$R_{i,V} \in Z_q$ $H : G \rightarrow Codes$	$f : Cand \rightarrow G$ $s_V \in Z_q$ $d_V : G \rightarrow Codes$	$f : Cand \rightarrow G$ $s_V \in Z_q$ $d : G \rightarrow Codes$
Encoding	$f(x) = g^x$	any f	$f(x) = g^{AES_{k_1}(x)}$
Code mapping	H: hash	d_V: prf	d: prf
Code format	$H(g^{R_{i,V}})$	$d_V(f(x)^{s_V})$	$d_{k_2}(g^{s_V \cdot AES_{k_1}(x)})$
Key generation	independent keys $(sk_m, pk_m), (sk_t, pk_t)$	shared keys $sk_m = sk_t + sk_{vc}$	independent keys $(sk_m, pk_m), (sk_t, pk_t)$
Security against	individual entities	individual entities	coalitions
PC's Complexity	8 encr, 1 sing	3 encr, 1 sign	16 encr, 1 sign
VC's Complexity	$2N + 6$ encr, 1 ver, 1 sign	9 encr, 1 ver, 1 sign	13 encr, 1 ver, 1 sign
MS's Complexity	N encr, 1 ver	10 encr, 1 ver	14 encr, 2 ver

Table 3.3: Comparison of the code verification protocols

Chapter 4

A New Vote Verification Protocol

In this chapter we propose a new Internet voting protocol that provides the voter with a receipt of her vote, i.e. a vote-verification protocol. In this approach the voter verifies that the her vote was successfully submitted to the electronic ballot box and was recorded as cast. Our protocol consists of a set of voters, voting through their personal computers, and online servers for vote submission and verification. The function of the tallying servers that decrypt and define the outcome after the election period is over is out of scope of this paper and can be implemented by using any suitable tallying protocol. Similar to previously proposed protocols [1] [2] [6] we require the existence a public key infrastructure for voter authentication purposes, assuming that each computer can digitally sign ballots and the signatures can be verified by the voting servers. In addition we require the existence of an untappable real-time communication channel, usually implemented as an SMS, for verification purposes, which is a stronger, physical assumption compared to the out-of-band channels of previous approaches that allowed an attacker to read the contents of at most one channel, without compromising vote secrecy. We also present a natural extension of our protocol into the code verification setting, relaxing the need for the untappable channel. In this case we require the existence of two secure out-of-band communication channels that are not likely to be both corrupted, similarly to the previous solutions. Finally we propose an adaptation of the vote verification protocol into a visual vote verification protocol, that uses images as receipts of the votes.

Our first scheme differs from the previous solutions in two main aspects: the need for security codes and the role of the online entities. Our protocol does not require any phase for code generation and distribution. Although this part was executed off-line, prior to the elections in

the previous protocols, they need trusted servers and printers to generate the codes, a secure out-of-band communication channel for code delivery, usually implemented through paper mail, and additional cryptographic operations and encrypted channels to distribute the essential data for code reconstruction to the voting servers. In our scheme the voter will be able to reconstruct the vote itself, using no security codes, and thus verify her vote. We assume that the voter is capable of executing simple calculations like addition of two-digit numbers (or at least she is able to verify the correctness of the calculation if she assigns it to a computer), which are reasonable assumptions for any user participating in Internet voting and having access to a computer. In order to prevent the voter from coercion we allow re-voting that counts only the last submitted ballot on behalf of a voter.

Second, our protocol makes no distinction between the online voting servers, having no dedicated vote collector and messenger servers. It uses two or more identical voting servers, depending on our privacy concerns, that serve both as vote collectors and messengers, each of whom provides the voter with a share of her vote. By including additional online servers in our setting we overcome the main drawback of previous protocols, where the collaboration of the online servers could violate voter's privacy. By employing a simple k-out-of-k secret sharing scheme we achieve easy vote verification on the voter's part and perfect vote secrecy on the servers' part, as long as one server remains honest.

Our proposed scheme is ideal for small scale elections as it can be set-up to run elections instantly, eliminating code generation and distribution phases. Furthermore, our protocol is flexible as we may include any number of voting server's to enhance privacy against coalitions of malicious servers, unlike the previous protocols that strictly separate the role of the online servers and cannot be naturally generalized.

Extensions. Our protocol can be easily transformed into a code verification protocol with a minimum number of modifications. In this approach we ask for a secure code generation phase before the elections as well as for two secure communication channels, which we will call the pre-channel and post-channel from now on. These channels, that by-pass the the possibly malicious PC, are used to distribute the codes to the voter prior to the election and forward the receipts after vote-submission. They are usually implemented as paper mail and SMS. Based on this infrastructure we propose two extensions of our protocol into a code verification protocol: the first functions similarly to our original protocol during vote submission and asks the

voter to reconstruct her receipt, while the second introduces an additional, untrusted entity that possesses no secrets and reconstructs the code for the voter, who simply checks that is matches her choice. The two adaptation differ in re-voting, with the first excluding re-voting as a trade-off for privacy, while the second supports it, taking advantage of the additional entity.

The idea of our vote-verification protocol can be adapted in a setting that uses visual cryptography techniques for verification purposes. Visual cryptography was introduced by Shamir and Naor who designed the first visual secret sharing protocol [27], for black and white images. Motivated by their construction we present a 2-server vote-verification protocol that allows the voter verify her vote by overlaying two black and white images, which leak no information about the vote when separated, but reveal the vote when combined.

4.1 The vote verification protocol

We define the notion of security of our scheme in terms or privacy and integrity. Throughout our discussion will refer to malicious entities. We clarify that a malicious PC wants to violate integrity by modifying the vote, as it already knows the vote, while malicious voting servers wish to violate privacy by learning the vote, as by their construction they cannot substitute or alter encrypted and signed submitted ballots. Specifically we ask that the following requirements are met:

- Cast as intended: A malicious PC cannot submit a forged ballot x' on behalf of an honest voter voting for x without being detected by the voter.
- Vote secrecy/Voter privacy: For a k server instantiation of the protocol ($k \geq 2$) any coalition of up to $k-1$ servers gets absolutely no information about the submitted vote.

Definition 4.1 (Cast as intended). Let us consider the following game.

Game1:

1. Let A and C denote two entities, an adversary that is given access to the public keys, voter IDs (and possibly any code sheets Csh if any) and a challenger that runs the voting protocol.
2. We allow A to pick a voter ID and corrupt his PC.

3. Let voter ID cast a ballot for option x.

4. Then C runs the voting protocol and outputs the encrypted vote E, the secret receipt R and all other necessary public information announced $PUBLIC(x) = (c_1(x), \ldots, c_k(x))$.

5. Let the predicate $V(R, PUBLIC, (Csh)) = 1$ if the receipt R generated by C is consistent with the PUBLIC information announced, $V(R, PUBLIC, (Csh)) = 0$ otherwise.

6. A wins the game if $V(R, PUBLIC, (Csh)) = 1$ and $Dec(E) \neq x$.

We say that the protocol satisfies "Cast as intended" if $Pr[A()$ wins$] \leq \epsilon$, where ϵ is a negligible function.

Definition 4.2 (Voter Privacy). Let us consider the following game.

Game2:

1. Let A and C denote two entities, an adversary that is given access to the public keys, voter's voting servers's IDs and a challenger that runs the voting protocol.

2. We allow A to pick and corrupt ut to $k-1$ voting servers.

3. Then A picks a voter ID and two options x_0, x_1 of his choice, which we provides to the corrupted PC of the voter.

4. Then C runs the voting protocol, picking at random a bit $b \leftarrow \{0, 1\}$ and encrypting $E = Enc(x_b)$. Then he outputs the encrypted vote E, the secret receipt R, all other necessary public information $PUBLIC(x) = (c_1(x), \ldots, c_k(x))$ and sends E to the voting servers.

5. Then A in possession of the information PUBLIC and the private values of at most $k-1$ server's that controls tries to learn the value x_b that was submitted, outputting a bit b^*.

6. A wins the game if $b^* = b$.

We say that the protocol satisfies "vote secrecy/voter privacy" if $Pr[A()$ wins$] \leq 1/2 + \epsilon$, where ϵ is a negligible function.

4.1.1 The main idea

We are now ready to describe the vote verification protocol. For simplicity we discuss the construction that uses two voting servers, say A_1 and A_2, however our construction is generalized naturally to any number of voting severs, say $k \geq 2$. We assume that the N candidates participating in the elections are represented as elements in $[0, N-1]$ and their values are globally known by the voters.

Our protocol uses the additive version of the ElGamal crypto-system and Pedersen commitments over a subgroup $G \subset Z_p$ for prime order q, generated by $\langle g \rangle$, where p, q are large primes such that $q|p-1$. Our message space is Z_m defined by the value m that characterizes the system. We chose m so as to facilitate the vote reconstruction and verification by the voter. Specifically we chose m to be the smallest power of 10 such that $N \leq m < q$. As we consider small scale elections with at most a few hundred options in total, typical values for m will be 100 or 1000. By this trick we avoid the modular operation that would normally require the vote verification step which boils down to simple addition of two two-digit or three-digit numbers by the voter. By introducing k voting servers, $(2 \leq k < q)$ the voter needs to add the corresponding k numbers.

During vote submission a voter casts her ballot through her PC voting for candidate x. Then the PC splits the vote by picking two random values x_1, x_2 such that $x = x_1 + x_2 \bmod m$, using the simplest secret sharing form, with the PC being the dealer and the servers the share-holders such that the voter can easily reconstruct the secret. Then PC encrypts the vote, computes commitments to the shares x_1, x_2 and sends them to the online servers. In addition it sends to each online server A_i the opening of the commitment $C_i = Com(x_i)$ for verification.

The online servers A_i need to verify the additive relation between the submitted vote and the committed values. Our protocol uses Pedersen commitments $Com_{g,h}(x) = g^x h^r$ for $r \leftarrow Z_q$. Let $E_t = (C_x, C_r) = (g^x pk_t^r, g^r)$ be the encryption of the vote under the tallier's public keys pk_t and $C_1 = g^{x_1} h^{r_1}, C_2 = g^{x_2} h^{r_2}$ be the commitments to shares x_1, x_2 possessed by the severs. Thus, server A_1, which holds share x_1, needs to verify that his share x_1, the committed value x_2 and the encrypted vote x satisfy the relation $x = x_1 + x_2 \bmod m$. As $x_1, x_2 \in Z_m$ then for their sum it must hold that either $0 \leq x_1 + x_2 \leq m-1$ or $0 \leq x_1 + x_2 - m \leq m-1$ and thus we need to prove that the vote's ciphertext C_x, which

	Public Input: g, h_1, h_2 $Com_1 = g^x h_1^{r_1}, Com_2 = g^x h_2^{r_2}$	
Prover Private input: x, r_1, r_2 $w, \rho_1, \rho_2 \xleftarrow{\mathtt{r}} Z_q$ $y_1 = g^w h_1^{\rho_1},\ y_2 = g^w h_2^{\rho_2}$		Verifier
	$\xrightarrow{y_1,\ y_2}$	
		$c \xleftarrow{\mathtt{r}} Z_q$
	$\xleftarrow{c}$	
$s = w + cx$ $\rho_1' = \rho_1 + cr_1$ $\rho_2' = \rho_2 + cr_2$	$\xrightarrow{s,\ \rho_1,\ \rho_2}$	$g^s h_1^{\rho_1'} \stackrel{?}{=} y_1(Com_1)^c$ $g^s h_2^{\rho_2'} \stackrel{?}{=} y_2(Com_2)^c$

Figure 4.1: ZKP Equality of committed values

can be viewed as a commitment with public key pk_t, contains the same value with either $C_{sum} = C_1 \cdot C_2 = g^{x_1+x_2} h^{r_1+r_2}$ or $C'_{sum} = g^{x_1+x_2-m} h^{r_1+r_2}$. Thus we need to built a zero knowledge proof $\pi_{sum} = PK(x, y, r_x, r_y :\ C_x = g^x pk_t^{r_x} \wedge C_y = g^y h^{r_y} \wedge (y = x \vee y = x+m))$ (figure 4.2) which is based on the proof for equal committed values (figure 4.1) by adapting the technique of [17]. The server A_2 functions similarly.

