VOTING ON THE INTERNET

by

GURCHETAN S. GREWAL

A thesis submitted to the
University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
17 November 2015

Abstract

We address some of the challenges in achieving internet voting for real world elections. One challenge is that home-based computers are likely to be infected by malware, threatening both the integrity and privacy of the vote. Another concern is the possibility that a voter may be coerced to vote in a particular way, for example by a family member or organised crime ring. Moreover, any voting system intended to be used on a large scale should not require complex operations by voters whose purpose is hard to understand.

We introduce a series of novel proposals for internet voting, presented across three parts. First we examine how the problem of malware-infected computers in internet voting could be solved. We propose to use a dedicated hardware token (which is not required to be trustworthy) that helps remove the need to trust the voting computer and the server.

Second we examine how the outcome verification methods provided by internet voting can be made more intuitive. We show how using trial votes help voters achieve more intuitive verifiability.

Third we examine how the tension between verifiability and incoercibility can be reconciled while maintaining the usability of the voting systems. We propose a new property which we call "coercion-evidence" that helps improve usability, reduce trust assumptions, while maintaining the security of the system.

Acknowledgements

At the very outset, I would like to acknowledge my gratitude to my advisor, Mark Ryan. Without his patience, guidance, and thorough examination of the protocols, this thesis would not be the same what it is today. He has helped me explore ideas, to critically evaluate my work, and to express my results clearly. His enthusiastic encouragement and support helped me a lot in my personal and professional development.

I have also benefited from my research monitoring group members, Tom Chothia and Uday Reddy. Their comments and suggestions have informed my research, in more ways than one. Tom introduced me to research during my MSc project and motivated me to do a PhD, thank you very much for that.

During my PhD, I have had the opportunity to work with Sergiu Bursuc, Liqun Chen, Michael Clarkson, and Peter Ryan. I learned a lot from them, both technically and on a range of other subjects. I am truly grateful to all of them for their efforts.

I also made some good friends during the course of my PhD in the school of computer science. Andreea Radu, Mihai Ordean, Flavio Garcia, David Oswald, Loretta Mancini, Lenka Mudrova, Vivek Nallur, Masoud Koleini, Vincent Cheval, Michael Mistry, Gurpreet Grewal-Kang, Myrto Arapinis, Ben Smyth, Miriam Paiola, Padma Reddy, Abhinav Mehrotra, Khulood AlYahya, Esra Alzaghoul, Marwah Alansari, Volker Sorge, Paul Levy, and many others have contributed to a rich, and fulfilling period of stay at Birmingham.

I would like to especially thank one person who has been with me through many of my major life events during my PhD: Shehnila Zardari. Thanks for always being there, as a person to talk to and a support to lean on. My journey through the PhD would have been lonelier, if Shehnila wasn't here in this department.

Some friends are somehow always around you irrespective of where you are and what you are doing. Arti Pundir, Tashika Sindhar, Cherry, Shruti Sharma, Peter Singla, Amrit Parimoo, Raman Gill, and Anand Sharma are among those friends. Thank you very much for your blessings my dear friends.

My brother, Rajinder Grewal, deserves a big thank you for helping me in every possible

way. I have learned so much from you, Rajinder, thank you.

In addition to those already mentioned, I received very useful comments and feedback on parts of this thesis. I am thankful to Jeremy Clark, Veronique Cortier, Aleksander Essex, David Galindo, Rui Joaquim, Steve Kremer, Olivier Pereira, Ron Rivest, Peter Rønne, and Steve Schneider.

And finally, I would like to thank my parents who are the most important people in my life. Mom and Dad, thank you very much for supporting and encouraging me to achieve my dreams and ambitions. I love you, and to you both, I dedicate this thesis.

UNIVERSITY OF
BIRMINGHAM

This research was primarily funded by the Engineering and Physical Sciences Research Council as part of the project *Trustworthy Voting Systems* (EP/G02684X/1). Additional funding came from the project *Analysing Security and Privacy Properties* (EP/H005501/1). The University of Birmingham provided funding to cover the international student fee. The George Washington University partly supported research work done for four months at The George Washington University, Washington DC, US.

Statement of collaboration

The research in Chapter 3 is my work in collaboration with Mark Ryan, Michael Clarkson, and Liqun Chen. I developed the main idea, system, and security analysis. The construction of the zero knowledge proof in that chapter was mostly done by Liqun Chen and Mark Ryan. Chapter 4 is my work in collaboration with Sergiu Bursuc and Mark Ryan. The main idea is my own and it was supervised by Sergiu Bursuc and Mark Ryan. The research in Chapter 5 is my work in collaboration with Mark Ryan, Sergiu Bursuc, and Peter Ryan. The main idea emerged from Mark Ryan and Peter Ryan. I developed the idea, definitions, system, and proofs under Mark and Sergiu's supervision. The proof of coercer independence was mostly done by Sergiu Bursuc.

Contents

List of Tables

List of Figures

CHAPTER 1

Introductory remarks

The elections play a very important role in any democratic entity and, therefore, it is crucial to conduct elections in a trustworthy manner. In the traditional voting systems most of the trust lies in the hands of few people who are responsible for counting the ballots. This single point of trust could become vulnerable to corruption, and attacks on the integrity of the system – bribing the people who are responsible for counting the votes can change the entire outcome of the elections. The governments around the world have also been investing in technologies that could be used to run elections in a secure manner. For example, using computers for voting and counting in the elections could help as computers can't be bribed, and in case of any issues these computers can be audited and actual problem could be detected. Moreover, this opens the scope for internet voting which along with better accuracy has the potential to increase voter participation. In a survey, 60.6% of people overall said they are more likely to vote if they could vote over the internet, and 81.7% of the 18-35 year age group [Nat14].

For all its advantages, however, internet voting is very challenging to secure [KSRW04, CFH+07, BCE+07, CS11, WWH+10, KTV12, WWIH12]. In contrast with internet banking and social networking, it is not easy to put mistakes right if they are uncovered after the results of the election have been declared. Moreover, it is notoriously difficult to prove that the *software* that might be running on a voter's computer or on the server has the expected properties. Nonetheless, internet voting systems are being deployed for national elections [HLW11, Gjø11] and organizational elections [dMPQ09, Uni15, BVQ10, HBH10]. This deployment has been enabled, in part, by improvements in cryptographic protocols that ensure ballot *privacy* as well as *verifiability* of the election outcome, even when the authorities and servers running the election cannot be trusted. The rise of the concept of *election outcome verifiability* or *end-to-end verifiability* has also helped in encouraging this deployment – anyone can verify the results based on the inputs and outputs of the voting system without relying on the software that is running behind the scenes. Moreover, it is more practical to verify the results produced by the software, than the software itself. Therefore, election outcome verifiability is sometimes referred to as *software independence*.

To achieve election outcome verifiability encrypted data about the elections is posted on a public bulletin board (for example, a website) in a way that anyone could verify the correctness without revealing the actual votes.

To ensure free and fair elections, ballot secrecy is another vital aspect. A strong form of ballot secrecy, which is known as incoercibility, ensures that voters cannot convince anyone else about how they voted, with high probability. This prevents them from selling their vote, or being coerced i.e. forced to chose a particular candidate. Coercion is not an issue in traditional voting methods, because the polling booth provides a completely private place where a voter cannot be influenced by the coercer at the time of voting. In Internet voting, coercion could take many forms – voters could be coerced by a family member, an employer or by organized criminals. To avoid coercion and to achieve verifiability at the same time, strong cryptography is used, which ends up making these systems difficult to use by voters. This becomes even more difficult in the presence of a malicious computer. A specially-produced malware could affect the vote at the time it is cast on a voter's home computer. Therefore, a very careful approach needs to be taken for making internet voting systems verifiable while preserving privacy. This motivated us to find techniques to make internet voting more trustworthy by reconciling verifiability and privacy while keeping usability and the problem of malicious computers in our mind.

Real world implementations: Some countries are already using internet voting for legally binding elections. In Estonia, it was first used in 2005 for local elections and in 2007 for parliamentary elections [HLW11]. Voters use Estonian ID cards for authentication, which is a mandatory national identity document. The Estonian system uses procedural controls to ensure transparency and to guard against attacks. However, Springall et al. have shown that those procedures are insufficient and have recommended Estonia to discontinue use of their current internet voting system [SFD+14].

Norway also carried out internet voting pilot projects during the parliamentary elections in 2011 and 2013 [Gjø11, AHL+09]. These were later discontinued as it didn't improve

voter turnout and due to the concerns that the voters have limited knowledge about the security mechanisms in the system [Nor14].

Another system called Helios has been successfully used in elections where coercion is not considered to be a serious concern. Université catholique de Louvain in Belgium has used Helios for electing the university president in 2009 [dMPQ09]. The International Association for Cryptologic Research (IACR) also ran mock elections using Helios in 2010 [BVQ10]. This system gives anyone the ability to audit elections. It takes reasonable steps as well to preserve voter privacy.

Coercion and vote buying are among the problems in internet voting that are difficult to avoid and therefore some electronic voting systems have been designed to be used in polling stations. End-to-end verifiable voting systems like Scantegrity and Prêt à Voter are used in controlled environments (polling stations); this enables them to give stronger security guarantees compared to internet voting systems. Scantegrity II was successfully used in municipal elections at Takoma Park [CCC+09, CCC+10]. The Victorian Electoral Commission (VEC) in Australia used a system based on Prêt à Voter in Victoria's state election in 2014 [BCH+12].

1.1 Problem statement

Much work has been done in recent decades to make electronic voting a reality [Adi06, Cla11, Ess12]. But the conflict in the properties that internet voting systems are expected to achieve, has made it difficult to conduct internet voting for large scale elections. Many of the complexities in internet voting schemes arise because of the tension between election verifiability ("transparency") and voter privacy ("opacity"). The existing systems that achieve both of these properties are complex to use by the voters and therefore can't be deployed for legally binding election in the near future. The verifiability property in such systems is usually achieved by using cryptographic proofs which are non-intuitive, and unduly burden users to verify them [SLC+11]. Moreover, such systems typically assume

the computers used by the voters to submit their votes are trusted for privacy [Adi08, HLW11, Gjø11], or for both privacy and verifiability [JCJ05, CCM08]. Some estimates are that 30–40% of home computers are infected, and one has to assume that determined attackers could produce and distribute malware specifically designed to thwart a national election [Sam12, Dan10]. It is crucial to find a trade-off between the security properties which seem to be contradicting with usability properties.

1.2 Thesis statement

We briefly argued in the previous section that there is a tension between election verifiability and stronger forms of voter privacy. An important question therefore is whether this tension could be mitigated while maintaining usability.

Thesis Statement: *We aim to demonstrate that the conflicting properties of verifiability and privacy can be reconciled for internet voting while maintaining the usability of the systems, even in the presence of untrusted platforms.*

Objective and Scope: The objective of this thesis is to present techniques to balance verifiability and privacy for future deployable internet voting systems. We also aim to reduce trust in the computers used by voters during the election. In particular, we will propose solutions to meet these three objectives:

- **Objective 1: Improve verifiability and privacy in the presence of untrusted computers.** Provide solutions to improve verifiability when the computers used during the election might be untrusted, while preserving voter privacy.
- **Objective 2: Improve individual verifiability for voters.** Propose techniques to make individual verifiability more intuitive for voters without increasing complexity on the voter's side.

- **Objective 3: Improve usability by relaxing coercion-resistance.** Propose techniques to relax coercion-resistance while maintaining same level of incoercibility, to improve usability for voters.

Ideally we would like to achieve above mentioned objectives in a single voting system. However, it is not possible at the moment under reasonable trust assumptions and, therefore, more research is needed. The independent contributions presented in this thesis will let election administrators pick and choose the properties they want. This will help in the future to design internet voting systems that closely match the requirements of the users.

1.3 Overview of this thesis

An overview of this thesis is outlined below.

- **Chapter 2: Preliminaries and related work**

  In this chapter, we describe the desired properties for internet voting systems. We recall the cryptographic primitives (the basic building blocks) that are used in this thesis. We present previous work that is related to this thesis. This chapter also presents a detailed overview of the internet voting system JCJ/Civitas, which is required to understand work presented in Chapter 4 and Chapter 5.

- **Chapter 3: Du-Vote: Internet voting with untrusted computers**

  In Chapter 3 we present the design of Du-Vote; an internet voting protocol that eliminates the often-required assumption that voters trust general-purpose computers. Trust is distributed in Du-Vote between a simple hardware token issued to the voter, the voter's computer, and a server run by election authorities. Verifiability is guaranteed with high probability even if all these machines are controlled by the adversary, and privacy is guaranteed as long as at least either the voter's computer, or the server and the hardware token, are not controlled by the adversary. A new non-interactive zero-knowledge proof is employed to verify the server's computations.

Du-Vote is a step towards tackling the problem of internet voting on user machines that are likely to have malware. We anticipate that the methods of Du-Vote can be used in other applications to find ways of securely using platforms that are known or suspected to have malware.

- **Chapter 4: Trivitas: Voters directly verifying votes**

  Individual verifiability is the ability of an electronic voting system to convince a voter that her vote has been correctly counted in the tally. Unfortunately, in most electronic voting systems the proofs for individual verifiability are non-intuitive and, moreover, need trusted computers to check these proofs. Based on the remote voting system JCJ/Civitas, in this chapter we propose Trivitas, a protocol that demonstrates an intuitive way of achieving individual verifiability, while at the same time preventing coercion.

  The technical contributions of this chapter rely on two main ideas, both related to the notion of credentials already present in JCJ/Civitas. Firstly, we propose the use of trial credentials, as a way to track and audit the handling of a ballot from one end of the election system where a voter submits her vote, to the other end where all the votes are counted, without increased complexity on the voter end. Secondly, due to indistinguishability of credentials from random values, we observe that the association between any credential and its corresponding vote can be made public at the end of the election process. We also observe that this doesn't make voters vulnerable to coercion. This way, the voter has more intuitive and direct evidence that her intended vote has not been changed and will be counted in the final tally.

- **Chapter 5: Caveat Coercitor: Coercion evidence in electronic voting**

  The balance between the requirement of election verifiability, coercion-resistance: inability of a coercer to coerce voters, and usability remains unresolved in internet voting despite significant research over the last few years. In this chapter, we propose a change of perspective, replacing the requirement of coercion-resistance with a new

requirement of coercion-evidence: there should be public evidence of the amount of coercion that has taken place during a particular execution of the voting system.

We provide a formal definition of coercion-evidence that has two parts. Firstly, there should be a *coercion-evidence test* that can be performed against the bulletin board to accurately determine the degree of coercion that has taken place in any given run. Secondly, we require *coercer independence*, that is the ability of the voter to follow the protocol without being detected by the coercer.

To show how coercion-evidence can be achieved, we propose an internet voting scheme, Caveat Coercitor, and we prove that it satisfies coercion-evidence. Moreover, we argue that Caveat Coercitor makes weaker trust assumptions than other remote voting systems, such as JCJ/Civitas and Helios, and at the same time has better usability properties.

- **Chapter 6: Discussion, further work, and conclusion**

  We discuss what we have achieved with respect to the objectives of this thesis, present possible further research work and open questions that have emerged, and offer some concluding remarks.

1.3.1 Publications

In this section we provide a list of publications that were generated during work on this thesis. This thesis contains, and must be considered, the definitive reference of details and ideas, presented in these publications.

- **Conference**

  1. Gurchetan S. Grewal, Mark D. Ryan, Liqun Chen, and Michael R. Clarkson. Du-Vote: Remote electronic voting with untrusted computers. In *Proceedings of the 28th IEEE Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 155–169. IEEE, 2015

2. Gurchetan S. Grewal, Mark D. Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. Caveat Coercitor: Coercion-Evidence in electronic voting. In *2013 IEEE Symposium on Security and Privacy, (S&P 2013), Berkeley, CA, USA, May 19-22, 2013*, pages 367–381. IEEE Computer Society, 2013

3. Sergiu Bursuc, Gurchetan S. Grewal, and Mark D. Ryan. Trivitas: Voters directly verifying votes. In *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers*, volume 7187 of *Lecture Notes in Computer Science*, pages 190–207. Springer, 2011

- **Miscellaneous**

4. Mark D. Ryan and Gurchetan S. Grewal. Online voting is convenient, but if the results aren't verifiable it's not worth the risk. https://theconversation.com/online-voting-is-convenient-but-if-the-results-arent-verifiable-its-not-worth-the-risk-41277, 2015. The Conversation website [Online; accessed 21-July-2015]

5. Mark D. Ryan and Gurchetan S. Grewal. Digital Democracy lets you write your own laws. http://www.theconversation.com/digital-democracy-lets-you-write-your-own-laws-21483, 2013. The Conversation website [Online; accessed 04-Jan-2015]

6. Mark D. Ryan and Gurchetan S. Grewal. Internet voting: coming to a computer near you, though more research is needed to eliminate the risks. http://www.democraticaudit.com/?p=3149, 2014. Democratic Audit blog [Online; accessed 04-Jan-2015]

CHAPTER 2

Preliminaries and related work

2.1 Desired properties

In this section, we briefly describe the properties that electronic voting systems are expected to satisfy. We first discuss the core requirements of verifiability and privacy, then we go on to briefly explain two other desirable properties: fairness and usability. For the core requirements we will discuss various sub-requirements as well.

2.1.1 Verifiability

Verifiability is an integrity property. It assures the voters that their votes are recorded in the system and the election outcome is computed correctly. Any attempt to corrupt the integrity of the elections must be detected. Election verifiability in presence of a bulletin board may be conveniently split in three notions [KRS10]:

- **Individual verifiability:** To any voter, individual verifiability should offer the possibility to verify that her cast ballot has been correctly recorded and tallied by the system [Cha04, Nef04, AN06]. Ideally, the voter should also have an assurance that her cast ballot correctly encodes her cast vote. In some systems this additional assurance is offered by an option to audit a ballot [Ben06, Ben07, Adi08] or a ballot form [CRS05] before casting a vote.

- **Universal verifiability:** To any external observer, universal verifiability should offer the possibility to verify that all recorded votes have been correctly tallied [JJR02, Nef01, FS01, HS00].

- **Eligibility verifiability:** To any external observer, eligibility verifiability should offer the possibility to verify that the set of tallied votes corresponds to votes cast by eligible voters [JCJ05, CCM08, KRS10]. Eligibility may be enforced at any stage in the system: either during the casting of a vote [CRS05], or during its recording [Adi08], or during its tallying [JCJ05, CCM08].

2.1.2 Privacy

Privacy is a confidentiality property. It assures the voter that no one can learn the link between her identity and her vote. It can be divided into three types based on the strength of privacy:

- **Ballot secrecy:** To any voter, ballot secrecy guarantees, with a high probability, that the way she voted is not revealed to anyone by the voting system. This property assures that the link between a voter and her vote remains secret [KR05, DKR06, DKR09]. For instance, if *Alice* and *Bob* be two arbitrary honest voters voting *yes* and *no* respectively, then even if they swap their votes an attacker cannot know how they voted. This is the weakest privacy property as it assumes that the voter doesn't voluntarily reveal how she voted.

- **Receipt-freeness:** A voter after participating in an election is unable to later prove to someone how she voted by revealing the information she obtains while interacting with the voting system [BT94, Oka96, DKR06, MN06, DKR09]. In other words, a voting system satisfies receipt-freeness if an attacker cannot distinguish with high probability between a situation where a voter genuinely cooperates with him to cast a vote to the attackers desired candidate and a situation where she pretends to cooperate but casts a different vote, based on the information provided by the voter and the voting system. This property is stronger than ballot secrecy as it doesn't let the voter later prove how she voted even if she is willing to do so.

- **Coercion-resistance:** A coercer interactively communicating with a voter to instruct her to vote in a certain way is unable to distinguish if the voter followed her instructions or not, even if the voter is willing to cooperate [Oka97, DKR09, JCJ05, DKR06]. More precisely, for any strategy available to the coercer there exist a strategy for the voter that allows her to cast her intended vote while it is indistingushable to the coercer with high probability from her casting a vote according to the coercer's intent. This is the strongest privacy property and it

subsumes receipt-freeness and ballot secrecy.

As observed by authors in [Cla11, Ess12], receipt-freeness was proposed mainly in the literature in the context of polling station voting, whereas coercion-resistance was proposed for internet voting. The primary difference is that an attacker has more capability in coercion-resistance as compared to receipt-freeness [DKR09]. In coercion-resistance voter gives any information that she gets during the voting process to the attacker and use information that the attacker provides to her. Whereas in receipt-freeness the attacker just receives information from the voter and any publicly available information. A comprehensive analysis of various ballot privacy definitions is presented in [BCG+15].

2.1.3 Other properties

Along with the above properties, some other properties are also important:

- **Fairness:** No early results can be obtained which could influence the remaining voters. The voting system should not favour any candidate in any way.
- **Usability:** The voting system should be easy to understand and use by the voters.

All these properties should be satisfiable even in the presence of the corrupt election authorities.

2.2 Cryptographic primitives

In this section, we briefly outline some of the main cryptographic primitives that will be used in this thesis. These cryptographic primitives are the basic building blocks for electronic voting protocols, and these are standard among the electronic voting literature.

Notations for encryption, and security assumption: Encryption of a message m with a public key y using randomness r is denoted $\mathsf{enc}(m; y; r)$. We omit y when it is clear from context. And when the particular random value r is not relevant, we omit it. For simplification, we also sometimes write encryption of m as em. Decryption of a ciphertext c with a private key z is denoted $\mathsf{dec}(c; z)$. We omit z when it is clear from context.

Throughout this thesis, we assume that the encryption enc is perfect: given a ciphertext $\mathsf{enc}(m; y; r)$, the plaintext m can only be recovered by the holder of the corresponding private key z (we assume the random r has been discarded with the encryption).

Distributed ElGamal: The encryption scheme being used is the ElGamal encryption scheme [Gam84, Gam85] and we will make use of distributed decryption [DF89, BCGG99, Ped91b, Bra05]. It works over a multiplicative group of integers modulo a prime $p = 2kq+1$, where q is also prime. The plaintext space and the ciphertext space $\mathcal{M}$ is of the order q subgroup of $\mathbb{Z}_p^*$. Each party involved in the decryption process has a private key, which is a number $z_j \in \mathbb{Z}_q^*$ and a corresponding share of the public key y_j, where $y_j = g^{z_j} \mod p$. The product $\prod_j y_j$ of all the public keys can itself be used as a public key, denoted y. The encryption of a message m with a public key y i.e. $\mathsf{enc}(m; y; r)$ is a pair $(g^r \mod p, m \cdot y^r \mod p)$ where r is a fresh random number in $\mathbb{Z}_q^*$. Decryption of a ciphertext encrypted under y requires participation of all the tellers, each using their own private key. To decrypt a ciphertext (a, b) each party publishes a partial decryption share $a_i = a^{z_i} \mod p$ using their private key z_i. The final decryption can then be publicly computed as $\frac{b}{a_1 \cdots a_n} \mod p$ (where n is the number of distributed parties). The scheme works over a cyclic group $\mathcal{G}$ with generator g.