Upon successful verification server A_i sends x_i to the voter through the untappable channel. Finally the voter verifies her vote by checking that the received values satisfy the relation $x = x_1 + x_2$ mod m. After the end of the vote submission phase, one server, which may be predefined or selected by the tallier, forwards its contents to the tallier server to produce the outcome.

4.1.2 Commitments' announcement

Upon successful verification of the zero knowledge proof that the encrypted vote, the possessed share and the committed share satisfy the additive relation, the server accepts the vote as valid and stores it. We avoid to use a broadcast channel from the PC to the voting servers since we try to keep the construction simple, making no special assumptions about the environment. However this setting has a potential threat in the case that a malicious PC attempts to violate

Prover Private input: x, x_1, x_2, r_x, r_y	Public Input: g, h_1, h_2, m $Com_x = g^x h_1^{r_x}, Com_y = g^{x_1+x_2} h_2^{r_y}$	Verifier
If $x = x_1 + x_2$: $w, \rho_a, \rho_b, c_2, s_m, \rho_{m_1}, \rho_{m_2} \xleftarrow{r} Z_q$ $a = g^w h_1^{\rho_a}, b = g^w h_2^{\rho_b}$ $a_m = (Com_x)^{-c_2} g^{s_m} h_1^{\rho_{m_1}}$ $b_m = (\frac{Com_y}{g^m})^{-c_2} g^{s_m} h_2^{\rho_{m_2}}$ If $x = x_1 + x_2 - m$: $w, \rho_a, \rho_b, c_1, s, \rho_1, \rho_2 \xleftarrow{r} Z_q$ $a = (Com_x)^{-c_1} g^s h_1^{\rho_1}$ $b = (Com_y)^{-c_1} g^s h_2^{\rho_2}$ $a_m = g^w h_1^{\rho_a},\ b_m = g^w h_2^{\rho_b}$		
	$\xrightarrow{a,\ b,\ a_m,\ b_m}$	
		$c \xleftarrow{r} Z_q$
	$\xleftarrow{c}$	
If $x = x_1 + x_2$: $c_1 = c - c_2, s = w + xc_1$ $\rho_1 = \rho_a + r_x c_1, \rho_2 = \rho_b + r_y c_1$ If $x = x_1 + x_1 - m$: $c_2 = c - c_1, s_m = w + xc_2$ $\rho_{m_1} = \rho_a + r_x c_2, \rho_{m_2} = \rho_b + r_y c_2$		
	$\xrightarrow{c_1, c_2, s, s_m, \rho_1, \rho_2, \rho_{m_1}, \rho_{m_2}}$	$c \stackrel{?}{=} c_1 + c_2$ $g^s h_1^{\rho_1} \stackrel{?}{=} a(Com_x)^{c_1}$ $g^s h_2^{\rho_2} \stackrel{?}{=} b(Com_y)^{c_1}$ $g^{s_m} h_1^{\rho_{m_1}} \stackrel{?}{=} a_m(Com_x)^{c_2}$ $g^{s_m} h_2^{\rho_{m_2}} \stackrel{?}{=} b_m(\frac{Com_y}{g^m})^{c_2}$

Figure 4.2: Proof $\pi_{sum} = \text{ZPK}(x, y, r_x, r_y :\ C_x = g^x h_1^{r_x} \wedge C_y = g^y h_2^{r_y} \wedge (y = x \vee y = x + m))$

integrity by preparing encryptions and valid shares of two different votes. In this scenario we assume that the voter submits a vote for candidate x but the PC wants to submit a vote for candidate $y \neq x$ without being detected by the voter. The PC encrypts y, computes the valid shares $x_1 + x_2 = x \bmod m$ for the real vote and sends to servers inconsistent commitments, giving A_1 the tuple $(Enc_{pk_t}(y), x_1, Com_{g,h}(y-x_1))$ and A_2 $(Enc_{pk_t}(y), x_2, Com_{g,h}(y-x_2))$. Both tuples are accepted as valid, satisfying the additive relation, and the servers forward the shares x_1, x_2 to the voter who accepts the verification check.

In order to face this issue each server needs to verify that the commitment he holds is compatible with the shares given to the other server. Thus we establish communication between the online servers that exchange the commitments they hold. As we have the PC open the commitment of C_i to the server A_i, the latter verifiers that the commitments are valid for the share x_i he holds. Hence the PC sends to servers A_1, A_2 the tuples $(Enc_{pk_t}(x), x_1, r_1, C_1, C_2)$, $(Enc_{pk_t}(x), x_2, r_2, C_1, C_2)$ respectively, where r_1, r_2 are the openings to commitments C_1, C_2. Then the online servers exchange the commitments they hold and test that $C_1 = g^{x_1}h^{r_1}, C_2 = g^{x_2}h^{r_2}$. If the values do not match the servers detect a forgery attempt.

4.1.3 Range proof

A malicious computer has another way to alter the submitted ballot undetected. Although the PC is no longer capable of voting for another candidate, the system is vulnerable to randomization attacks, where the PC may try to de-validate the vote and submit a random value. In theis scenario that the PC, instead of the value x given by the voter, casts a vote for value $y = x + m$. By submitting this value y that belongs in Z_q and picking shares $y_1, y_2 \in Z_m$ that satisfy $y = y_1 + y_2 \bmod m$ and $y_1 + y_2 \notin Z_m$, the PC will generate a fake vote whose shares pass the proof π_{sum}, as it will be the case that $g^{y_1+y_2}h_2^{r_y}$ is consistent with the commitment $C'_{sum} = g^{x+m}h_1^{r_x} = g^y h_1^{r_x}$. Moreover the voter will accept the vote as valid, as the votes are equivalent modulo m. This attack can be prevented by checking that the submitted vote is a valid vote by performing a range proof $\pi_R = ((x, r_x) : C_x = g^x pk_t^{r_x} \wedge (0 \leq x \leq N-1))$.

To show our statement we employ the range proof in exponents presented in [24]. The proof is based on the fact that any number $x \in [0, N-1]$ can be written in the form $x = \sum_{j=0}^{\lfloor log_2(N-1) \rfloor} \mu_i H_j$, where $H_j = \lfloor (N-1+2^j)/2^{j+1} \rfloor$ and $\mu_i \in \{0, 1\}$. Then it commits to all

values μ_j and uses a standard Schnorr OR proof (see figure 4.11) to show that $\mu_j \in \{0, 1\}$, requiring $log_2 N$ single bit proofs. However, for small values of N the proof remains efficient for our purpose. Both the prover and the verifier precompute the coefficients H_j and the verifier can confirm that the committed values μ_j represent μ by checking that $g^{\mu} = \prod_{j=0}^{\lfloor log_2(N-1) \rfloor} (g^{\mu_j})^{H_j}$. The full scheme is shown in figure 4.3.

4.1.4 Adding more voting servers

The protocol can be generalized by adding multiple servers to enhance voter privacy, say k servers, as long as $2 \leq k < q$. In this case the PC uses a k-out-of-k secret sharing scheme, by picking k values x_i such that $x = \sum_{i=1}^{k} x_i \bmod m$ and send each x_i to the corresponding server A_i. Commitments C_i of all shares x_i are computed and sent to all servers along with the opening of C_i which is given to server A_i. The range proof that the encrypted vote x belongs in $[0, N-1]$ remains the same, while the additive relation proof is a generalization of figure's 4.2 proof, stating that $\pi_{sum} = \text{ZPK}(x, y, r_x, r_y : \ C_x = g^x pk_t^{r_x} \wedge C_y = g^y h^{r_y} \wedge (y = x \vee y = x + m \vee y = x + 2m \vee \cdots \vee y = m + (k-1)m))$. The full proof π for the 2-server construction is given in protocol 17 and the voting protocol in protocol 18 and figure 4.4.

4.1.5 Security guarantees and performance

In this section we analyze the security guarantees in terms of privacy and integrity as well as the protocol's overall efficiency. We guarantee that the protocol meets our security requirements in the following corruption scenarios:

- The voter's PC is malicious.
- A subset of $k-1$ (or less) out of k voting servers are honest-but-curious, i.e. they follow the protocol but share their information with an attacker.

In the above cases if the voter does not complain about a forgery (due to wrong candidate reconstruction) and does not re-vote, then she can be sure that her original vote will be counted in the final outcome. In both cases the vote remains perfectly secure. However we cannot provide any guarantees against a coalition of a malicious PC with a malicious voting server that deviates from the protocol, since in this strong corruption scenario fake ballots can

	Public Input: g, h, N $\nu = \lfloor log_2(N-1) \rfloor, C = g^x h^r$	
Prover Private input: $x \in [0, N-1], r$ Computes $\mu_j \in \{0,1\}$ s.t. $x = \sum_{j=0}^{\nu-1} \mu_j H_j$ For $j = 0, \ldots, \nu-1$: Pick $r_j \leftarrow Z_q$ s.t. $\sum_{j=0}^{\nu-1} r_j H_j = r$ $C_j = g^{\mu_j} h^{r_j}$ Case $\mu_j = 0$: $w_j, c_{2j}, \rho_{2j} \leftarrow Z_q$ $y_{1j} = h^{w_j},\ y_{2j} = h^{\rho_{2j}}(C_j/g)^{-c_{2j}}$ Case μ_j=1: $w_j, c_{1j}, \rho_{1j} \leftarrow Z_q$ $y_{1j} = h^{\rho_{1j}}(C_j)^{-c_{1j}},\ y_{2j} = h^{w_j}$	$\xrightarrow{\{C_j, y_{1j}, y_{2j}\}_{j=0}^{\nu-1}}$ $\xleftarrow{c}$	Verifier $c \leftarrow Z_q$
Case $\mu_j = 0$: $c_{1j} = c - c_{2j}, \rho_{1j} = w_j + c_{1j} r_j$ Case $\mu_j = 1$: $c_{2j} = c - c_{1j}, \rho_{2j} = w_j + c_{2j} r_j$	$\xrightarrow{\{c_{1j}, c_{2j}, \rho_{1j}, \rho_{2j}\}_{j=0}^{\nu-1}}$	$C \stackrel{?}{=} \prod_{j=0}^{\nu-1} C_j^{H_j}$ For $j = 0, \ldots, \nu-1$: $c \stackrel{?}{=} c_{1j} + c_{2j}$ $h^{\rho_{1j}} \stackrel{?}{=} y_{1j}(C_j)^{c_{1j}}$ $h^{\rho_{2j}} \stackrel{?}{=} y_{2j}(C_j/g)^{c_{2j}}$

Figure 4.3: Range proof in Pedersen commitments

Protocol 17 The full ZKP for the vote-verification protocol

Public Input: $\langle p, q, g, m\rangle$ the system parameters, h, pk_t the commitment key and the tallier's public key, N the number of candidates and $\nu = \lfloor log_2(N-1)\rfloor$, $E_t = (E_x, E_r) = (g^x pk_t^r, g^r), C_1 = g^{x_1}h^{r_1}, C_2 = g^{x_2}h^{r_2}$.