Exponential ElGamal: We also consider an *exponential* version of ElGamal encryption, in which during encryption the plaintext message is also used as an exponent of the group generator [CGS97]. The encryption of a message m with a public key y is a pair $(g^r \mod p, g^m \cdot y^r \mod p)$ where r is a random number. In Chapter 3, we use a variant

of exponential ElGamal in which another group generator h is raised to the power of the plaintext message. Exponential encryption of message m with public key y using randomness r is denoted $\mathsf{enc\text{-}exp}(m; y; r)$, where $\mathsf{enc\text{-}exp}(m; y; r) = (g^r, h^m y^r)$. Sometimes we omit parameters to $\mathsf{enc\text{-}exp}$, as with enc. The security of the scheme relies on the standard assumption that it is computationally infeasible to determine $\log_g h$—that is, the discrete logarithm relationship between g and h [Ped91a].[1]

The exponential ElGamal encryption scheme satisfies the homomorphic property. Let a ciphertext encrypting a plaintext m under the public key y be $C = (C_1, C_2) = (g^r, h^m y^r)$. Given two ciphertexts $\hat{C} = (\hat{C}_1, \hat{C}_2) = (g^{\hat{r}}, h^{\hat{m}} y^{\hat{r}})$ and $\bar{C} = (\bar{C}_1, \bar{C}_2) = (g^{\bar{r}}, h^{\bar{m}} y^{\bar{r}})$, a new ciphertext, written as $\tilde{C} = (\tilde{C}_1, \tilde{C}_2) = (g^{\tilde{r}}, h^{\tilde{m}} y^{\tilde{r}})$, can be created by computing $\tilde{C}_1 = \hat{C}_1 \bar{C}_1$ and $\tilde{C}_2 = \hat{C}_2 \bar{C}_2$, where $\tilde{r} = \hat{r} + \bar{r}$ and $\tilde{m} = \hat{m} + \bar{m}$. Therefore, $\mathsf{enc\text{-}exp}(m_1; y; r_2) \cdot \mathsf{enc\text{-}exp}(m_2; y; r_2) = \mathsf{enc\text{-}exp}(m_1 + m_2; y; r_1 + r_2)$.This homomorphic property is commonly called the additive homomorphic property. The homomorphic property of the ElGamal scheme is used to reencrypt a ciphertext (or encrypted vote) in order to hide the plaintext (or vote) from its original encryptor in order to guarantee the public verification.

Re-encryption and mix nets: Given a ciphertext $\mathsf{enc}(m; y; r)$ constructed using the public key y any party can compute another ciphertext $\mathsf{enc}(m; y; r + r')$ that encrypts the same plaintext using the same key y, by using a new random r' [PIK93, Cha03, JJR02]. We will denote the re-encryption of a ciphertext $\mathsf{enc}(m; y; r)$ (having plaintext message m) with a given random r' by $\mathsf{renc}(m; y; r')$. We omit y when it is clear from context. Re-encryption is sometimes also referred to in the literature as Re-randomization.

A re-encryption mix net is a set of agents Mix that takes as input a sequence of ciphertexts $\mathcal{S} = m_1, \ldots, m_k$ and outputs a sequence of ciphertexts $\mathcal{S}' = m'_1, \ldots, m'_k$ that is a re-encryption mix of $\mathcal{S}$. Specifically, $\mathcal{S}'$ is a formed by re-encryption of elements in

[1] One way to realize this assumption is based on a *Schnorr group*. Let p and q be large prime numbers, such that $p = \ell q + 1$ for some ℓ. Define Schnorr group $\mathcal{G}$ to be an order-q subgroup of $\mathbb{Z}_p^*$. To construct g and h from $\mathcal{G}$, first choose any two bit strings a and b, such that $a \neq b$. Then let $g = (\mathsf{hash}(a) \mod p)^\ell$ and $h = (\mathsf{hash}(b) \mod p)^\ell$. Check that $g \neq 1$ and $g \neq -1$, and likewise for h. Also check that $g \neq h$. If any checks fail (which happens with low probability), then try again with some other values for a and b.

a permutation of $\mathcal{S}$: there is a permutation σ of $\{1, \ldots, k\}$ and a sequence of randoms $r_1, \ldots, r_k$ such that $m'_1 = \mathsf{renc}(m_{\sigma(1)}; r_1), \ldots, m'_k = \mathsf{renc}(m_{\sigma(k)}; r_k)$. Moreover, if at least one element of Mix is not controlled by an adversary $\mathcal{C}$, the permutation σ remains secret to $\mathcal{C}$.

Plaintext equivalence test: Given two ciphertexts $\mathsf{enc}(m_1; y; r_1)$ and $\mathsf{enc}(m_2; y; r_2)$, encrypted with the same key y, whose plaintexts are m_1 and m_2 respectively, a plaintext equivalence test (pet) allows the holders of the decryption key to demonstrate that $m_1 = m_2$, without revealing the decryption key or any information about m_1 or m_2 [JJ00]. Consider two ElGamal ciphertext pairs (a_1, b_1) and (a_2, b_2), roughly, holders of the decryption key compute $(a, b) = (\frac{a_1}{a_2}, \frac{b_1}{b_2})$ and perform a distributed decryption of (a, b); if the result of the decryption is 1, then the encrypted messages are equal, otherwise they are not equal.

For two ciphertexts c_1 and c_2, we will denote by $\mathsf{pet}(c_1, c_2) = \mathsf{ok}$ iff the plaintext equivalence test holds for c_1 and c_2. Key holders only provide pets as indicated by the protocol.

Zero-knowledge proofs: A zero-knowledge proof (ZKP) is a proof between a prover and a verifier that demonstrates to the verifier that a certain statement is true, without revealing any information except the validity of the statement [GMR89]. Non-interactive zero knowledge proofs (NIZKP) are a variant of ZKP that do not require interaction between the prover and the verifier. If the statement being proved asserts that the prover knows a secret signing key, this type of NIZKP can be a signature on some arbitrary value using the signing key, and is called a signature-based proof of knowledge (SPK). A Schnorr signature [Sch91] is commonly used for this purpose.

Decryption, re-encryption mixnets and plaintext equivalence tests can be accompanied by non-interactive zero-knowledge proofs that attest of the fact that these operations have been correctly performed, without revealing sensitive information like the decryption key. Zero-knowledge proofs are crucial in order to ensure universal verifiability of the election,

while preserving user privacy. We will denote by $\mathsf{petproof}(c_1, c_2, res)$ a zero-knowledge proof that the result of the plaintext equivalence test applied to c_1 and c_2 is res.

Ring signatures and OR-proofs: Consider a set $\{P_1, \ldots, P_n\}$ of participants, and suppose that each participant P_i has a private key sk_i and a corresponding public key pk_i. We suppose P_i possesses his own private key sk_i and all of the public keys $\{pk_j\}_{1 \leq j \leq n}$. A *ring signature* is a cryptographic signature that such a participant P_i can make, and has the property that any verifier in possession of the public keys $\{pk_j\}_{1 \leq j \leq n}$ can be assured that one of the participants $\{P_1, \ldots, P_n\}$ made the signature, without being able to tell which one it was [AOS02, CLMS08]. In a ring signature, there is one "actual" signer, in our example P_i; the signing process uses his secret key sk_i and the public keys $\{pk_j\}_{1 \leq j \leq n,\, j \neq i}$ of the other ring members. The actual signer does not need the permission or involvement of the other ring members.

If the statement being proved asserts that the prover knows a secret value satisfying a disjunction of properties, then a ring signature can be used. For example, given a finite set $\Omega = \{x_1, \ldots, x_n\}$, a group generator g, and a function f, we may prove in zero knowledge that we know a d such that $g^d = f(x_1)$ or $\ldots$ or $g^d = f(x_n)$. Such a proof is a ring signature on an arbitrary value, where the actual signing key is d and the public keys are $\{f(x) \mid x \in \Omega\}$. We write such a signature proof of knowledge as

$$SPK\{d \mid \exists x : x \in \Omega \wedge g^d = f(x)\}.$$

It is useful to generalise this to a *double signature*. Let Ω be a set of pairs, and g_1, g_2 be two group generators, and f_1, f_2 be two functions. We write

$$SPK\{d \mid \exists x_1, x_2 : (x_1, x_2) \in \Omega \wedge g_1^d = f_1(x_1) \wedge g_2^d = f_2(x_2)\}$$

to mean a proof of knowledge of d satisfying $g_1^d = f_1(x_1)$ and $g_2^d = f_2(x_2)$ for some $(x_1, x_2) \in \Omega$. The signer's private key is d, and the ring of public key pairs is $\{(f_1(x_1), f_2(x_2) \mid$

$(x_1, x_2) \in \Omega\}$.

A signature-based proof of knowledge and OR-proof are also used as zero-knowledge proofs. Indeed, we use them in Chapter 3.

2.3 Related work

In recent years we have seen technology help people become more involved in debate about all aspects of society. So it seems that technology can play a much greater role in political participation as well. With this hope a lot of work on electronic voting has been done in past 30 years. In this section we will briefly describe some of those systems that are relevant to our research. We will mainly focus on some internet voting protocols that: (1) provide verifiability of elections, (2) make efforts to resist coercion, or (3) allow voting to be conducted on an untrustworthy platform.

2.3.1 Coercion-resistance

Coercion-resistance is a fundamental, and strong, property of internet voting systems. It states that a voter should be able to cast her vote as intended, even in presence of a coercer that may try to force her to cast a different vote. A formal definition of coercion-resistance based on observational equivalence is proposed in [DKR09]. They prove that the Lee protocol [LBD+03] satisfies it. Another definition based on observational equivalence is proposed in [BHM08], where an automated proof with ProVerif has been carried out for JCJ/Civitas.

2.3.1.1 JCJ/Civitas

Juels et al. [JCJ05] proposed the first internet voting system that achieves coercion-resistance and verifiability. It was implemented by Clarkson et al. as Civitas [CCM08]. We refer to these systems as JCJ/Civitas. Chapter 4 and Chapter 5 of this thesis are

closely related to JCJ/Civitas, therefore we describe it in detail here.

The main idea in JCJ/Civitas is the notion of credentials (with a private and a public part), that allow eligible voters to authenticate their ballots. To allow coercion-resistance, JCJ/Civitas distributes credential generation among a set of parties called registrars. To resist coercion, the voter has the ability to generate a fake credential and fake proof, that is indistinguishable from the real credential and real proof for the coercer.

The participants of the protocol are

R - the set of registrars, whose role is to authenticate eligible voters and help generate their credentials.

T - the set of tellers, whose role is to generate and publish the public key of the election. Each of them holds a secret share of the corresponding private key, that will be used for distributed decryption and plaintext equivalence tests.

$V_1, \ldots, V_n$ - the set of eligible voters.

P - voting platform/computer used by a voter.

Mix - a re-encryption mix net, whose role is to anonymize the set of cast ballots before verification of their eligibility and their decryption.

BB - the bulletin board, whose role is to record the manipulation of ballots at all stages of the election, from the voting phase to the tallying phase. It is a repository of the information collected throughout the election and this information is made publicly available for anyone to inspect [CS14]. It also records proofs of correct ballot handling submitted by R, T and Mix, that can be checked by external auditors.

Trust assumptions: A coercer may control some of $V_1, \ldots, V_n$, some of R, some of T and some of Mix but not all. For a voter V_i to achieve coercion-resistance, at least one V_j, $j \neq i$, one of R, one of T and one of Mix, and the voting platform P must be outside the control of the coercer. Moreover, there must be an untappable channel from V_i to

the trusted registrar, so that the coercer can not get hold of the real credential, and an anonymous channel for the voter to submit the ballot. The voter must also trust the voting platform to correctly construct the voting ballot and to verify the zero-knowledge proofs provided by registrars. A summary of the protocol is as follows:

Initialisation: The election starts with tellers T generating the public key pk_T of the election in a distributed manner, such that no minority of tellers can recover the private key sk_T [Ped91a] and the decryption is distributed [Bra05] as discussed in section 2.2. The public part of the key is published on the bulletin board.

Voting credentials: In order to cast their vote, every eligible voter needs credentials. Each credential has two parts: public and private. The voter receives private part of the credentials from the registrars which is explained in the registration phase. Public part of the credentials is the encryption of the private part with some random and public key pk_T of the tellers T. Each private credential can have more than one public credentials. Private credentials will be denoted by the letter s (decorated with various indices). For a given private credential s, there exists a corresponding public credential $\mathsf{enc}(s; pk_T; r)$, which will be published on the electoral roll. We sometimes denote public credentials (and their re-encryptions) by the letter es (decorated with indices).

Registration: By running a separate protocol with each of the registrars, the voter V obtains a private share s_V^i of her voting credential. Let $m = |R|$. The private voting credential of the voter is the sum of all private credential shares, i.e. $s_V = s_V^1 + \ldots + s_V^m$, and the public credential is the homomorphic combination of all public credential shares registered on the electoral roll, i.e. $\mathsf{enc}(s_V; pk_T; r) = \mathsf{enc}(s_V^1; pk_T; r_1) * \ldots * \mathsf{enc}(s_V^m; pk_T; r_m) = \mathsf{enc}(s_V^1 + \ldots + s_V^m; pk_T; r_1 + \ldots + r_m)$.

Validity of voter credentials: Each registrar also provides the voter with a non-transferable

proof (designated-verifier re-encryption proof) P_Q of the fact that the public share $\mathsf{enc}(s_V^i; pk_T; r_i)$, that is published on the electoral roll ER, correctly encodes the private part s_V^i [HS00]. The proof can be verified on the voting platform of the voter.

Resisting coercion: The voter has the ability to construct a fake credential by replacing the credential share s_V^i of a trusted registrar with a fake credential share s'^i_V. The voter also has the ability (with the help of the voting platform) to construct a fake proof P'_Q that causes $\mathsf{enc}(s_V^i; pk_T; r_i)$ to appear as an encryption of s'^i_V.

Voting: The ballot $(\mathsf{enc}(s_V; pk_T; r_s), \mathsf{enc}(\nu; pk_T; r_\nu), \mathsf{P}_{s\nu}, \mathsf{P}_{\mathsf{corr}})$, from the voter V, contains the encryption of the private credential s_V (with the key pk_T and with a different random than in the electoral roll) and the encryption of the intended vote ν (with the key pk_T). To prevent the re-use of the same credential by a party that does not hold the private part, the ballot contains additionally a zero-knowledge proof $\mathsf{P}_{s\nu}$ of the fact that its creator knows both s_V and ν. Additionally, a zero-knowledge proof $\mathsf{P}_{\mathsf{corr}}$ proves that ν is a valid vote, according to the specification decided by election authorities. The ballot $(\mathsf{enc}(s_V; pk_T; r_s), \mathsf{enc}(\nu; pk_T; r_v), \mathsf{P}_{s\nu}, \mathsf{P}_{\mathsf{corr}})$ is constructed by the voting platform and submitted to the bulletin board. All such ballots are added to the set of cast ballots that we call as BB_{cast}.

Verification of proofs and mixing: Before tabulation starts, the zero-knowledge proofs $\mathsf{P}_{s\nu}$ and $\mathsf{P}_{\mathsf{corr}}$ of cast ballots, present in the set BB_{cast}, are verified (e.g. by the tellers) and ballots with invalid proofs are discarded. The set of cast ballots with valid zero-knowledge proofs of correctness are stored in the set BB_{valid}. The valid ballots (without the proofs) and the credentials from the electoral roll are then sent to the re-encryption mix net Mix for anonymization. The mixed ballots are recorded in the set BB_{mix}, and the mixed encrypted electoral roll as the set BB_{aer}.

Tallying: Credentials from anonymized ballots (i.e. from the set BB_{mix}) are compared, using plaintext equivalence tests (pet), to credentials from the anonymized electoral roll (BB_{aer}), to ensure that votes to be counted are cast by eligible voters only. If multiple ballots are submitted with the same credentials, only one copy is kept according to a predefined policy, e.g. only the last vote counts. Duplicate elimination is done by plaintext equivalence tests and can be performed either before, or after the mix. Finally, the decided set of countable votes is decrypted.

The re-encryption mix, the plaintext equivalence tests, and the decrypted votes are accompanied by zero-knowledge proofs that ensure the operations have been correctly performed.

2.3.1.2 JCJ/Civitas variants

Several attempts have been made to improve various aspects of JCJ/Civitas. To solve the problem of quadratic complexity of JCJ/Civitas, Smith [Smi05] and Weber et al. [WAB07] proposed systems to linearise the complexity. However, it was later shown that those systems do not satisfy coercion-resistance [AFT07, AFT10]. More systems were purposed after that to successfully achieve coercion-resistance in linear-time [AFT07, AFT10, AT13, SHKS11, SKHS11, SHD10]. Clark and Hengartner proposed a system to simplify aspects related to usability and coercion resistance [CH11]. Our work proposed in Chapter 4 simplifies aspect related to verifiability, and Chapter 5 improves usability by weakening coercion-resistance [BGR11, GRBR13].

2.3.2 Voting on an untrusted platform

As we argued in the previous sections, verifiability and privacy are two fundamental properties of electronic voting protocols. What is more, these properties need to be ensured even in the presence of an untrusted platform, that may tamper with or coerce the voter's choice at any moment between voting and vote counting. Therefore, an ongoing

thread of research over the last few years has been in minimizing or removing the trust assumption that the voters need to make about their voting platform. In this section we briefly describe some of the system that completely or partially deal with the untrusted platform problem. Our work in Chapter 3 also propose solutions to deal with the untrusted platform problem.

SureVote [Cha01] was the first system to propose solutions for an untrusted platform. It avoids the assumption of a trusted platform, by bypassing to use the voter's computer except as an untrusted communication channel. In SureVote, voters receive a *code sheet* (shown in Figure 2.1) through a trusted channel, such as physical mail. The voter cast her vote by logging into the untrusted computer, and entering the Ballot ID and the Vote code of the preferred candidate. The server verifies the Ballot ID and based on the submitted vode code sends back the acknowledgement code. As the computer is used only to send and receive codes from the sheet, it does not learn the link between the candidate and the submitted code. Due to the insufficient knowledge about the codes on the code sheet, the untrusted platform can't change the codes. SureVote ensures privacy with respect to the untrusted computer, but voters cannot verify the construction of code sheets. Moreover, by using the code sheet and the acknowledgement code received back from the server, voters can prove how they voted in the elections.

Candidate	**Vote Code**	**Ack Code**
Alice	3857	8372
Bob	4753	7730
Charlie	9109	2515
Dave	7635	1456

Ballot ID: JS5Ah6VEV7Y2I7

Figure 2.1: An example code sheet received by a voter in SureVote.

Pretty Good Democracy (PGD) [RT09] improves SureVote by using an end-to-end backend to ensure stronger integrity properties. In PGD, the acknowledgement mechanism is distributed among several trusted tellers at the server end, thereby ensuring greater

guarantees that if the user receives the correct acknowledgement code then the correct vote is logged. The voters in PGD submit an encryption of their preferred candidate's vote code. The tellers perform plaintext equivalence tests (pet) of the encrypted code with their own data to find the acknowledgement code. In PGD, a single acknowledgement code is located to each code sheet as shown in Figure 2.2. This makes PGD receipt-free, in the sense that the voter cannot prove to a coercer after the event how she cast her vote.

Candidate	Vote Code
Alice	3857
Bob	4753
Charlie	9109
Dave	7635

Ack Code: 9876
Ballot ID: DS5Ah6VEC7Y2I3

Figure 2.2: An example PGD code sheet.

JCJ/Civitas [CCM08, JCJ05] discussed in detail in Section 2.3.1.1, provides coercion-resistance but requires voters to trust their computers for both integrity and privacy. Neumann et al. [NFVK13, NV12] extend Civitas with a smart card to manage voter credentials [CCM08]; that scheme does not protect the voter's privacy from an untrusted platform. For integrity, trust has been moved from the untrusted platform to the smart card and cut-and-choose technique is used to assure integrity of ballot construction.

Haenni and Koenig [HK13] propose a trusted device for voting over the internet on an untrusted platform. In their system, the platform generates encryptions in the form of barcodes, the voter makes her selection on the dedicated trusted hardware device by scanning chosen candidate's barcode. The voter's chosen vote is later transferred to a trusted computer using a smart card or a USB connector. The device requires a camera, matrix barcode reader, and uplink to a computer. Moreover, the hardware device learns voter's vote.

Remotegrity [ZCC+13] extends the Scantegrity II [CCC+09] paper-ballot voting system to use for remote voting. In Remotegrity, a voter receives a paper ballot and an *authorization card* by mail. Part of the ballot is printed with invisible ink, and parts of the authorization card are covered with scratch-off coating. A voter marks the ballot with a special pen, revealing a previously invisible code for a candidate. She then scratches off a field on the card, revealing a previously hidden authentication code. As part of the voting protocol, she submits both the candidate and authentication code to a bulletin board. Remotegrity assumes a secure means of distributing the ballots and authorization cards.

Helios [Adi08, dMPQ09] uses a *cast-xor-audit* [Ben06, Ben07] technique (also know as cut and choose) to assure integrity of ballot construction. A voter can either cast her ballot or audit it. Auditing challenges the computer to prove it acted honestly in encrypting her candidate choice. But to verify that proof, the voter must seek out a trustworthy computer. The voter's computer automatically learns how the voter voted, thus violating privacy.

Helios is designed for low-coercion elections. It makes a few efforts to resist potential coercion, for example by keeping secret from voters the randoms in their ballots, but these efforts are easily defeated. On the positive side, Helios 2.0 enjoys individual and universal verifiability (but not eligibility verifiability). The most interesting feature of Helios is its high usability, which has been demonstrated by running large elections without failure.

2.3.3 Other related systems

We discussed remote voting systems in the previous section. In this section we describe Prêt à Voter, a polling station voting system that is designed to provide end-to-end verifiability and transparency while ensuring secrecy of the ballots.

Prêt à Voter [CRS05, RS06, RBH+09] is a paper ballot based polling station voting

system which provides end-to-end verifiability to the voters and auditors. In this system, the ballots, shown in Fig 2.3, are generated by the trusted authority before the start of the elections. The ballot has two columns, left hand column contain the candidate names and the right hand column for voter to mark her selection. The candidate names on the left column are in a randomized order, the offset of this order is encrypted and written on the bottom of the right hand column.

Candidate	**Tick box**
Dave	
Alice	
Bob	**X**
Charlie	
	ab34kq9c

Figure 2.3: An example ballot in Prêt à Voter.

To cast a vote, the voter ticks the box next to the candidate of her choice, and then tears off the two columns and shreds the left hand column, which contains the list of candidates. The right hand column is then scanned by an optical scanner. The voter keeps this right hand column as a receipt and later compares it with the bulletin board where the scanned copy should appear. The receipt assures the voter that her vote is submitted as cast but it doesn't reveal to anyone else how she voted. This scanned right hand column is mixed by using mix nets and decrypted by tellers in a distributed manner.