Prover's Input: x, x_1, x_2, r, r_1, r_2.

1. The Prover:
   - Computes $\mu_j \in \{0,1\}$ s.t. $x = \sum_{j=0}^{\nu-1} \mu_j H_j$ where $H_j = \lfloor (2^j + N - 1)/2^{j+1}\rfloor$
   - For $j = 0, \ldots, \nu-1$:
     - Picks $r_j \leftarrow Z_q$ s.t. $\sum_{j=0}^{\nu-1} r_j H_j = r$.
     - Commits to μ_j as $\mathcal{E}_j = g^{\mu_j} pk_t^{r_j}$.
     - If $\mu_j = 0$ he picks $w_j, c_{2j}, \rho_{2j} \leftarrow Z_q$ and sets $y_{1j} = pk_t^{w_j}$, $y_{2j} = pk_t^{\rho_{2j}}(\mathcal{E}_j/g)^{-c_{2j}}$.
     - if $\mu_j = 1$ he picks $w_j, c_{1j}, \rho_{1j} \leftarrow Z_q$ and sets $y_{1j} = pk_t^{\rho_{2j}}(\mathcal{E}_j)^{-c_{1j}}$, $y_{2j} = pk_t^{w_j}$.
   - If $x = x_1 + x_2 \bmod q$ he picks $w, \rho_a, \rho_b, c_2, s_m, \rho_{m_1}, \rho_{m_2} \leftarrow Z_q$ and sets $a = g^w pk_t^{\rho_a}, b = g^w h^{\rho_b}$, $a_m = (E_x)^{-c_2} g^{s_m} pk_t^{\rho_{m_1}}$, $b_m = (C_1C_2/g^m)^{-c_2} g^{s_m} h^{\rho_{m_2}}$.
   - Else if $x = x_1 + x_2 - m \bmod q$ he picks $w, \rho_a, \rho_b, c_1, s, \rho_1, \rho_2 \leftarrow Z_q$ and sets $a = (E_x)^{-c_1} g^s pk_t^{\rho_1}$, $b = (C_1C_2)^{-c_1} g^s h^{\rho_2}$, $a_m = g^w pk_t^{\rho_a}$, $b_m = g^w h^{\rho_b}$
   - He sends $(a, b, a_m, b_m, \{\mathcal{E}_j, y_{1j}, y_{2j}\}_{j=0}^{\nu-1})$ to the Verifier.
2. The Verifier picks $c \leftarrow Z_q$ and sends it to the Prover.
3. The Prover:
   - For $j = 0, \ldots, \nu-1$:
     - If $\mu_j = 0$ he sets $c_{1j} = c - c_{2j}$, $\rho_{1j} = w_j + c_{1j} r_j$.
     - if $\mu_j = 1$ he sets $c_{2j} = c - c_{1j}$, $\rho_{2j} = w_j + c_{2j} r_j$.
   - If $x = x_1 + x_2 \bmod q$ he sets $c_1 = c - c_2$, $s = w + xc_1$, $\rho_1 = \rho_a + rc_1$, $\rho_2 = \rho_b + (r_1 + r_2)c_1$.
   - Else if $x = x_1 + x_2 - m \bmod q$ he sets $c_2 = c - c_1$, $s_m = w + xc_2$, $\rho_{m_1} = \rho_a + rc_2$, $\rho_{m_2} = \rho_b + (r_1 + r_2)c_2$.
   - He sends $(c_1, c_2, s, s_m, \rho_1, \rho_2, \rho_{m_1}, \rho_{m_2}, \{c_{1j}, c_{2j}, \rho_{1j}, \rho_{2j}\}_{j=0}^{\nu-1})$ to the Verifier.
4. The Verifier accepts if all the following tests succeed, otherwise he rejects:
   - $E_x = \prod_{j=0}^{\nu-1} \mathcal{E}_j^{H_j}$.
   - $c = c_1 + c_2$ and $g^s pk_t^{\rho_1} = a(E_x)^{c_1}$ and $g^s h^{\rho_2} = b(C_1C_2)^{c_1}$ and $g^{s_m} pk_t^{\rho_{m_1}} = a_m(E_x)^{c_2}$ and $g^{s_m} h^{\rho_{m_2}} = b_m(C_1C_2/g^m)^{c_2}$.
   - For $j = 0, \ldots, \nu-1$: $c = c_{1j} + c_{2j}$ and $pk_t^{\rho_{1j}} = y_{1j}(\mathcal{E}_j)^{c_{1j}}$ and $pk_t^{\rho_{2j}} = y_{2j}(\mathcal{E}_j/g)^{c_{2j}}$.

Protocol 18 The Splitting Vote-Verification Protocol

Let N be the number of candidates, $(pk_{A_i}, sk_{A_i}), (pk_t, sk_t)$ be the public/secret key pairs of server A_i ($\forall\ i = 1, \ldots, k$) and the tallier respectively, (sk_V, vk_V) and (sg_{A_i}, vk_{A_i}) be the signing/verification key pairs of voter V and server A_i, $\langle g \rangle$ the group generator of G and h the commitment public key.

- The voter V:
  1. Submits a vote for candidate x.
  2. Waits for shares $x_1, \ldots, x_k$ from the servers and checks that $x = x_1 + \cdots + x_k \bmod m$.
- The PC $(sk_V, pk_t, \{pk_{A_i}\}_{i=1}^k)$:
  1. Picks $x_1, \ldots, x_{k-1} \leftarrow Z_m$ and sets $x_k = x - \sum_{j=1}^{k-1} x_k \bmod m$.
  2. Encrypts x as $E_t = Enc_{pk_t}^{r \leftarrow Z_q}(g^x)$.
  3. For all $i = 1, \ldots, k$ encrypts x_i and commits to it as $e_i = Enc_{pk_{A_i}}^{\rho_i \leftarrow Z_q}(x_i)$ and $C_i = g^{x_i} h^{r_i}$ respectively, with $r_i \leftarrow Z_q$, and encrypts the randomness $R_i = Enc_{pk_{A_i}}^{r'_i \leftarrow Z_q}(r_i)$.
  4. Produces zero-knowledge proof $\pi = (x, r, \{x_i\}_{i=1}^k, \{r_i\}_{i=1}^k \mid E_t = Enc_{pk_t}^{r \in Z_q}(g^x) \wedge \{C_j = g^{x_i} h^{r_i}\} \wedge x = \sum_{i=1}^k x_i \bmod m \wedge x \in [0, N-1])$.
  5. Signs the vote $\sigma = Sing_{sk_V}(E_t, \pi)$.
  6. For all $i = 1, \ldots k$ sends to Server A_i $(E_t, e_i, R_i, \{C_j\}_{j=1}^k, \pi, \sigma, V)$.
- Server A_i $(sg_{A_i}, sk_{A_i}, vk_V)$:
  1. Sends C_j to all servers A_j, $j \neq i$, and receives C_i from them. Verifies that all C_i values are the same.
  2. Decrypts r_i, x_i, and verifies that it is a valid opening of $C_i = g^{x_i} h^{r_i}$.
  3. Verifies the proof π and the signature σ.
  4. Upon success of all verifications he signs the vote E_t, $\sigma' = Sign_{sg_{A_i}}(E_t)$, stores it and sends x_i to the voter V through the secure post-channel. If any of the verification steps fails he stops and notifies the voter about a forgery.
- The Tallier:
  - When the election is over the tallier gets the signed votes from a predefined sever, verifies the server's signatures and runs a suitable decryption protocol.

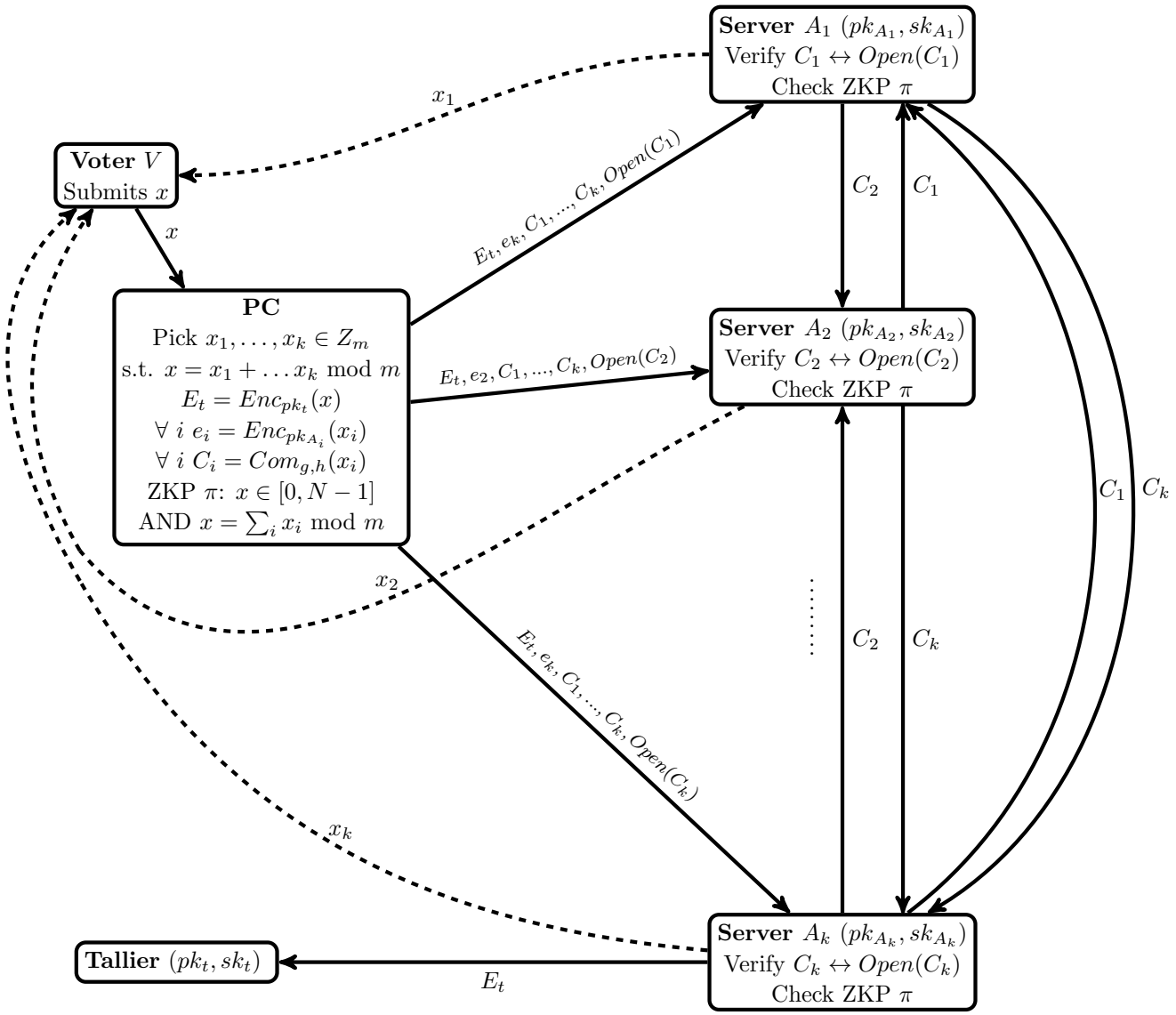

Figure 4.4: Overview of the splitting-vote verification protocol

be successfully submitted undetected. No attacker can access the untappable post-channel by default.