Prêt à Voter also employs cast-xor-audit technique to let voters and observers verify the correctness of generated ballots. The voters can ask to audit a ballot at the polling station (which will annul their ballot) and then they can start the process to cast their vote again.

CHAPTER 3

Du-Vote: Internet voting with untrusted computers

3.1 Introduction

Most of the existing internet voting protocols typically assume that the computer used by the voter to submit her vote is trusted for privacy [Adi08, HLW11, Gjø11], or for both privacy and verifiability [JCJ05, CCM08]. This assumption might seem necessary, because humans cannot compute the encryptions and other cryptographic values necessary to run such protocols. But the assumption is problematic, because a voter's computer might well be controlled by an attacker [ED10].

In this chapter, we introduce a new protocol Du-Vote (Devices that are Untrusted used to Vote), that eliminates the need for voters to trust their computers. As depicted in Figure 3.1, we propose the use of a simple hardware token (referred in this chapter as H) such that a voter V interacts with her computing platform P and the hardware token H to cast a vote on server S. The encryption of candidates take place on the computing platform, and the voter's choice of encrypted candidate is made with the hardware token, such that neither the platform nor the hardware token learns the voter's plaintext choice. The server verifies that the computing platform is behaving honestly and anyone (e.g. observers and voters) can verify the computations done by the server, to check its honesty.

Moreover, Du-Vote requires very little functionality from the hardware token. It needs only

- to accept short inputs (e.g., twenty decimal digits) and produce very short outputs (e.g., four decimal digits),
- to store a secret value, similar to a cryptographic key, that can be pre-installed before delivery to the voter, and
- to compute modular exponentiations and multiplications.

The hardware token H does not need any connection to another computer, so there are no requirements for a general-purpose operating system, drivers, etc. Indeed, Du-Vote requires H to be unable to communicate with anyone other than the voter. The only means for

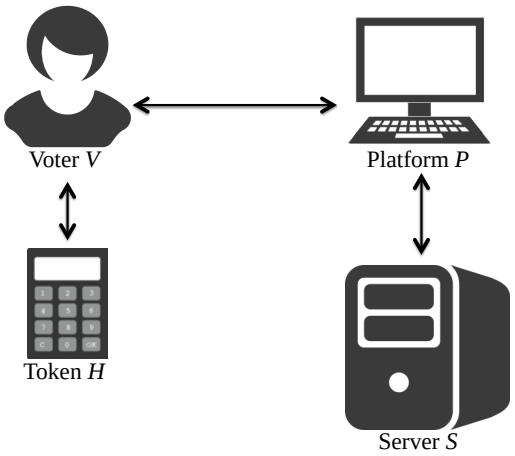

Figure 3.1: Du-Vote architecture.

H to communicate with the outside world should be by using a built-in keyboard and screen. H can even be *software closed*, such that its software cannot be modified. These requirements can be satisfied by hardware tokens similar to those shown in Figure 3.2, which are used by e.g. banks for two-factor authentication. Such tokens have a decimal keypad, an LCD screen that can display a few digits, and a *command* button that causes the device to compute an output.

Because of its simple design, the trustworthiness of H could be established through verification and/or testing. In fact, given a trustworthy H, Du-Vote does not need to make any assumptions about the computing platform P, either for privacy or for integrity. Our security analysis (in Section 3.4) shows that, when H is trustworthy, Du-Vote guarantees verifiability of the election outcome and of individual votes even if both P and S are controlled by the adversary. Du-Vote guarantees privacy of votes if at least one of P and S is not controlled by the adversary. So one contribution of Du-Vote is the relocation of trust from a large computational device (the voter's general-purpose computer) into a small computational device (a token). Although other systems (including SureVote [Cha01], Pretty Good Democracy [RT09], and Helios [Adi08]—all discussed in Section 3.6) have similarly worked to relocate trust, Du-Vote is the first to ensure both privacy and verifiability with a trusted computing base that consists only of a hardware token like that used in real-world banking applications.

Figure 3.2: Hardware tokens used in banking. These devices store secret information into a tamper-resistant hardware. One on the left, has device with screen and keyboard; on the right, a device with screen and just one key.

Du-Vote offers substantial protection of integrity and privacy even if the manufacture of the hardware token H is controlled by the adversary. Surprisingly, verifiability of the vote is assured with high probability even if all the machines involved (H, P, and S) for a given voter V are controlled by the adversary (Section 3.4). Privacy is guaranteed if either P, or H and S, are not controlled by the adversary. Another contribution of Du-Vote is, therefore, a voting system that achieves verifiability and privacy without requiring trust in the computers used by voters. The assumptions we make to achieve privacy are justifiable because voters who are technically competent can protect their privacy by making sure that their own computer (P) is free from any election related malware. The voters who don't trust their computers can rely on trustworthiness of devices (H and S) provided by the election authorities.

The verifiability in Du-Vote involves a novel non-interactive zero-knowledge proof that enables the server S to prove that it correctly selects and re-encrypts the ciphertext chosen by the voter, without revealing it.

Du-Vote is the *front end* of a voting system: it is designed to collect votes and post them on a bulletin board. The *back end* used to tally the bulletin board is modular; it could be instantiated with verifiable reencryption mixnets [Cha03, JJR02] or verifiable homomorphic tallying [CGS97, dMPQ09].

Our main contributions: 1) We introduce Du-Vote, a new voting system that addresses

the problem of voting using untrusted voting machines and untrusted servers. Du-Vote is the first voting system to ensure both privacy and verifiability with a TCB that consists only of a hardware token like that used in real-world banking applications. 2) We show that even if this TCB is in fact untrusted (Section 3.4.1.2), it is practically impossible for it to change some voters' votes without detection.

We proceed as follows. Section 3.2 introduces Du-Vote from the perspective of a voter. Section 3.3 presents the design of the Du-Vote voting scheme. Section 3.4 analyses the security of Du-Vote, including what happens when various components turn out to be malicious. Section 3.5 discusses different aspects of Du-Vote and possible improvements. Section 3.6 compares Du-Vote with related work, and section 3.7 concludes the chapter.

3.2 Voter experience

We begin by describing Du-Vote from the voter's point of view. No cryptography is involved in this description, in part to illustrate that the voter experience is straightforward. Section 3.3 details all the cryptographic protocols.

Registration: Each voter registers to vote with the authority running the election. This registration may occur online. The voter establishes a username and password for server S to cast her vote. The authority issues a hardware token to all voters, either in person or by postal mail (perhaps in tamper-evident envelopes to reduce the chances of disruption of elections by a delivery service).

Voting: Using her computing platform P, voter authenticates to server S using her voter ID and password.[1]. The platform displays a *code page* to voter, as shown in Figure 3.3[2]

The instructions in Figure 3.4 are conveyed to the voter by the computing platform.

[1]Neither privacy not integrity of the election depend on the security of the password. It is present only to prevent data disruption that could anyway be detected in the verification stage.

[2]Du-Vote's code pages appear superficially similar to Chaum's code sheets [Cha01]. But Du-Vote constructs and uses code pages in a different way, as we describe in Section 3.3.

Candidate	Column A	Column B
Alice	3837	7970
Bob	6877	2230
Charlie	5525	1415
Dave	9144	2356
Enter your vote code here		

Ballot ID: JSAhVEVYIFRTLXByb2dyYW

Figure 3.3: An example code page as displayed by P.

These instructions are also widely publicised—for example, in newspapers, TV channels, and online media—so that they are common knowledge among voters. Following them, the voter enters five codes into her hardware token, obtains a new *vote code* from the hardware token, and enters that vote code into the computing platform. Then the voter records her ballot ID and vote code.

Please follow these instructions to cast your vote:

- Check that your computer has displayed the candidates in alphabetical order and has displayed two codes (in column A and B) for each candidate.
- Flip a coin.
- If the coin landed **heads**:
  - Enter all the codes, from top to bottom, from column A into your token.
  - Find the code for the candidate for whom you wish to vote in column B. Enter that code into your token.
- If the coin landed **tails**:
  - Enter all the codes, from top to bottom, from column B into your token.
  - Find the code for the candidate for whom you wish to vote in column A. Enter that code into your token.
- You should now have entered a total of 20 digits into your token. Press the command button on your token. Your token will display a new, four-digit vote code. Enter that vote code into your computer.
- Record your ballot id and vote code to later double-check that your vote was received.

Figure 3.4: Voting instructions, assuming $n = \kappa = 4$. The voter is in possession of a hardware token similar to that of Figure 3.2 (left), and a computer P displaying a code page similar to Figure 3.3 .

Auditing: A log of all the votes received by the server is made available for review by

all voters. To double check that her vote was correctly received and recorded, V confirms that her ballot ID and vote code are in that log.

The cryptography behind the curtain: Although voters don't need to know this, the codes on the code page are *truncated* probabilistic encryptions of candidate names—that is, the last four digits of a decimal representation of a ciphertext. The computing platform P computes those ciphertexts. That leads to three problems, all solved with the help of the hardware token:

- The computing platform P might try to cheat in constructing the ciphertexts. So voters are required to enter an entire, randomly chosen column to their hardware tokens. The vote code output by the hardware token is based on that column. The server uses the vote code to verify that the column was encrypted correctly.

- The computing platform might try to change the order of codes displayed to the voter. So the vote code output by the hardware token is also based on the order in which voter has seen codes in the audit column. This later informs server in what order a voter saw the codes.

- The computing platform might try to learn the voter's plaintext vote. Clearly, since P knows the plaintexts, voter cannot reveal its chosen candidate code directly to P. So voter makes her selection on her hardware token. The vote code output by the hardware token is based on that selection. The server uses the vote code to recover the voter's encrypted candidate.

How the server uses the vote code to accomplish these tasks, and all the other cryptographic details, we explain, next.

3.3 Voting scheme

In addition to the principals already introduced (hardware token H, voter computing platform P, and server S), Du-Vote uses a set T of *decryption tellers*, who are principals entrusted with shares of the decryption key under which votes are encrypted. Du-Vote also uses a bulletin board BB and a cryptographic hash function $\mathsf{hash}(\cdot)$.

3.3.1 Setup

Du-Vote employs distributed ElGamal encryption and exponential ElGamal encryption (as explained in Chapter 2 Section 2.2). Each decryption teller has a private key which is a number $z_j \in \mathbb{Z}_q^*$ and a corresponding share of the public key y_j, where $y_j = g^{z_j}$. The product $\prod_j y_j$ of all the public keys is used as a public key, denoted y. Decryption of a ciphertext encrypted under y requires participation of all the tellers, each using their own private key.

A variant of exponential ElGamal encryption is also used in Du-Vote. In this variant, a group generator g is raised to the power of randomness r, and another group generator h is raised to the power of the plaintext message. Recall that r is a fresh random number in $\mathbb{Z}_q^*$.

3.3.2 Registration

The electoral roll of authorized voters is determined by the election authority in advance. Each authorized voter V may register to establish a voter ID and password for server S. Du-Vote does not require any particular protocol for this registration.

As part of registration, the election authority issues to each voter a hardware token. Each hardware token is configured with its own, unique value K, where $K = y^k$ for a randomly chosen k.[1]At the time H is issued to V, the authority causes server S to record

[1]Our notation is meant to suggest keys, and indeed k will be used in Section 3.3.6 to compute a value that resembles an El Gamal encryption.

an association between k and V. Note that S can, therefore, derive the K stored on V's token. We assume that H is issued over a private channel (perhaps in person or through postal mail in a tamper-evident envelope).

Given a sequence of decimal digits as input, H interprets that input as an integer d. When the command button is pushed, H outputs $(Kh^d \ mod \ p)^*$, where h is another group generator used in exponential ElGamal encryption and * denotes a *truncation* operator that produces short representations of large values, as follows. Let κ be a security parameter of Du-Vote. Given a large value L, let L^* denote the final κ digits of some canonical representation of L as a decimal integer. Therefore $(Kh^d)^*$ denotes the final κ digits of the decimal representation of Kh^d (we skip $mod \ p$ as it is clear from the context). Here are two other examples of how to form L^*:

- If L is a bit string, then convert that bit string to an integer and yield its final κ decimal digits.
- If L is an ElGamal ciphertext (g^r, my^r), first represent (g^r, my^r) as a bit string $g^r||my^r$, where $||$ denotes concatenation, then proceed as in the previous example.

Denote the output $(Kh^d)^*$ of the hardware token on input d as $H(d)$.

Typically, we assume $\kappa = 4$. More digits would require more typing into the hardware token, affecting the usability of Du-Vote. Fewer digits would affect the security of Du-Vote by reducing resistance to randomization attacks (discussed in Section 3.4.1.2, Remark 4) in which hardware token could causes voter's vote to change to an unpredictable candidate.

3.3.3 Opening of election

In advance of an election, the set Can of n candidate names $\{a_1, \ldots, a_n\}$ is published on BB. An algorithm for generating a publicly verifiable but unpredictable *election nonce* I is also declared and published. This election nonce could be generated by using stock market data [CCC+10, CEA07, CH10]. The election nonce must be generated after the hardware tokens have been manufactured. It is critical for integrity that hardware tokens

Candidate	Column A	Column B
a_1	A_1^*	B_1^*
a_2	A_2^*	B_2^*
a_3	A_3^*	B_3^*
a_4	A_4^*	B_4^*
Enter your vote code here		

Ballot ID: $\mathsf{hash}(A, B)$

Figure 3.5: The cryptographic construction of a code page. Each entry A_i^* denotes the truncation of an encryption $A_i = \mathsf{enc\text{-}exp}(a_i)$ of candidate a_i. Likewise, each B_i^* is also the truncation of an encryption $B_i = \mathsf{enc\text{-}exp}(a_i)$ of a_i. In the ballot ID, value A denotes the lexicographically sorted sequence of ciphertexts $A_1, \ldots, A_4$ in column A, and likewise for B in column B.

are unable to learn the election nonce (that is why they have no input interface except the short value keypad).

3.3.4 Voting: preparation of the vote by platform P

To vote, voter first authenticates to server through her computing platform using her voter ID and her password. The platform retrieves candidate names *Can* and election nonce from the bulletin board.

Second, P prepares a code page for the voter. The essential idea in preparing that code page is that P creates two different encryptions of each candidate name. The two encryptions will be used later to verify that P computed the candidate's encryptions honestly. The construction of a code page with four candidates is shown in Figure 3.5. Each candidate's name is encrypted once in column A, and again in column B. To defend against an attack (cf. Section 3.4.1.2, Remark 5) in which a malicious H and P conspire to violate integrity, those encryptions must be chosen by P in a specific way, detailed below. This method is designed to prevent a dishonest P from having a covert channel with which to communicate to a dishonest H.

- P computes a set $\{c_1, \ldots, c_{2n}\}$ of distinct decimal codes determined by I and voter ID, each code having κ digits (recall that n is the number of candidates). To achieve

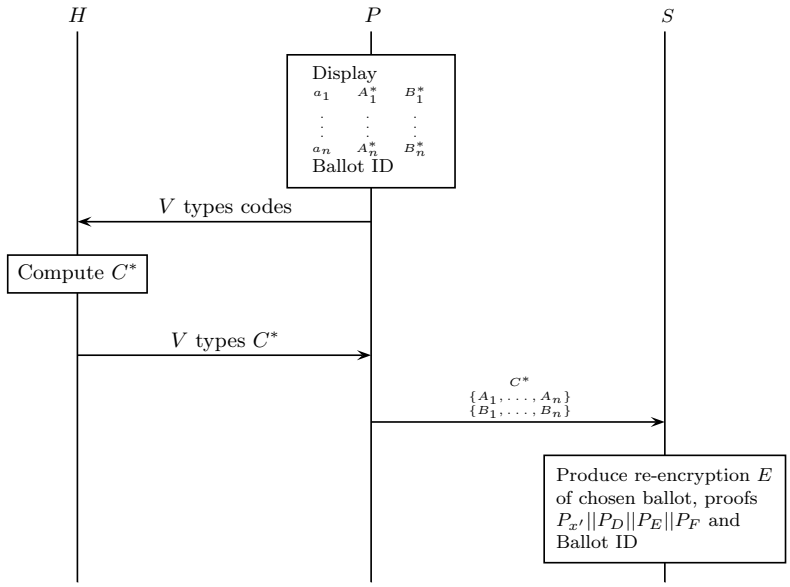

Figure 3.6: Du-Vote voting protocol. Platform P displays the candidates and the codes. Token H computes vote code C^* from the codes entered by the voter. The server S later produces re-encryption E of the chosen candidate's encryption and NIZKPs to prove its honesty.

this, P generates a bit stream seeded by $\mathsf{hash}(I, \text{voter ID})$ and repeatedly consumes an appropriate number of bits from it to form each of the codes in turn. If any code would be identical to a previous one, P discards it and produces a new one, until it has the required $2n$ distinct codes.

- P chooses a random α such that $1 \leq \alpha \leq n$ and cyclically assigns the subset $\{c_1, \ldots, c_n\}$ of codes to candidates: c_α to a_1, $c_{\alpha+1}$ to a_2, ..., wrapping around and assigning $c_{\alpha-1}$ to a_n. Let $\mathsf{code}_A(i)$ denote the code assigned to a_i according to α.

- Likewise, P chooses a random β such that $n+1 \leq \beta \leq 2n$ and assigns the subset $\{c_{n+1}, \ldots, c_{2n}\}$ of codes to candidates: c_β to a_1, $c_{\beta+1}$ to a_2, ...; Let $\mathsf{code}_B(i)$ denote the code assigned to a_i according to β.

- For each candidate a_i, platform P chooses a random r and computes $\mathsf{enc\text{-}exp}(a_i; y; r+j)$ for each $j \in \{0, 1, 2, \ldots\}$ until the ciphertext $A_i = \mathsf{enc\text{-}exp}(a_i; y; r+j)$, satisfies

$A_i^* = \mathsf{code}_A(i)$. [1]

- Likewise, P finds ciphertexts B_i such that $B_i^* = \mathsf{code}_B(i)$.
- P generates a code page as depicted in Figure 3.5. This code page also contains a ballot ID where P commits to the ciphertexts. The ballot ID is defined as $\mathsf{hash}(A, B)$ where A denotes the lexicographically sorted sequence of ciphertexts $A_1, \ldots, A_n$ in column A, and likewise for B in column B.

Third, P invites V to vote with the code page. To vote, V follows similar instructions as in Figure 3.4. For example, if V's coin flip yields tails and V wants to vote for a candidate whose ciphertext is x, then V enters $B_1^*||\ldots||B_n^*x^*$ into H, where $x^* \in \{A_1^*, \ldots, A_n^*\}$, and presses the command button. Let d be the decimal representation of what the voter enters. In response to the command button, H calculates $C = Kh^d \ mod \ p$, and outputs vote code C^*, which is the final four digits of C. The voter enters C^* into P. Finally, P sends C^*, along with $\{A_1^*, \ldots, A_n^*\}$ and $\{B_1^*, \ldots, B_n^*\}$, to S.

3.3.5 Voting: processing of the vote by server S

First, to verify P's honesty, S checks that P computed the codes in A and B correctly. To do so, S computes bit string $\mathsf{hash}(I, \text{voter ID})$ and—just as P should have—produces a set $\{c_1, \ldots, c_{2n}\}$ of codes from it. Then S checks that A contains the first half of that set, and B the second half (both in some cyclic order). S posts C^*, A, and B on BB. S also generates ballot ID (just as P should have) and posts that on BB as well. This ballot ID forces P to commit to the ciphertexts; it cannot later choose different ciphertexts (with different plaintexts) that match the codes.

Second, S does a brute-force search to determine what code V entered as a candidate choice. The input space for this search is the set of inputs V could have entered into H, given that V's code page was constructed from A and B. Anyone who knows I and voter

[1] This is efficient, because if $\mathsf{enc\text{-}exp}(a_i; y; r + j) = (\alpha, \beta)$ then $\mathsf{enc\text{-}exp}(a_i; y; r + j + 1)$ can be computed as $(\alpha g, \beta y)$. See section 3.5. Moreover, it is secure; we prove the security of this variant of Elgamal in Appendix A.2.

ID can compute those inputs. The size of the input space is $2n^2$: the full column entered by the voter is 1 of 2 columns; the cyclic permutation of that column starts at 1 of n candidates; and the candidate chosen by the voter is also 1 of n.

Computing the output of H, however, requires knowledge of value K stored in H. Recall (from Section 3.3.2) that S can derive that value. S, therefore, simulates the voter's token to find the ciphertext x that produces C^* (recall that ciphertext x is an ElGamal pair so x can be written as $x_1||x_2$ or (x_1, x_2). Indeed in next section we will use these.) The final κ digits of x is the code for the voter's candidate. S recovers the full ciphertext for that code from A or B. S also determines which column was fully entered by the voter; call that the *audit column*.

Third, S requests from P the random coins used to encrypt each candidate in the audit column. S posts those on BB, then checks whether the encryptions for each candidate in that column were computed correctly by P (Note that S knows the order in which V has seen the audited codes.). If that check fails (or if the brute force search previously failed), then one or more of P, H, and V is faulty or dishonest. In that case, S refuses to do any further processing of V's vote.

Fourth, S reencrypts x to a new ciphertext E and posts E on BB as V's vote. The reencryption is necessary, because P (having generated the encryptions) knows the plaintext corresponding to x. So if S directly posted x to BB, then P would easily be able to violate V's privacy.

Fifth, S proves in NIZK that it followed all the above protocol honestly by posting a proof on BB.

Finally, voter V finishes by recording the vote code and the ballot ID, and by logging out of S. The voter must check that this recorded information is present on BB, or convey this information to a third party who does the check on behalf of V. Note that conveying it does not harm V's privacy.

3.3.6 Voting: S proves its honesty

As mentioned above, S must create universally-verifiable proofs that it correctly selected and reencrypted the ciphertext x chosen by the voter. The proofs will be posted on BB and anyone will be able to check it. Here we briefly discuss those proofs and we provide their technical details in the Appendix A.1. The proof employs standard cryptographic techniques for constructing a non-interactive zero-knowledge proof, based on ring signatures [AOS02, CLMS08] and the Fiat-Shamir heuristic [FS86].

To prove its honesty, S provides:

- **Proof of selection:** The server S produce two zero-knowledge proofs $P_{x'}$ and P_D, to show that it correctly computed the voter's chosen ciphertext $x = (x_1, x_2)$ from the inputs C^* and x^* that S receives from P. This proof requires S to compute a pair $D = (D_1, D_2) = (g^d, y^d h^{x_2})$, for some value d. To prove that D is of this form, among other steps, S computes $C_{x'} = y^{r'} h^{10^\kappa x'}$, and publishes it, where r' is chosen randomly and x' is driven from x_2 i.e. $x_2 = x'||x^*$.