Privacy guarantees. Regarding vote privacy the PC sees the submitted vote like in all previously proposed protocols, which happens inevitably to avoid approaches like code voting. Each online server obtains a share x_i of the vote which reveals no useful information, being randomly chosen. Each server that sees two shares $x_i, x'_i \in Z_m$ that correspond to two not necessarily different candidates $x = x_i + \sum_{i \neq j} x_j$ and $x' = x'_i + \sum_{i \neq j} x'_j$, cannot distinguish between x_i, x'_i. This property implies that even in case of re-voting the servers cannot distinguish if the voter votes for a new candidate of not, unlike the previous protocols, since shares x_i for the same vote in two different submission are unrelated and random. Hence if each entity executes each part honestly no information regarding the vote is leaked.

Example 4.1. Let us consider an example of small scale elections with 9<10 candidates. In the billowing table we see the values seen by the voting servers and how they hide they cannot be correlated on their own with the actual votes.

Server 1	5	4	9	6	2	5	1
Server 2	4	7	3	7	8	9	2
Server 3	2	3	9	5	4	4	5
Actual votes	1	4	1	8	4	8	8

By using a simple k-out-of-k secret sharing scheme to split the vote privacy is breached only in the case of coalition of all online servers who can combine their shares and get the initial vote. As we do not distinguish between the voting servers, by adding multiple servers we overcome the main vote secrecy drawback of the previous protocols. The scheme guarantees that as long as one voting server remains honest the vote is perfectly secure. We do not examine coalitions between voting server's and personal computers in terms of integrity since in any vote/code verification protocol the PC can trivially break privacy by announcing the vote it sees.

Integrity guarantees. Regarding vote integrity, which is the main target in the presence of untrusted platforms, our protocol guarantees that the PC is incapable of altering a vote undetected. The opening of the commitments combined with the range proof prevents the

PC from modifying the vote in a way that will lead in a successful vote reconstruction on the voter's side. Any other attempt of altering the vote will be detected through the reconstruction of a wrong candidate number. No malicious voting server can submit fake ballots on behalf an honest voter, as votes are signed. Nonetheless all voting protocols that allow re-voting, in order avoid coercion, cannot guarantee that the vote collector forwards the correct vote to the tallier. Thus in case a corrupted voting server forwards the votes to the tallier we have no means to check that he sends the most recently submitted vote on behalf of the voter. Without re-voting any verifiable shuffle cryptographic scheme would be enough to guarantee the integrity of the outcome.

In the corruption scenario where a malicious PC collaborates with a malicious server that deviates arbitrarily from the protocol, a forged ballot y can be submitted instead of x, by sending to the honest servers the fake ballot y accompanied by valid commitments to the shares y_i. The corrupted server should ignore all the necessary checks and forward the share $x_i^* = x - \sum_{j=1, j\neq i}^{k} y_j \bmod m$ to the voter, with y_j being the shares of the forged ballot stored by the honest servers. We underline that this is a strong corruption scenario requiring the full collaboration of a malicious PC and the voting server, which is also present in all previous approaches, where the collaboration of a malicious PC and messenger can submit fake ballots. Unfortunately we cannot eliminate this attack by adding more servers, since as long as one malicious server A_i^* deviates from the protocol the voter obtains the correct value, while a fake ballot is submitted.

4.1.6 Complexity analysis

Below we give a summary of the complexity of the k-server protocol, ($k \geq 2$) counting the number of online exponentiations, signatures and signature verifications that each entity performs.

- The PC: Encrypts the vote x as well as each share x_i, $i = 1, \ldots, k$ and computes the Pedersen commitments of the shares and encryptions of their opening, requiring $6k + 2$ exponentiations. To create the valid range proof the PC commits to all $\nu = \lfloor log_2(N-1) \rfloor$ components of x and runs ν single bit proofs requiring $4\lfloor log_2(N-1) \rfloor$ exponentiations, depending on the number N of the candidates. Finally the valid shares proof requires 3 exponentiations for the commitment step and $5(k-1)$ exponentiations for the simulated

steps. All the above yield an overall complexity of $4\lfloor log_2(N-1)\rfloor + 11k$ exponentiations. In addition the PC needs to sign one message.

- Server A_i: Decrypts two values and verifies a commitment with 4 exponentiations. Running the verification part of the range proof requires 4ν exponentiations for the single bit proofs plus ν exponentiations for verifying the validity of the representation. The verifications part of the valid shares' proof requires $5k$ exponentiations, giving a total of $5\lfloor log_2(N-1)\rfloor + 5k + 4$ exponentiations for each voting server. Additionally the sever performs one signing and one signature verification.

4.2 Extension to code verification

4.2.1 The trusted channels and the security codes

Our proposed protocol is ideal for small scale elections since it can be used immediately for voting, requiring no previous registration and distribution of security codes, if we have access to an untappable channel, i.e. a channel that cannot be either read or written by an attacker. All existing code verification protocols [1] [2] demand a secure post-channel where the attacker has no "write" access, to eliminate forged ballot submission by overtaking a PC and the post-channel. However, since the channel was used to post (pseudo)random values, an attacker that reads the channel obtained no useful information. Our protocol transmits the shares of the actual vote through the channel, dictating to keep the post-voting channel completely hidden from the attacker, otherwise privacy is lost with the attacker obtaining the vote similar to the voter.

If we need to eliminate the assumption of a completely secure real-time channel we need to sent over the post-channel security codes, which are generated for each voter specifically during a setup phase run by trusted servers, and distribute them before the elections using a secure pre-channel. In order to achieve privacy the attacker should not be able to tamper with the channel in any of the following ways:

- An attacker should not be able to write in the pre-channel or re-arrange its contents.
- An attacker should not be able to write in the post-channel.

- An attacker may read the contents of either the the pre-channel or the post-channel but not both of them.

Requesting that the attacker cannot alter the contents of the channels is a necessary requirement, since otherwise he could substitute the correct values by values of his own choice causing possible fake vote submission or reception of wrong security codes and skepticism about the system's credibility. Furthermore we require that the attacker cannot read both channels since by obtaining all the secrets of the voter he violates privacy. The usual implementation of the channels as paper mail that includes a voting card and SMS with the security codes (or their shares) satisfies our security requirements.

The security codes are necessary in order to assure the voters that their votes reached the voting servers while preventing the post-channel from leaking information regarding the vote to an attacker that reads its contents. We propose a simple approach for creating the security codes with an adaptation to the original protocol. We create the security codes through an one time pad scheme of the actual votes, setting the code for voter V and candidate x to be $x + b_V \bmod m$, with b_V being a secret "padding" value, selected for each voter and possessed by the voting servers in a secret-shared form. The new protocol satisfies our security model with a third property regarding the channel's security, namely post-channel secrecy stating that an observer of the post-channel can not extract any information about the submitted vote from the security codes. Specifically we need that $\forall\ c \in Code, \forall\ x \in Cand\ Pr[c = code(x)] = c_0$ where $c_0 \geq 0$ is a constant. .

4.2.2 First code verification protocol

We proceed by describing our first code verification adaptation. During code generation and set-up phase, for each voter V we pick a set of k values $\{b_{1V}, b_{2V}, \ldots, b_{kV}\} \leftarrow Z_m^k$, compute $b_V = \sum_{i=1}^{k} b_{iV} \bmod m$ and set the code for candidate x to be $code_V[x] = x + b_V \bmod m$. Then we send the corresponding pairs $\langle x, code_V[x] \rangle$ to the voter V, through the secure pre-channel, and the values $\langle V, b_{iV} \rangle$ to the voting server A_i. During election phase a voter casts a vote for x through her personal computer, which functions identically to the original protocol. The same holds for the voting servers that verify the additive relations and the range proof, but upon success, instead of the share x_i, each server A_i forwards the value $c_i = x_i + b_{iV} \bmod m$ which is sent through the secure post-channel. Finally the voter adds the received code shares

$\sum_{i=1}^{k} c_i = \sum_{i=1}^{k} x_i + \sum_{i=1}^{k} b_{iV} = x + b_V \bmod m$ and verifies it matches the security code $code_V[x]$.

To avoid storing the padding values b_{iV} for each voter in the voting servers, we generate them using a suitable pseudo-random function family $\{\mathcal{F}\} : \{0,1\}^n \times \{0,1\}^n \rightarrow Z_m$, where n bits suffice for representing all voters. For each server A_i a secret key $k_{A_i} \in \{0,1\}^n$ for a function $f \in \mathcal{F}$ is chosen and values b_{iV} are set to be $f_{k_{A_i}}(V)$. Thus upon receiving a vote from voter V the server A_i generates the correct code share computing $a_i = x_i + f_{k_{A_i}}(V) \bmod m$.

4.2.2.1 Security guarantees

In this approach, if the attacker reads the post-channel he can reconstruct only the security code from which he cannot extract the vote. Both our encoding domain and code domain are Z_m since the codes correspond to a fixed cyclic swift of positions. The scheme leaks no information about the vote to an attacker of the post-channel, since any code could correspond to different candidates for different padding values. It provides the security guarantees of the original approach plus the channel privacy requirement. In this setting re-voting is not allowed since an attacker of the post-channel learns the relative difference of the submitted votes, which leaks the voting pattern and critical information about the vote if the elections use a subset $Z_N \subseteq Z_m$ and not the whole domain.

4.2.3 Second code verification protocol

For important elections coercion is considered a main threat which should be dealt with re-voting (or other applicable approaches), while for low-coercion elections re-voting is not a necessary feature of voting protocols, yet it remains desirable. In order to overcome the drawback of the previous scheme that reveals information about the votes in re-voting, a natural approach would be to restrict the candidate encodings and padding values to Z_N, transforming the protocol to work under modulo N operations, for an arbitrary number of candidates, instead of modulo m for a fixed power of 10. Under this transformation for any difference value δ of two submitted votes there are always N possible pairs of candidates that differ by δ, so that the attacker cannot eliminate any options about the vote. However, we have chosen m to be a power of 10 to allow easy operations on the voter's part, who we assume incapable of executing modular additions in general. Hence we can employ a setting similar to existing code verification protocols [4]

[1] where a single entity forwards the code to the voter, who just compares it with her code sheet.

For this purpose we introduce a new server that is responsible to compute the code from the code shares and send it to the voter. Unlike previous protocols, where the messenger server contributed to the code reconstruction by applying secret values, our new entity will be an untrusted server that posses no secrets. The only function of this server, which we call the calculator sever, is to perform modular addition of the shares $c_i = x + b_{Vi} \bmod N$ he receives from each voter, obtaining $code = \sum_{i=1}^{k} c_i \bmod N$. Finally he sends the code to the voter through the post-channel. The protocol is depicted in figure 4.5.

4.2.3.1 Security guarantees and overhead

Since our first code verification attempt satisfies privacy guarantees against an attacker of the post-channel the same holds for the calculator who sees exactly the same information when each voter submits exactly one vote. In case of revoting both the calculator and an observer of the post-channel learn the relative difference between the submitted ballots, yet for any candidate there is another candidate that yields the fixed difference. Hence we obtain a secure code verification protocol that achieves all necessary security requirements and supports re-voting, inheriting the security properties of our code verification protocol and security against the post-channel. The protocol introduces an corruption scenario, present in all previous protocols [4] [1]: the coalition of a malicious PC and a malicious calculator can submit undetectably forged ballots. In this scenario the malicious PC, after the voter submits her ballot, re-submits a forged ballot. The malicious calculator server drops the second security code that corresponds to the fake ballot and forwards only the first code he has recorded. We note that this attack is present in all code verification protocols where a single entity is responsible for sending the security codes. In our protocol if the malicious PC wants to submit a fake ballot then it must compromise either the calculator server and apply the previous attack or one voting server who sends a modified code share, like in the attack in our vote verification protocol.

Regarding the protocol's complexity, as trade-off for the user's convenience and re-voting option, we add more cryptographic operations to achieve privacy, since each voting server should encrypt his code share c_i and send it to the calculator, who decrypts and adds k values. Hence we have an additional overhead of $3k$ exponentiations, $2k$ for each server and k for the calcula-

tor, corresponding to k encryptions and decryptions.

4.3 Extension to visual vote verification

In this section we introduce the visual vote verification protocol, that provides visual receipts whose overlaying reveals the submitted vote. Based in our vote verification protocol we proposed, we derive a visual verification scheme for small scale elections. The protocol uses two voting server and requires the existence of the untappable one-way channel to forward the visual shares to the voter.

Our approach uses visual receipts in a new way, going further that concealing black and white pixels through seemingly random patterns. We use black and white shares of an image, that is split between two online voting servers, who forward them to the voter for verification. We introduce a visual encoding of the votes so that their shares leak no useful information on their own, while their combination reveals the vote. The only action needed by the voter is to overlay visually or mentally the provided images and deduce the number of white cells, which correspond to the submitted vote. Hence we achieve a user-friendly and easy way to verify votes in remote voting.

4.3.1 Previous work

Our motivation is derived from Shamir and Naor's visual 2-out-of-2 secret sharing protocol [27] that uses subdivisions of pixels containing 2 black and two white cells to share a visual secret, i.e. a black or white pixel (see figure 4.7). In this setting a pixel is black if it contains at least 4 black cells and white is it contains at most 2 black cells. Complementary shapes yield fully black pixels while identical shapes while half-black or white pixels according to our threshold.

The idea of providing visual receipts in electronic voting is not new, as it was first described by Chaum [28], who gave a scheme for providing visual receipts in kiosk-voting. Chaum exploits the ideas of Naor and Shamir's visual secret sharing to conceal the written form of the submitted candidate in a black and white image. The scheme applies only in supervised voting as it requires the use of a special receipt printer that can print an image in two complementary transparent

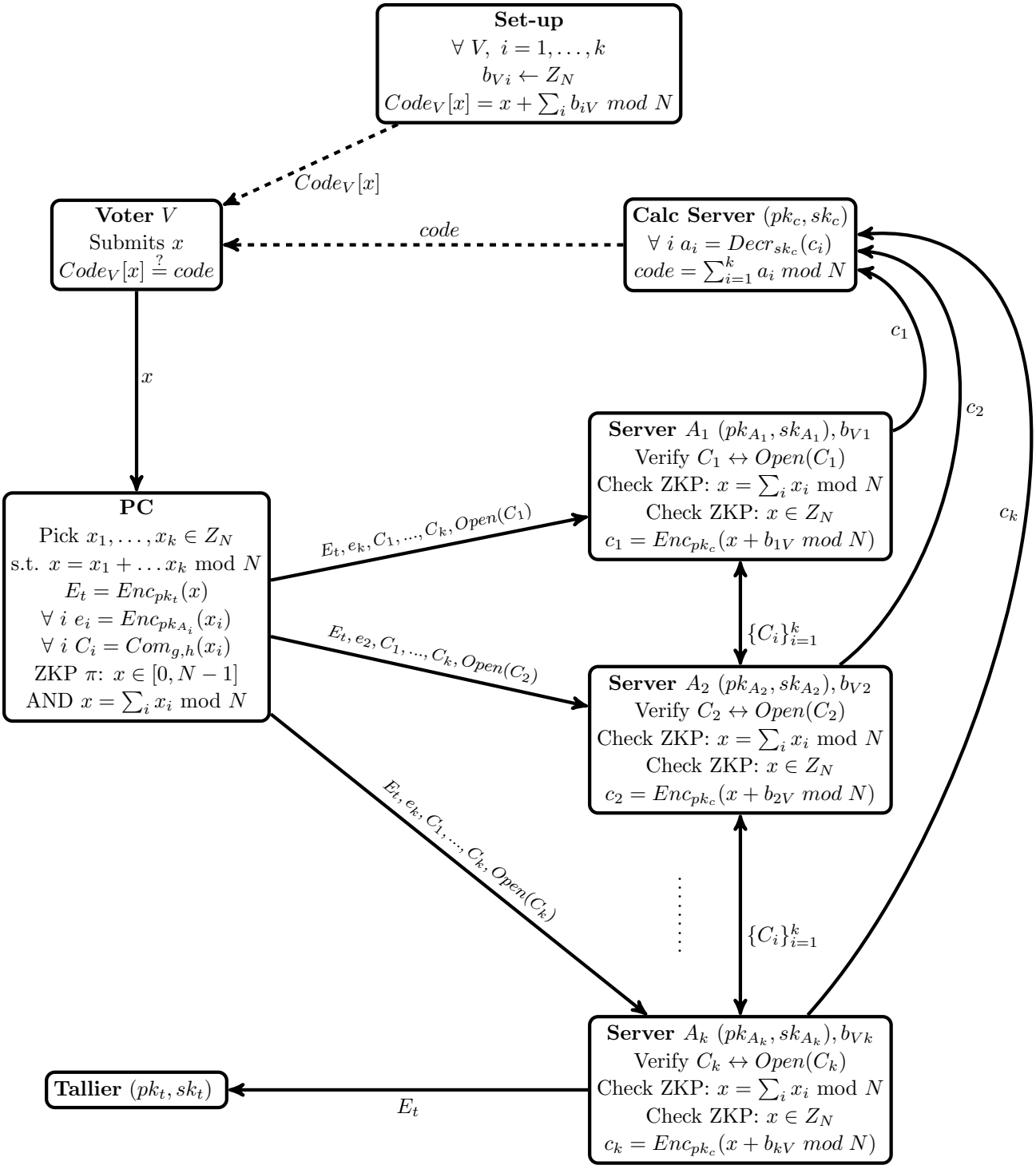

Figure 4.5: Overview of the code verification protocol

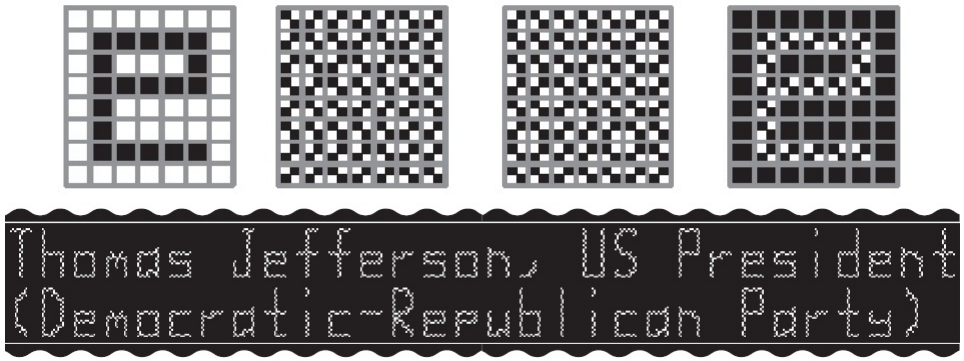

Figure 4.6: Chaum's visual receipts

sheets. The overlaying of both sheets reveals a receipt of the vote and the hiding property of the 2-out-of-2 secret sharing scheme used ensures that each share leaks no information about the submitted vote. An example of printing the visual receipts is given in figure 4.3.1 (source [28]).

After submitting a vote the and the receipt is printed, the user decides which sheet she will keep as a receipt of her vote. The fact that printing takes place before the final choice of the sheet to be kept ensures that the voting machine will be honest and cannot change the contents of the ballot without risk to be detected with 1/2 probability by the user. After making her selection the user verifies the vote by overlaying the printed images and the selected sheet is also posted publicly on the ballot box. Later the voter can verify that her ballot was cast correctly by accessing the ballot box and comparing her receipt sheet with the one corresponding on the posted receipt under the same ballot identification number. The other receipt sheet is destroyed by the officials supervising the elections to achieve receipt-freeness.

Vote submission and consequently counting is different from the schemes we have discussed so far. The ballot box stores a copy of the receipt sheet chosen by the voter which is used from the voting system to reconstruct the initial vote as human readable plaintext image. The encryption used corresponds to an one-time-pad scheme with the chosen receipt being the vote to be encrypted and the complementary sheet being the secret key. Tallying takes place by using specialized software that identifies the final images and counts the ballots. A tabulation process transforms the posted receipts into the original images to be tallied. This phase takes place after the end of the elections and is performed by a set of tallier servers that mix the votes to

ensure anonymity while changing the vote coding to gradually reveal the vote through a number of steps. The posted receipt is passed through all participating servers and is transformed in each step: the input image is combined with a secret image possessed by each server and for each white pixel of the aligned image a pixel symbol is printed in a new sheet, while for each black pixel a different pixel is printed. The new image is given as input to the next server and the process continues to the final step that reveals the vote.

4.3.2 The visual vote encoding

We begin our discussing by defining the appropriate vote encoding the facilitates visual vote verification. Our construction is based on the fact that each candidate is represented by a number of white cells within the receipt image, independently of their order, allowing a candidate to have multiple visual representations. It follows that the number of candidates affects the number of cells required to encode our options without revealing information about the vote.

Our construction represents visually a candidate as an image with black and white cells that has a 1-1 correspondence to a bit-string with "1" denoting a black cell and "0" a white cell. The visual representation of vote is split in two images whose overlaying yields the initial vote. When overlaying the visual shares a white cell is created when two white cells are aligned, while the alignment of a black cell with any other cell yields a black cell. We model visual overlaying as the bit-wise OR operation on the bit-strigs that correspond to the visual shares. We map each candidate x into a set of visual descriptors $\{D_x\}_{x \in Cand} \subset \{0,1\}^\lambda$, for an appropriate choice of λ bits, with every string in $\{D_x\}$ having exactly x zeros. This encoding imposes an exponential increase in the domain requiring $\lambda = min\{4, 2^{\lceil log_2(N-1)+1 \rceil}\}$ visual bits for N candidates, normally represented by $[log_2(N-1)] + 1$ bits. We will see that this increase is necessary to guarantee voter privacy against the voting servers.

Our building block for the visual vote representation is the 4-cell image of Naor and Shamir's 2-out-of-2 secret sharing, through images with exactly two black and to white cells. As 4 cells suffice for encoding 3 options, with zero, one or two white cells respectively, for a candidate $x \in \{0,1,2\}$ we have some randomly selected valid visual shares $v_1, v_2 \in \{0,1\}^4$ with equal number of "1" and "0" such that the value $v = v_1 \vee v_2$ has x zeros, where $\vee$ denotes the bit-wise OR operator and $v \in D_x$ is the visual representation of candidate x. We extent the visual encoding

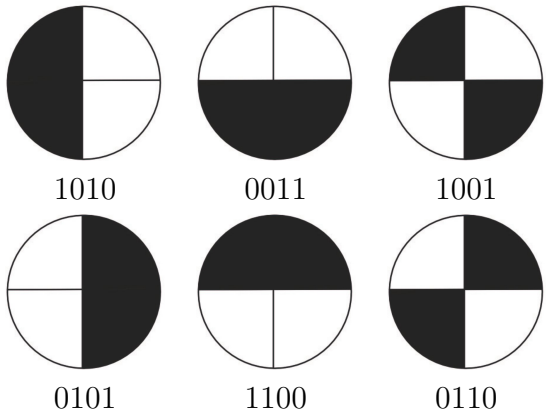

Figure 4.7: The visual building block

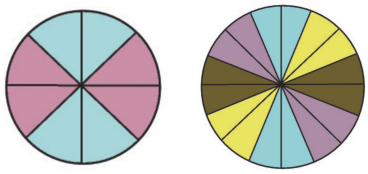

Figure 4.8: 8-bit and 16-bit subdivisions of the building block

to an arbitrary number of N candidates by sub-dividing each cell in our building block. We repeating this procedure until we have $\lambda = (2^{\lceil log_2(N-1)+1 \rceil})$ cells, that are sufficient to encode N candidates without revealing information. We observe that each distinct old pair of opposite angles yields a new building block, which we require to have equal number of black and white cells (see figure 4.8).