- **Proof of re-encryption of a ciphertext:** S computes a re-encryption $E = (E_1, E_2) = (g^e x_1, y^e x_2)$ of x (where e is chosen at random) so that the computing platform can't see how the voter voted. S provides a proof P_E to show that it correctly constructed the re-encryption.

- **Proof that the re-encryption is of the selected ciphertext:** The proof P_E described in the previous step shows that E is a re-encryption of one of the values in $\{A_1, \ldots, A_n\}$, but not that it is the one chosen by the voter. Therefore, to prove that the (x_1, x_2) used in the computation of E matches the code chosen by the voter, S has to show that the x_2 in E is the same as the x_2 in D. To demonstrate this, S computes $F = (F_1, F_2) = (E_1 D_1, E_2 D_2) = (g^f x_1, y^f x_2 h^{x_2})$ where $f = d + e$, and proves that F indeed has this form. S produces another proof P_F to show that F

has the correct form.

By putting $P_{x'}, P_D, P_E$ and P_F together, S actually proves that E is a re-encryption of $A_i = (A_{i,1}, A_{i,2})$ where $A_{i,2} = x_2$ and x_2^* is involved in C^*. The proofs $P_{x'}, P_D, P_E$ and P_F are bound together as a single Schnorr signature proof P_{total}. This proof P_{total} is also posted on the bulletin board BB. A more detailed description of these proofs is given in the Appendix A.1.

These constructions are novel, but they make use of the standard signature-based proof of knowledge (SPK) technique. As Fiat-Shamir [FS86] heuristic is used, we are assured that the proof is sound and complete, and has the computational zero-knowledge property.

3.3.7 Tabulation

The tellers T receive from S the ciphertext E for each voter. The voter's names or identities can be publicly associated with their encrypted vote. This allows any voter to check that their E is included.

The set of encrypted votes is tabulated by a back end. Du-Vote does not require any particular back end. One of the standard method from the literature that could be used based on the current structure of ballots is: a reencryption mixnet, followed by decryption [JCJ05, CCM08].

Another possibility is to use homomorphic combination, followed by decryption of the combinations [dMPQ09]. Although Du-Vote ballots are exponential ElGamal ciphertext (multiplication of the ciphertexts would correspond to addition of the plaintext), however, the current structure of the ballots would require modification to use homomorphic combination.

3.3.8 Verification

Voters and observers take the following steps to verify the integrity of the material on the bulletin board BB. As usual, we envisage that a variety of programs produced by different agencies would be available for performing the verification.

During vote processing, S publishes the following information for each vote:

- voter ID
- ciphertexts $\{A_1, \ldots, A_n\}$ from column A, and $\{B_1, \ldots, B_n\}$ from column B
- the identity (A or B) of the audited column
- the random values r used for encryption in the audited column
- ballot ID
- outputs $C, C_{x'}, D_1, E$, and P_{total} described in Section 3.3.6, where P_{total} includes $P_{x'}, P_D, P_E$ and P_F.

Any voter can ensure that her vote is present, by checking that BB contains an entry with her ballot ID, the value C^* that she obtained from the code page displayed by P during voting, and by comparing the audit column shown by P.

Any observer (including any voter) can verify the integrity of the other information, by performing the following BB **verification steps**:

1. Identify the randoms used in the audited column, and check the audited ciphertexts by reconstructing them. Without loss of generality, assume column B is audited and column A is used to cast vote
2. Verify the ballot ID by computing $\mathsf{hash}(A, B)$, where value A denotes the lexicographically sorted sequence of ciphertexts $A_1, \ldots, A_n$ in column A, and likewise B in column B

3. Identify C and check that the last κ digits of C are C^*

4. Compute $C_{x^*} = C/h^{\{10^\kappa(B_1^*||...||B_n^*)\}}$

5. Verify proof $P_{x'}$

6. Identify $C_{x'}$ and compute $D_2 = C_{x^*}C_{x'}$ and $D = (D_1, D_2)$, and verify P_D

7. Identify E and verify P_E

8. Compute F from D and E and verify P_F

9. Check E has been sent to T

These steps involve voters checking the bulletin board and also verifying the zero-knowledge proofs. We note that not all voters will be able to understand the security guarantees provided by these checks and they might need to rely on a trusted third party. It will be useful to make such proofs more intuitive. Indeed in Chapter 4 we will discuss techniques to make such verification more intuitive.

3.4 Verifiability and privacy analysis

Full security analysis of Du-Vote (similar to [DKR09] or [KRS10]) is beyond the scope of this chapter. The main contribution here is to introduce the novel ideas that allow secure voting to take place even if the backend server and the devices the voter interacts with are all malicious. In this section, we present some results that add detail to this claim.

Du-Vote's primary objective is to ensure integrity and verifiability of the vote, even if the voter's platform is untrusted. Section 3.4.1 is devoted to showing that any manipulation of a vote by the untrusted platform P or the untrusted server S will be detected with high probability, even if S and P are corrupt and controlled by the same attacker. Additionally in Section 3.4.1.2 , we consider the possibility that the hardware token H is corrupt too. There, we show that even if all three of S, P and H are corrupt and controlled by the same

attacker, we still obtain detection of large scale manipulation under certain reasonable assumptions.

A secondary objective of Du-Vote is to ensure privacy of votes. In Section 3.4.2.1 and 3.4.2.2, we assume H is trustworthy, and we argue that privacy of a vote holds even if one of S or P is corrupt and controlled by an attacker, provided the other one is honest. In Section 3.4.2.3, we consider the case that H is not trustworthy. We show that although a privacy attack is possible, it cannot be conducted on a large scale without being detected.

We take the view that integrity and verifiability of the election is more important than privacy, and that is why our assumptions to guarantee integrity are weaker than those required to guarantee privacy. Our view about the importance of integrity over privacy is justified by considering what happens if one of them fails, while the other is preserved. An election with privacy but no integrity is useless: the outcome is meaningless. But an election with integrity and no privacy, while not ideal, is still useful (indeed, show-of-hands elections in committee rooms are used all the time).

3.4.1 Verifiability

We divide our analysis into two parts. In the first part, we assume the hardware tokens H are trustworthy. In the second one, we consider the case that their manufacture is controlled by an adversary.

3.4.1.1 Verifiability analysis assuming honest H and corrupt P, S

We show that any manipulation of a vote by the untrusted platform P or the untrusted server S will be detected with high probability, even if S and P are corrupt and controlled by the same attacker.

We make the following two assumptions:

- First, we assume that the tokens H are honest.
- Second, we suppose that the *BB verification steps* detailed in Section 3.3.8 have

been carried out by some voters or observers, and have passed.

In our first remark we use the well-known *cut-and-choose* argument:

Remark 1. *Suppose a voter V votes and checks that her ballot ID and vote code is present on BB. With probability at least 50%, P correctly computes the ciphertexts $\{A_1, \ldots, A_n\}$ and $\{B_1, \ldots, B_n\}$ for V.*

Proof Sketch: Suppose P incorrectly computes one or more of $\{B_1, \ldots, B_n\}$. With 50% probability voter's coin toss is tails and she enters $(B_1^*|| \ldots || B_n^*|| x^*)$ into H and sends C^* to S. The server S identifies the audited column and requests the randoms for the ciphertexts corresponding to the audited column $\{B_1, \ldots, B_n\}$, and publishes the randoms, ciphertexts, voter ID and ballot ID on BB. An observer is now able to check that the received ciphertext corresponds to the randoms and plaintexts in the correct order, and that the ciphertexts are the original ones computed by P corresponding to the commitment in the ballot ID.

As a reminder, n is the number of candidates and κ is the length of codes.

Remark 2. *Suppose a voter V votes and checks that her ballot ID and vote code is present on BB. If P correctly computes the ciphertext x corresponding to the code x^* chosen by V, then with probability $\frac{(10^\kappa - 1)^{2n^2-1}}{10^{\kappa(2n^2-1)}}$, we have that:*

(a) S correctly computed the chosen code x^*,

(b) S correctly computed the chosen ciphertext x, and

(c) S correctly computed the re-encryption of ciphertext E.

For example, if $\kappa = 4$ and $n = 4$, this probability is 0.9969. Thus, with high probability, a correct encryption of the voter's chosen candidate is submitted to the tellers.

Proof Sketch: Suppose the ciphertexts $\{A_1, \ldots, A_n\}$ and $\{B_1, \ldots, B_n\}$ correctly encrypt the candidates $\{a_1, \ldots, a_n\}$. We prove each of the statements (a), (b), (c) in turn.

(a) Suppose without loss of generality that the voter chose to audit $\{B_1, \ldots, B_n\}$ and vote with $\{A_1, \ldots, A_n\}$; thus she entered $B_1^*|| \ldots ||B_n^*||x^*$ into H, generating C^*. By the assumption that the BB verification steps pass, the server receives C^* correctly. Next, the server computes possible inputs to H that could have generated C^*. Since the vote codes are pseudorandom, the probability that the voter has chosen inputs to H which result in a unique C^* can be calculated as $\frac{10^\kappa-1}{10^\kappa} \frac{10^\kappa-1}{10^\kappa} \ldots \frac{10^\kappa-1}{10^\kappa}$ (there are $2n^2-1$ factors), which equals $\frac{(10^\kappa-1)^{2n^2-1}}{10^{\kappa(2n^2-1)}}$. In this case, since there no collisions on C^*, S has no choice but to correctly identify x^*.

(b) Any observer that checks proof P_{total} is assured of the computation of x from x^*.

(c) Any observer that checks proof P_{total} is assured of the computation of re-encryption E from x.

Putting these remarks together, we obtain Remark 3:

Putting these remarks together, we obtain:

Remark 3. *Suppose a voter participates in the election, and later verifies that the bulletin board contains her ballot ID and vote code. Then the probability that her vote has not been altered and will be decrypted as cast is* $0.5 \times \frac{(10^\kappa-1)^{2n^2-1}}{10^{\kappa(2n^2-1)}}$.

In the case $\kappa = 4$ and $n = 4$, this probability is 0.4985. Since the attacker's chance of manipulating a vote is independent of all the others, the probability of him undetectably altering more than a few votes quickly becomes negligible.

3.4.1.2 Verifiability analysis assuming corrupt H, P, S

Even if all three of S, P and H are corrupt and controlled by the same attacker, we still obtain detection of large-scale vote manipulation under some reasonable assumptions. We explain these assumptions below:

- The first assumption is that H has no wireless or wired interfaces; the only way for it to communicate with the outside world is via its keypad and screen. The attacker controls S and P in real time; he can communicate directly with them. But his ability to control H is limited to manufacture time. The attacker can choose the firmware and other data that is installed on H, but once H has been manufactured he can no longer communicate with it because of the lack of wired or wireless interfaces. The idea is that H is prevented from learning the election nonce I, and this prevents it from being useful to the attacker. H's keyboard does provide a very low-bandwith channel by which the attacker can try to communicate with it, but that channel is far too low-bandwith to communicate the high-entropy I and produce the correct ballot at the same time (Indeed election nonce could be made as much as high-entropy as required to maintain the security of the elections.).

- The second assumption is that, although the attacker can spread malware to voters' machines and thus control the P that they execute, it cannot spread malware to all the machines. Some PCs are well-looked after and some users are cautious enough to avoid downloading malware.

  Some of the Hs may be programmed correctly and others may be programmed maliciously, in a proportion chosen by the attacker. But we assume he cannot effectively match dishonest Ps with dishonest Hs, because the malware distribution channels are completely different from the hardware distribution channels. This means that some honest Ps will be paired with dishonest H's; and we assume those ones represent a fraction f of all the P's ($0 \leq f \leq 1$).