We wish to design a scheme based on Naor and Shamir's 2-out-of-2 simple secret sharing scheme [27] for an arbitrary number of candidates. Thus we double the number of cells by sub-dividing each cell of our building block as many times as we need in order to have sufficient number of cells (say λ cells) so that $\lambda/2 + 1 \geq N$, where $\lambda = 4\mu$. By this construction we allow half cells to be black and half cells to be white, generalizing the property of the 4-cell image. Each sub-division yields new quadruples of cells that are treated for simplicity as a new building block. We chose to consider distinct pairs of opposite angles as our 4-cell building block, which is captured in the properties of the valid visual representations. As the number of white cells in an image denotes the candidate number that was submitted, λ cells suffice for representing $\lambda/2 + 1$ candidates, implying the relation $\lambda = 2^{\lceil log_2(N-1)+ \rceil}$.

4.3.3 The ideal security model and our guarantees

Let us define the properties of a scheme, which we refer as k Visual Sharing of Shape Descriptors (VSSD). In VSSD we wish to represents the elements of a group through splittable visual shares that can be reconstructed visually to yield the representation of the original input.

- Let $(\Lambda, \vee)$ be a commutative semigroup. Let $\{D_x\}_{x \in Cand}$ be the set of visual descriptors and Λ the set of valid visual shares. Let $P : Cand \rightarrow 2^{\Lambda^k}$ be the encoding function, mapping candidates to all possible splitting in visual shares. Then a VSSD scheme must fullfil the following properties.
- Solvability: $\forall x \in Cand\ \forall (v_1, \ldots, v_k) \in P(x)$ it must hold that $\vee_i v_i \in D_x$.
- t-Resilience: $\forall W \in (\Lambda \cup \{*\})^k$ (having t fixed values and $k - t$ " $*$ " unknown values) it must hold that $\forall x \in Cand\ Prob_{A \leftarrow P(x)}[W \sqsubseteq A] = c$, where $c \geq 0$ is a constant and $W \sqsubseteq A$ represents that W is part of visual representation A .

Our security guarantees. For our protocol we will need to relax the above security property of t-resilience, to achieve a user-friendly vote verification at the expense of a weaker model of privacy. Our scheme will focus on the 2-VSSD problem, working with 2 voting servers. In the setting our privacy guarantee states that given a visual share W of a vote x the share W may be part of a representation of any vote $x' \in Cand$ and we extract no useful information than those the were known before we saw share x.

4.3.4 Our visual sharing shape descriptor construction

In our construction we will split a visual representation into two shares. Below we describe the properties of our 2-VSSD construction.

Construction of valid visual shares. Let x be a candidate and $\{v_1, v_2\} \in P_\lambda(x)$, $v_1, v_2 \in \Lambda_\lambda$, a set of its corresponding λ-bit visual shares. Then the following must hold:

- $v_i \in \{0,1\}^\lambda$, for $i = 1, 2$.

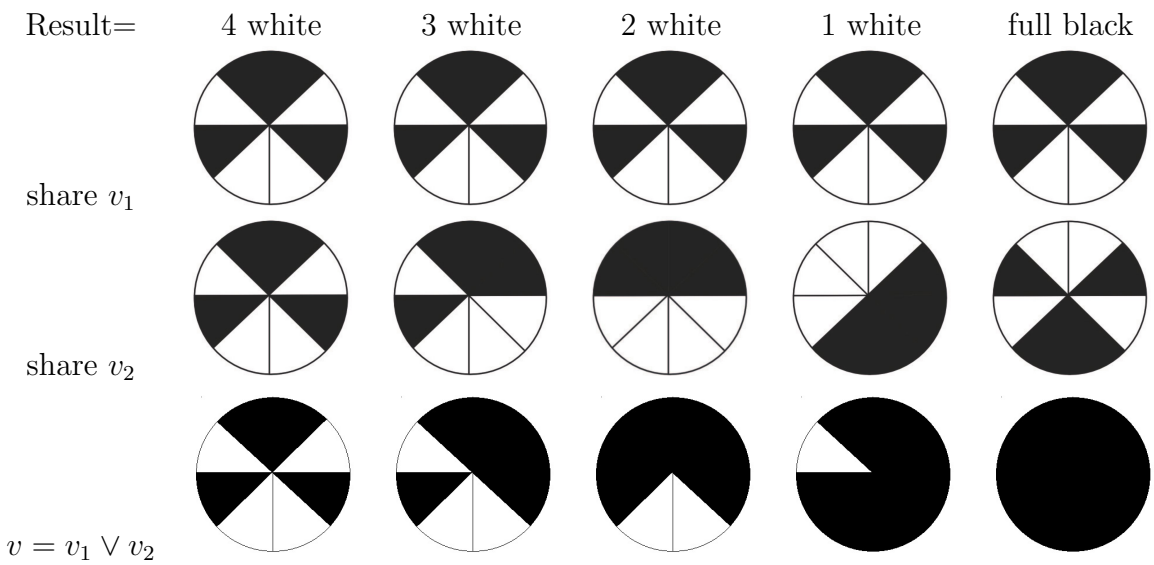

Figure 4.9: Example of encoding of 5 candidates using the same fixed share v_1.

- Let $v_i = (v_{i0}v_{i1}v_{i2}\ldots v_{i\frac{\lambda}{2}}v_{i\frac{\lambda}{2}+1}\ldots v_{i,\lambda-1})$, for $i = 1, 2$. Then each pair of bits $(v_{ij}, v_{i,j+1})$ for $j = 0, 2, \ldots, \frac{\lambda}{2} - 2$ shares with the pair $(v_{i,j+\frac{\lambda}{2}}, v_{i,j+\frac{\lambda}{2}+1})$ exactly two "0" and two "1".

Construction of valid visual encodings. Let x be a candidate and D_x^λ the set of all valid visual λ-bit representations for this candidate. Then for all $v \in D_x^\lambda$ it must hold that:

- $v \in \{0, 1\}^\lambda$.
- Let $v = (v_0v_1v_2\ldots v_{\frac{\lambda}{2}}v_{\frac{\lambda}{2}+1}\ldots v_{\lambda-1})$. Then each pair of bits (v_j, v_{j+1}) for $j = 0, 2, \ldots, \frac{\lambda}{2} - 2$ shares with the pair $(v_{j+\frac{\lambda}{2}}, v_{j+\frac{\lambda}{2}+1})$ at most two "0".
- The number of "0" in v equals x.

Figure 4.10 depicts the valid shares for an 8-bit constructions Λ_8 and figure 4.9 provides an example of encodings all possible 5 candidates with elements of Λ_8.

4.3.5 Vote submission and verification

Having described the properties of our visual encoding we are ready to design the visual vote verification protocol. Our protocol works over a finite cyclic group $G \subset Z_p$ of prime order q, with large primes $q|p-1$ and $\{0, 1\}^\lambda \subset Z_q$, such that we can use additive ElGamal and commit

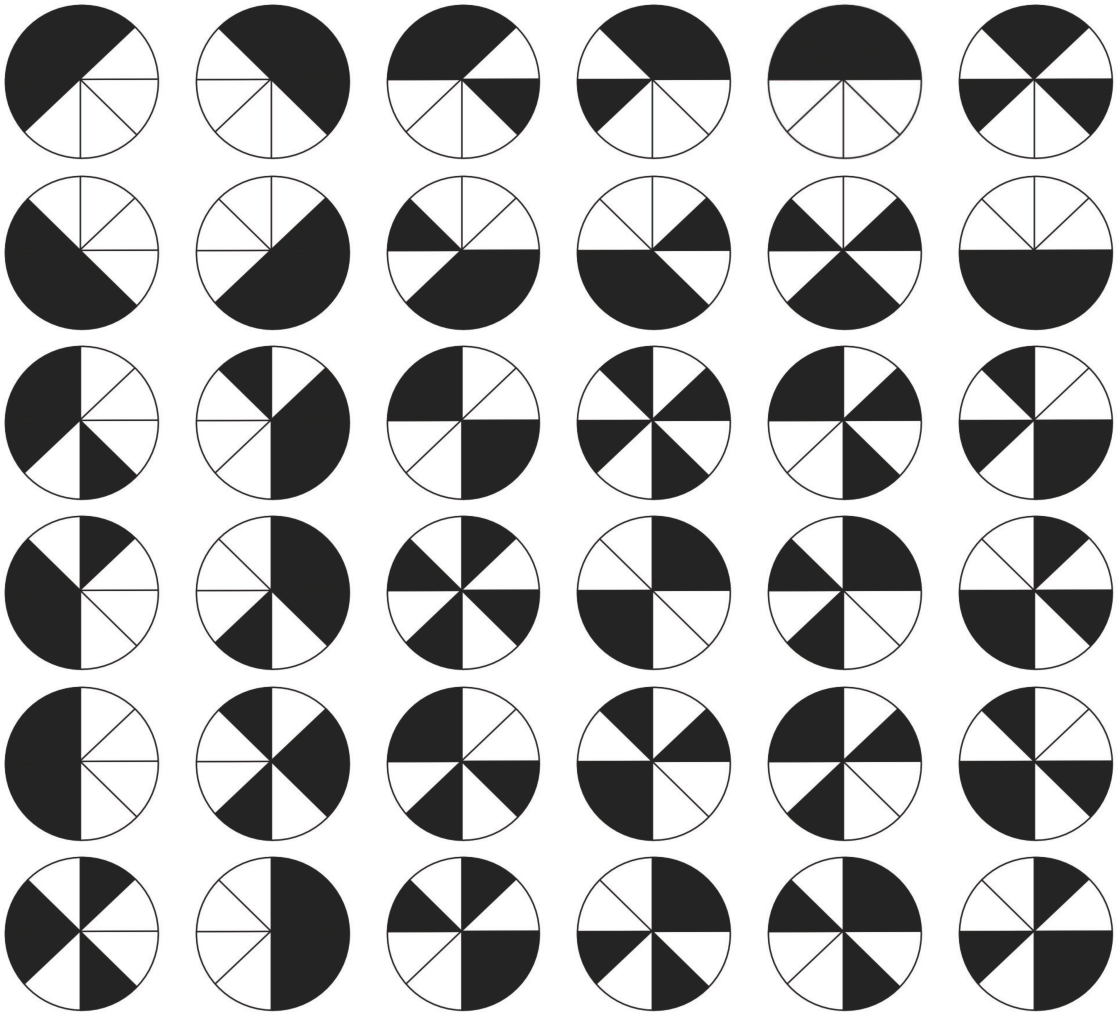

Figure 4.10: All valid shares of Λ_8 that are sufficient for encoding up to 5 candidates.

to visual representations by Pedersen commitments.

To cast a vote for candidate x a voter inputs the globally known value x to her computer. The PC encrypts and submits the vote and picks set of valid visual representation shares $\{v_1, v_2\} \in P(x)$ for the vote. The PC prepares a proof of compatibility of the visual representation $v_x = v_1 \vee v_2$, the shares v_1, v_2 and the vote x. It commits to each bit of v_x, v_1, v_2 as C_{xj}, C_{1j}, C_{2j} for all $j = 0, \ldots, \lambda - 1$ and proofs validity of bit commitments and the bit-wise OR relation using a Schnorr OR proof (figure 4.11). The latter is proved based on the fact that for all bits $j = 0, \ldots, \lambda - 1$ if $v_{xj} = v_{1j} + v_{2j}$, if the bits correspond to correct visual representations of the vote then the commitment $C_{OR_j} = (C_{1j}C_{2j})/C_{xj}$ hides a value in $\{0, 1\}$. The table in figure 4.12 shows the possible bit combinations.

In addition the PC proves that $v_x = v_1 \vee v_2$ is a valid visual representation of x by proving that the corresponding bit commitments hide exactly x zeros, showing that the commitments $C_x = g^x pk_t^r$ and $g^N / \prod_{j=0}^{\lambda-1} C_{xj}$ hide the same value. This done by using the zero knowledge proof that two commitments open in the same value of figure 4.1. The full proof $\pi' = PK(x, r, \{(v_{ij}, r_{ij}\}_{j=0}^{\lambda-1})_{i=1,2,x} \mid (\{C_{ij} = g^{v_{ij}} h^{r_{ij}} \wedge v_{ij} \in \{0,1\}\}_{j=0}^{\lambda-1})_{i=1,2,x} \wedge \{(C_{1j}C_{2j}/C_{xj}) = g^{b_j} h^{(r_{1j}+r_2-r_{xj})} \wedge b_j \in \{0,1\}\}_{j=0}^{\lambda-1} \wedge E_x = Enc_{pk_t}^{r \in Z_q}(g^x) \wedge x = N - \sum_{j=0}^{\lambda-1} v_{xj})$ is given in protocol 19, adapting the techniques of [17].

Each server receives a share and verifies its compatibility with the encrypted vote, checking that it is of the appropriate form. Upon successful verification the server forwards the corresponding image to the voter through the untappable channel. Then the voter can verify visually the result by overlaying the images and concluding about the number of white cells. When elections are over, one of the online servers forwards to the tallier the encrypted vote.

4.3.6 Security and complexity properties

The visual verification protocol we propose is an extension of the vote verification protocol in the restricted case for two voting servers. Hence it guarantees that a submitted vote remains unchanged or the voter is notified about a forgery in the presence of a malicious PC, as she reconstructs a wrong visual receipt.

	Public Input: g, h $C = g^v h^r$	
Prover Private input: $v \in \{0,1\}, r$ Case $v = 0$: $w, c_2, \rho_2 \xleftarrow{\mathtt{r}} Z_q$ $y_1 = h^w,\ y_2 = h^{\rho_2}(C/g)^{-c_2}$ Case $v = 1$: $w, c_1, \rho_1 \xleftarrow{\mathtt{r}} Z_q$ $y_1 = h^{\rho_1}(C)^{-c_1},\ y_2 = h^w$	$\xrightarrow{y_1,\ y_2}$	Verifier
		$c \xleftarrow{\mathtt{r}} Z_q$
	$\xleftarrow{c}$	
Case $v = 0$: $c_1 = c - c_2, \rho_1 = w + c_1 r$ Case $v = 1$: $c_2 = c - c_1, \rho_2 = w + c_2 r$	$\xrightarrow{c_1,\ c_2,\ \rho_1,\ \rho_2}$	$c \stackrel{?}{=} c_1 + c_2$ $h^{\rho_1} \stackrel{?}{=} y_1(C)^{c_1}$ $h^{\rho_2} \stackrel{?}{=} y_2(C/g)^{c_2}$

Figure 4.11: Schnorr OR ZK Proof

C_1	C_2	C_x	C_1C_2/C_x	OR	Valid
$g^0h^{r_1}$	$g^0h^{r_2}$	$g^0h^{r_x}$	$g^0h^{r_1+r_2-r_c}$	0+0=0	yes
$g^0h^{r_1}$	$g^1h^{r_2}$	$g^1h^{r_x}$	$g^0h^{r_1+r_2-r_c}$	0+1=1	yes
$g^1h^{r_1}$	$g^0h^{r_2}$	$g^1h^{r_x}$	$g^0h^{r_1+r_2-r_c}$	1+0=1	yes
$g^1h^{r_1}$	$g^1h^{r_2}$	$g^1h^{r_x}$	$g^1h^{r_1+r_2-r_c}$	1+1=1	yes
$g^0h^{r_1}$	$g^0h^{r_2}$	$g^1h^{r_x}$	$g^{-1}h^{r_1+r_2-r_c}$	0+0=1	no
$g^1h^{r_1}$	$g^1h^{r_2}$	$g^0h^{r_x}$	$g^2h^{r_1+r_2-r_c}$	1+1=0	no

Figure 4.12: Bitwise OR combinations

Protocol 19 Proof of valid visual votes

Public Input: $\langle p, q, g\rangle$, h, pk_t, N, $\lambda = (2^{\lceil log_2(N-1)+1\rceil})$, $E_t = (E_x, E_r) = (g^x pk_t^r, g^r)$, $(\{C_{ij} = g^{v_{ij}} h^{r_{ij}}\}_{j=0}^{\lambda-1})_{i=1,2,x}$. **Prover's Input:** $x, r, (\{v_{ij}, r_{ij}\}_{j=0}^{\lambda-1})_{i=1,2,x}$.

1. The Prover:
   - For $j = 0, \ldots, \lambda - 1$:
     - For $i = 1, 2, x$:
       - If $v_{ij} = 0$ he picks $w_{ij}, c_{2ij}, \rho_{2ij} \leftarrow Z_q$ and sets $y_{1ij} = h^{w_{ij}}$, $y_{2ij} = h^{\rho_{2ij}}(C_{ij}/g)^{-c_{2ij}}$.
       - Else if $v_{ij} = 1$ he picks $w_{ij}, c_{1ij}, \rho_{1ij} \leftarrow Z_q$ and sets $y_{1ij} = h^{\rho_{2ij}}(C_{ij})^{-c_{1ij}}$, $y_{2ij} = h^{w_{ij}}$.
     - If $v_{1j} + v_{2j} - v_{xi} = 0$ he picks $w_j, c_{2j}, \rho_{2j} \leftarrow Z_q$ and sets $y_{1j} = h^{w_j}$, $y_{2j} = h^{\rho_{2j}}(\frac{C_{1j}C_{2j}}{g \cdot C_{xj}})^{-c_{2j}}$.
     - Else if $v_{1j} + v_{2j} - v_{xi} = 1$ he picks $w_j, c_{1j}, \rho_{1j} \leftarrow Z_q$ and sets $y_{1j} = h^{\rho_{1j}}(\frac{C_{1j}C_{2j}}{C_{xj}})^{-c_{1j}}$, $y_{2j} = h^{w_j}$.
   - He picks $w, \rho_1, \rho_2 \leftarrow Z_q$ and sets $y_1 = g^w h^{\rho_1}$, $y_2 = g^w pk_t^{\rho_2}$
   - He sends $(y_1, y_2, \{y_{1j}, y_{2j}\}_{j=0}^{\lambda-1}, (\{y_{1ij}, y_{2ij}\}_{j=0}^{\lambda-1})_{i=1,2,x})$ to the Verifier.
2. The Verifier picks $\mathbf{c} \leftarrow Z_q$ and sends it to the Prover.
3. The Prover:
   - For $j = 0, \ldots, \lambda - 1$:
     - For $i = 1, 2, x$:
       - If $v_{ij} = 0$ he sets $c_{1ij} = \mathbf{c} - c_{2ij}$, $\rho_{1ij} = w_{ij} + c_{1ij} r_{ij}$.
       - Else if $v_{ij} = 1$ he sets $c_{2ij} = \mathbf{c} - c_{1ij}$, $\rho_{2ij} = w_{ij} + c_{2ij} r_{ij}$.
     - If $v_{1j} + v_{2j} - v_{xi} = 0$ he sets $c_{1j} = \mathbf{c} - c_{2j}$, $\rho_{1j} = w_{1j} + c_{1j}(r_{1j} + r_{2j} - r_{xj})$.
     - Else if $v_{1j} + v_{2j} - v_{xi} = 1$ he sets $c_{2j} = \mathbf{c} - c_{1j}$, $\rho_{2j} = w_{2j} + c_{2j}(r_{1j} + r_{2j} - r_{xj})$.
   - He sets $s = w + \mathbf{c}x$, $\rho_1' = \rho_1 + \mathbf{c}r$, $\rho_2' = \rho_2 + \mathbf{c}(-\sum_{j=0}^{\lambda-1} r_{xj})$
   - He sends $(s, \rho_1', \rho_2', \{c_{1j}, c_{2j}, \rho_{1j}, \rho_{2j}\}_{j=0}^{\lambda-1}, (\{c_{1ij}, c_{2ij}, \rho_{1ij}, \rho_{2ij}\}_{j=0}^{\lambda-1})_{i=1,2,x})$ to the Verifier.
4. The Verifier accepts if all the following tests succeed, otherwise he rejects:
   - For $j = 0, \ldots, \lambda - 1$:
     - For $i = 1, 2, x$: $\mathbf{c} = c_{1ij} + c_{2ij}$ and $h^{\rho_{1ij}} = y_{1ij}(C_{ij})^{c_{1ij}}$ and $h^{\rho_{2ij}} = y_{2ij}(C_{ij}/g)^{c_{2ij}}$.
     - $\mathbf{c} = c_{1j} + c_{2j}$ and $h^{\rho_{1j}} = y_{1j}(\frac{C_{1j}C_{2j}}{C_{xj}})^{c_{1j}}$ and $h^{\rho_{2j}} = y_{2j}(\frac{C_{1j}C_{2j}}{g \cdot C_{xj}})^{c_{2j}}$.
   - $g^s h^{\rho_1'} = y_1(E_x)^{\mathbf{c}}$ and $g^s pk_t^{\rho_2'} = y_2(\frac{g^N}{\prod_{j=0}^{\lambda-1} C_{xj}})^{\mathbf{c}}$.

Protocol 20 Visual Vote Verification

Let N be the number of candidates, λ be the bit-sting length for the encoding, h a randomly chosen public key in G such that no one knows its discrete logarithm and $(pk_{A_i}, sk_{A_i}), (pk_t, sk_t)$ the public/secret key pairs of the online severs A_i $(i = 1, 2)$ and the tallier respectively.

- The voter V casts a ballot for candidate x to her PC and waits for the images from the voting servers.
- The PC:
  1. Picks a valid visual representation $v_x \in S_{x,\lambda}$ for the candidate and shares $v_1, v_2 \in \mathcal{V}_\lambda$ s.t. $v_x = v_1 + v_2$.
  2. Encrypts the vote as $E_t = Enc_{pk_t}^{\rho \leftarrow Z_q}(g^x)$.
  3. For each $j = 0, \ldots, \lambda - 1$ it computes the Pedersen commitment to the j-th bit of v_x, v_1 and v_2 as $C_{xj} = g^{v_{xj}} h^{r_{xj}}, C_{1j} = g^{v_{1j}} h^{r_{1j}}, C_{2j} = g^{v_{2j}} h^{r_{2j}}$.
  4. Commits to shares v_i, $i = 1, 2$, by the Pedersen commitment $\mathcal{E}_i = g^{v_i} h^{\rho_i}$ where $\rho_i = \sum_{j=0}^{\lambda-1} r_{ij} 2^j$ and encrypts the randomness $Open_i = Enc_{pk_{A_i}}^{r_i' \leftarrow Z_q}(\rho_i)$.
  5. Encrypts each share v_i, $i = 1, 2$, as $e_i = Enc_{pk_{A_i}}^{r_i \leftarrow Z_q}(v_i)$.
  6. Prepares a zero knowledge proof $\pi' = PK\{\mu, \rho, \{(\mu_{ij}, \rho_{ij})_{j=0}^{\lambda-1}\}_{i=1,2,c} \mid E_t = Enc_{pk_t}^{\rho}(g^\mu) \wedge \{(C_{ij} = g^{\mu_{ij}} h^{\rho_{ij}} \wedge \mu_{ij} \in \{0, 1\})_{j=0}^{\lambda-1}\}_{i=1,2,x} \wedge \{\mu_{1j} + \mu_{2j} = \mu_{cj}\}_{j=0}^{\lambda-1} \wedge (N - \sum_{j=0}^{\lambda-1} \mu_{cj} = \mu)\}$
  7. Signs the encrypted message and the proof.
  8. Sends $(V, E_t, sing_V(E_t, \pi'), e_i, \mathcal{E}_i, Open_i, (C_0, \ldots, C_{\lambda-1}), (C_{1,0}, \ldots, C_{1,\lambda-1}), (C_{2,0}, \ldots, C_{2,\lambda-1}), \pi')$ to the server A_i, for $i = 1, 2$.
- Server A_i :
  1. Sends to server A_j $(j \neq i)$ the commitments $\{(C_{ik})_{k=0}^{\lambda-1}\}_{i=1,2,x}$ and checks that the received values from A_j match his values.
  2. Verifies the voter's signature.
  3. Decrypts $e_i, Open_i$ to obtain v_i, ρ_i and verifies that $\mathcal{E}_i = g^{v_i} h^{\rho_i}$ and checks that $v_i \in \mathcal{V}_\lambda$.
  4. Checks that $\mathcal{E}_i = \prod_{j=0}^{\lambda-1} (C_{ij})^{2^j}$ and verifies π'.
  5. If all checks are valid he sends the corresponding image to the voter through the post-channel, stores the vote E_t and signs it. Otherwise he complains about a forgery.
- The Tallier: Obtains the votes E_t from a voting sever and runs an appropriate protocol for decrypting and tallying the ballots.

The visual verification protocol we propose meets the relaxed criteria of 2-secure visual sharing of shape descriptors we have defined. Clearly the protocol meets the solvability property, which expresses the correctness of the scheme. Given to visual shares v_1, v_2 that are generated through $P(x)$ their visual overlying $v_i \vee v_2 = v_x$ is a valid visual representation candidate x.

In terms of privacy, we obtain a protocol that ensures that given a single valid visual share $v_1 \in \Lambda_\lambda$, the share v_1 may be part of a representation of any candidate $x \in Cand$, depending on the other, unknown, visual share. Thus without having their relative position the shares on their own carry no new information about the vote.

Let us clarify the notion of "new information" obtained from a share about a vote. Clearly in our scheme there are votes $x \in Cand$ that have bigger number of possible splittings $|P(x)|$ than other votes $x' \in Cand$ having $|P(x')|$ splittings, preventing us from achieving the t-resilient property. We note that the voting server before seeing any share at all, knows that the probability that a share belongs in $P(x)$ is higher than the probability that the same share belongs in $P(x')$, having that $\forall x \in Cand$, $\forall v \in \Lambda$ $Pr_{A \leftarrow P(x)}[(v, *) \in A] = p_x$ with $p_x > p_{x'}$ for any such pair (x, x') of candidates. Thus given a specific share $v_1 \in \Lambda_\lambda$, the voting server obtains no more information than those it already new about the possible encoding the share may belong. Thus the knowledge of the actual share does not help it to identify the vote.

We now calculate the complexity of the visual verification protocol in terms of the value $\lambda = 2^{\lfloor log_2(N-1)+1 \rfloor}$ which is linearly dependent in the number of candidates.

- The PC: The PC encrypts the vote and the visual two shares and the randomness used for committing to them with 10 exponentiations. Moreover it computes the bit commitments of the shares and the visual vote representation with 6λ exponentiations as well as the commitments to the shares with 4 more exponentiations giving a total of $6\lambda+14$ exponentiations. Running the provers part of the zero knowledge proofs requires 9λ exponentiations for the bit commitments and 3λ exponentiations for the bit-wise OR Schnorr proofs. The prover's part for the valid visual representation proof requires 3 exponentiations giving a total of $18\lambda + 17$ exponentiations. In addition he performs one signing.
- The servers: Each server A_i decrypts 2 values and checks one commitment with 4 exponentiations. The verification part of the zero knowledge proofs requires 12λ exponentiations for the single bit proofs and 4λ exponentiations for the OR proof. The visual representa-

tion proof requires 5 exponentiations and the compatibility check of the bit commitments and the commitment to the share requires λ exponentiations, giving a total of $17\lambda + 9$ exponentiations.

Conclusion

In this thesis we have tackled the problem of untrusted voting clients in remote Internet voting. We focused on the code verification protocols and reviewed the related work in the area. Our contribution is summarized in designing a new vote verification protocol with enhanced vote secrecy properties and no need for set-up phase for verification purposes, compared to the previous approaches. We used a simple secret-sharing scheme to split a vote in several shares given to different voting servers who perform validity tests and assure the voter about the correct submission of the vote by providing receipts. We modify the protocol into the code verification setting, proposed in the previous solutions, to relax the assumptions about the communications channels we use, requiring a set-up phase as a trade-off. We also study the properties of the visual sharing of shape descriptors and adapt the vote verification protocol so as to generate images that work as visual receipts of the votes for a relaxed security model.

Future work

An interesting direction is to design t-resilent visual shape descriptor schemes, whiteout seriously affecting the work required by the voter to verify her vote, which we want to be minimum. We also believe that it would be worthy to extend out visual voting scheme into a scheme with an arbitrary number of voting servers, using the notion of k visual sharing of shape descriptors, to enhance voter privacy guarantees. Such an adaptation would require to re-examine the format and the properties of the valid visual votes and shares, so that no critical information is leaked to reasonably sized coalitions of malicious voting servers. In addition it would be interesting to examine the possibility of designing visual code verification protocol with two or more voting servers. We also believe that it worthy to examine the application of visual sharing of shape descriptions to more general settings, like secure multiparty computation, where intermediate entities send data in part of users that need to verify the correct recording of their data from the other parts.

Abbreviations

DDH	Decisional Diffie Hellman
DL	Discrete Logarithm
E-Voting	Electronic Voting
EUF-CMA	Existential Unforgeability against Chosen Plaintext Attack
IND-CPA	Indistinguishability against Chosen Plaintext Attack
Mix-Net	Mixing Network
MS	Messenger
OT	Oblivious Transfer
PC	Personal Computer
POT	Proxy Oblivious Transfer
PK	Proof of Knowledge
PRF	Pseudo-random Function
VC	Vote Collector
VSSD	Visual Sharing of Shape Descriptors
ZKP	Zero Knowledge Proof

Bibliography

[1] "On E-Vote Integrity in the Case of Malicious Voter Computers", Sven Heiberg, Helger Lipmaa and Filip van Laenen. In Proceedings of 15th European Symposium on Research in Computer Security ESORICS '10, Springer, 2010.

[2] "Analysis of an Internet Voting Protocol", Kristian Gjøsteen. Technical Report 2010/380, International Association for Cryptologic Research, July 5, 2010.

[3] "Analysis of an Internet Voting Protocol", Kristian Gjøsteen. Technical Report, http://www.regjeringen.no , March, 2010.

[4] "The Norwegian Internet Voting Protocol", Kristian Gjøsteen. Proceedings of the Third International Conference on E-voting and Identity VoteID, Springer, 2011.

[5] "Pretty good democracy", Peter Y. A. Ryan and Vanessa Teague . In Workshop on Security Protocols, 2009.

[6] "Two Simple Code-Verification Voting Protocols", Helger Lipmaa. IACR Cryptology ePrint Archive 2011: 317, 2011.

[7] "SureVote", David Chaum. International patent WO 01/55940 A1 (02 August 2001), http://www.surevote.com/home.html

[8] "Priced Oblivious Transfer: How to Sell Digital Goods", William Aiello, Yuval Ishai, and Omer Reingold. In Proceeding of EUROCRYPT'01, Springer-Verlag 2001.

[9] "Oblivious Transfer And Polynomial Evaluation", Moni Naor and Benny Pinkas. Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing STOC'99, ACM Press, 1999.

[10] "Security and Trust for the Norwegian E-Voting Pilot Project *E-valg 2011*", Arne Ansper, Sven Heiberg, Helger Lipmaa, Tom André Øverland and Filip van Laenen. Proceedings of NordSec 14th Nordic Conference on Secure IT Systems, Lecture Notes in Computer Science Springer, 2009.

[11] "Transparency and Technical Measures to Establish Trust in Norwegian Internet Voting", O. Spycher, M. Volkamer, R. Koenig. Proceedings of the Third International Conference on E-voting and Identity VoteID, Springer, 2011.

[12] "Secure Electronic Voting Protocols", Helger Lipmaa. The Handbook of Information Security, John Wiley & Sons, 2005.

[13] "A Secure and Optimally Efficient Multi-Authority Election Scheme", Ronald Cramer, Rosario Gennaro and Berry Schoenmakers. In Proceedings of EUROCRYPT '97, Lecture Notes in Computer Science, Springer-Verlag, 1997.

[14] "Commitment Schemes and Zero-Knowledge Protocols", Ivan Damgård. Lectures on Data Security, Springer, 1999.

[15] "Non-interactive and information-theoretic secure verifiable secret sharing", T. P. Pedersen. In Advances in Cryptology/CRYPTO '91, Lecture Notes in Computer Science, Springer-Verlag, 1992.

[16] "How to Prove Yourself: Practical Solutions to Identification and Signature Problems", Amos Fiat and Adi Shamir, Crypto 1986.

[17] "Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols", Ronald Cramer, Ivan Damgard, and Berry Schoenmakers. In CRYPTO 1994, volume 839 of LNCS, pages 174-187, Springer-Verlag, 1994.

[18] "Guaranteed Correct Sharing of Integer Factorization with Off-Line Shareholders", W. Mao. In Proceedings of Public Key Cryptography 1998, LNCS, vol. 1431, pages 60-71, Springer, 1998.

[19] "Introduction to Modern Cryptography", Jonathan Katz and Yehuda Lindell. Chapman and Hall/CRC Press, 2007.

[20] Lecture notes in "Cryptography Primitives and Protocols", Aggelos Kiayias.

[21] "Introduction to Cryptography: Principles and Applications", Hans Delfs and Helmut Knebl, Springer, 2007.

[22] "Mix and match: Secure function evaluation via ciphertexts", M. Jakobsson and A. Juels. In ASIACRYPT, 2000.

[23] "Electronic Voting", Aggelos Kiayias. Handbook of Financial Cryptography, 2010.

[24] "Secure Vickrey Auctions without Threshold Trust", Helger Lipmaa, N. Asokan, and Valtteri Niemi. In Proceedings of the 6th international conference on Financial Cryptography '02, Springer-Verlag, 2002.

[25] "Additive Combinatorics and Discrete Logarithm Based Range Protocols", Rafik Chaabouni, Helger Lipmaa and Abhi Shelat. In Proceedings of ACISP '10, Springer-Verlag, 2010.

[26] "A threshold cryptosystem without a trusted party", T. Pedersen. In Advances in Cryptology EUROCRYP '91, volume 547 of Lecture Notes in Computer Science, pages 522-526, Springer-Verlag, 1991.

[27] "Visual Cryptography", M. Naor and A. Shamir. Proceedings of Eurocrypt '94, Springer, 1994.

[28] "Secret-Ballot Receipts: True Voter-Verifiable Elections", D. Chaum. IEEE Security and privacy magazine, 2004.