- As before, we also assume that the *BB verification steps* detailed in Section 3.3.8 have been carried out by some voters or observers, and have passed.

To obtain our result, we note that there are precisely two ways in which a dishonest H could contribute to an integrity attack:

1. H could produce a random output instead of the correct one.

2. H could produce an output calculated according to some attack strategy that has been built into it at manufacture time.

Intuitively, a random output will be very likely to be detected, because S will fail to find a value x^* that matches one of the available ciphertexts. We formalise this intuition in the following remark.

Remark 4. *If a dishonest H modifies votes randomly by replacing the code x^* for a voter's chosen candidate with x', it gets detected with probability $1 - \{(n-1)/(10^\kappa - (n+1))\}$ by S. If $n = 4$ and $\kappa = 4$, this probability is $1 - 3/9995 \approx 0.9996$.*

Proof Sketch: Suppose without loss of generality that the voter chose to audit $\{B_1, \ldots, B_n\}$ and vote using $\{A_1, \ldots, A_n\}$; thus she entered $(B_1^*||\ldots||B_n^*||x^*)$ into H, where $x^* \in \{A_1^*, \ldots, A_n^*\}$. The dishonest H chooses x'^* randomly instead of x^* and generates C'^*. To succeed in substituting codes, H has to guess x'^* s.t. $x'^* \in \{A_1^*, \ldots, A_n^*\} - \{x^*\}$. Device H already knows $n+1$ codes of κ digits that can't be repeated. So, the probability of getting a correct code is $(n-1)/(10^\kappa - (n+1))$. If $x'^* \notin \{A_1^*, \ldots, A_n^*\} - \{x^*\}$ then C'^* will not match as expected by S and the attack will be detected with probability $1 - \{(n-1)/(10^\kappa - (n+1))\}$. (Recall that S publishes the set of values $C, C_{x'}, D_1, E, P_{\text{total}}$, and by our trust assumption, they are verified by some voters and observers.)

Substitution attacks: Instead of relying on H to replace the code x^* with a random value (as done in previous remark), an attacker can have strategically placed some data onto H. Such data allows dishonest P to choose code sequences that covertly trigger malicious behaviour within H. The attacker can arrange it so that H produces an output corresponding to a choice which is valid but different from the one made by the voter. We call such attacks as *substitution attacks*. For instance, P and H can agree to assign a code, say c_1, to one of the candidates in one column and, say, c_2 to attacker's chosen

candidate in the other column. While voting, if code c_1 appears then H substitutes V's chosen candidate's code with the code c_2.

The risk for the attacker mounting substitution attacks arises when an honest P inadvertently triggers the malicious behaviour, resulting in H substituting the voter's choice for an invalid one. This will be manifested by some voters having their votes rejected. Considering the above example, if an honest P assigns code c_1 to one of the candidates in a column but code c_2 is not assigned to any candidate in the other column, then if H replaces V's code then the resulting vote will be rejected. We formalise this intuition in the following remark.

Remark 5. *Suppose a fraction f of the platforms P are honest. The server S and dishonest P's are controlled by the same attacker, who also manufactured the devices H. In order to modify on average N votes, the attacker will on average cause $R \times N$ voters to have their votes rejected, where*

$$R = \frac{f\ (1 - n/10^{\kappa})\ 10^{\kappa}}{((1-f)n + f)\ n}$$

For example, if $f = 0.5$ and $n = \kappa = 4$, then $R \approx 500$.

Proof Sketch: Suppose the attacker has configured dishonest H with data that allows P to trigger a substitution attack. Such data can be represented as a series of pairs of the form (z, Q) where z is the substitute code, and Q is a predicate on the input received by H which determines whether the substitution is triggered. Thus, if H receives input satisfying the predicate Q, then H should substitute z for the voter's chosen code and compute its response accordingly. The predicate Q can express a set of codes to use and/or a particular order in which to use them. We write codes(Q) to mean the set of codes in inputs that satisfy Q. We suppose the attacker has chosen ℓ lines, $(z_i, Q_i)_{1 \leq i \leq \ell}$. Without loss of generality, we assume that the sets codes(Q_i) are all disjoint (if they are not, it introduces ambiguity about how P should behave if the codes it should use match

several sets Q_i). We also suppose that the codes of Q_i satisfy the predicate Q_i only in one particular order (Allowing more than one order of those codes would increase the risk of an honest P inadvertently triggering the attack.).

Suppose P has calculated the allowed codes based on I, and the voter has chosen whether to vote using the left or right column. A dishonest P had the possibility to choose to trigger the attack if code z_i appears in the "vote" column and Q_i is true of the "audit" column. The first of these events occurs with probability $n/10^\kappa$. The second one occurs with probability, say, p. Because the two probabilities are independent, their conjunction occurs with probability $np/10^\kappa$. Because these joint events are exclusive for different values of i, we have that P had with probability $np\ell/10^\kappa$ the possibility to trigger the attack.

Unfortunately for the attacker, an honest P may also trigger the attack. This will happen if it inadvertently chooses codes that match the codes of some Q_i (this occurs with probability p), and puts in the order of Q_i (since there is only one order, this occurs with probability $1/n$). If this inadvertent triggering happens, it will cause the voter's vote to be rejected if the code z_i is not a valid code; this occurs with probability $1 - n/10^\kappa$. Thus, the proportion of votes that the attacker causes to be rejected because of honest P's is $(p\ell/n)(1 - n/10^\kappa)$.

If an honest P inadvertently triggers the attack, the attacker may be lucky because the substituted code turns out to be a a valid code. The probability of this combination of events is $(p\ell/n)(n/10^\kappa)$.

Recall that the fraction of honest P's is f. Thus, the proportion of votes the attacker inadvertently causes to be rejected is R times as many as he successfully changes, where

$$R = \frac{f(p\ell/n)(1 - n/10^\kappa)}{(1-f)np\ell/10^\kappa + f(p\ell/n)(n/10^\kappa)} = \frac{f\ (1 - n/10^\kappa)\ 10^\kappa}{((1-f)n + f)\ n}$$

Even if the fraction f of honest P's is much lower, say 10%, still about 70 times more votes will inadvertently be rejected than those successfully modified.

3.4.2 Ballot privacy

If an attacker controls P and S together, it can easily find how a voter voted, even if H is honest. P can leak the link between candidates and codes to S. Nonetheless, we argue that an attacker that controls only one of P or S cannot discover how a voter voted.

3.4.2.1 Privacy analysis assuming honest S, H and corrupt P

We claim that if S is honest, an attacker who controls P cannot find how a voter voted. An attacker who controls P has all the ciphertexts and their randoms and plaintext, together with the value C^* entered by the voter, and the set of values C, $C_{x'}$, D_1, E, P_{total} published by S. The values C^*, C, $C_{x'}$, D_1, E are determined pseudorandomly from the high entropy value K and randoms chosen by S that are not available to the attacker. Therefore these values do not reveal the vote.

The proof P_{total} is a non-interactive zero-knowledge proof (NIZKP). Such a NIZKP does not reveal the secret value that is the knowledge to be proved to its verifier; therefore, P is not able to learn how the voter voted. Our use of a signature-based proof of knowledge (SPK) technique, by using the Fiat-Shamir heuristic, to implement the NIZKP, ensures universal verifiability while preserving user privacy [1].

3.4.2.2 Privacy analysis assuming honest P, H and corrupt S

We claim that if P is honest, then an attacker who controls S can't find how a voter voted. The candidates $a_1, \ldots, a_n$ are encrypted by P with the teller's public key y and the server S does not know the corresponding private key z. (We show that our new variant of Elgamal is secure, in Appendix A.2.) Moreover, P being honest does not reveal the link between candidates and codes for the column that voter has used for voting. Therefore, S has a ciphertext representing V's choice, but no other information about it to find how V

[1] Pass [Pas03] argued that in the common reference string model or the random oracle model non-interactive zero-knowledge proofs do not preserve all of the properties of zero-knowledge proofs, e.g. they do not preserve deniability from the prover, although the proof does not reveal the secret value to the verifier. However deniability from a prover is not required in our case.

voted.

3.4.2.3 Privacy analysis assuming corrupt H

Assuming H has no wireless or wired interfaces, the situation for privacy is same in the case when P is honest and dishonest H and S are controlled by the attacker as the case that H is honest. However, if H and P are controlled by the same attacker then H could leak some information to P via the κ digit code C^* that V enters into P. For example, instead of performing the correct computation, H could just output the last entered code (which belongs to the chosen candidate). This violates the voter's privacy as the attacker will learn how the voter voted. However, if the attacker performs this attack on a large scale then this misbehaviour would be detected, as S will reject the vote and a lot of rejections will lead to an enquiry to find what went wrong.

3.5 Discussion

Performance: One might think that the search that platform P has to perform to find ciphertexts matching a given code is inefficient. On average, the platform P has to compute 10^κ El Gamal ciphetexts in order to find one that matches the code. If $\kappa = 4$, this is 10,000 ciphertexts. However, as explained, P does not have to compute each of these from scratch. It computes the first one from scratch, and then from a ciphertext (α, β) it computes the next one as $(\alpha g, \beta y)$. Hence, to find a ciphertext matching the code, P computes one ciphertext (i.e. two exponentiations and a multiplication) followed by (on average) 10^κ multiplications.

We conducted a simple experiment to see how long this would take in practice. We programmed the relevant part of the platform P in Python, and we used a Macbook Air computer. Once again, we assume $\kappa = 4$. For a given plaintext and code, finding a ciphertext that matches the code took on average 0.24 seconds. (The worst case in our sample of 1000 trials was 1.72 seconds.) If there are $n = 4$ candidates, then 8 codes are

required; thus, the platform P computes the entire code page in $8 * 0.24 < 2$ seconds on average.

Re-usability of token H: In order for H to be reused for multiple elections, a different key K must be used each time. To see the necessity of this requirement, we show that an adversary who controls P could manage to learn K over the course of two elections using the same K. Since P knows $2n$ candidates for x^*, and since $C_{x^*} = Kh^{x^*}$ is publicly known, adversary can compute $2n$ candidates keys for K, as $C_{x^*}\ (h^{x^*})^{-1}$. If K is re-used in two elections, he can intersect the set of $2n$ candidates from the first election with the $2n$ candidates from the second election, and will likely find only one value in common, namely real key K.

Therefore, a different key K must be used for each election. A naive way to achieve this is to store multiple values K_1,K_2, ..., K_n, each one dedicated to a particular election. However, this would require fixing the number of elections in advance. To overcome this restriction, we suppose that the server S has a fixed set of keys $k_1, ..., k_\ell$ for each voter, and H has the public counterparts $K_i = y^{k_i}$. For each election, a set of ℓ short random values $e_1, \ldots, e_n$, are publicly announced. Each user programs their H by typing these values into H. It computes a new election key $K = \prod_{1 \le i \le \ell} K_i^{e_i}$, and the server computes the voter's key as $k = \sum_{1 \le i \le \ell} e_i \cdot k_i$.

Recovering key from H: An attacker might try to recover the key K stored in H, e.g. by solving Hidden Number Problem [Aka09]. However, such attack would require very high number of calls which would be impractical (and H could be rate-limited). Even if an attacker manages to recover the key, it affects privacy (not integrity), and it requires the co-operation of the voter.

Error-detecting codes: As described in Section 3.3.5, S must use a brute-force search to determine what code V entered as a candidate choice. This search might fail if S is

unable to match voter's submitted vote code C^* due to a typing mistake by V.

Error-detecting codes could be employed to help collaborating and honest S,P,H detect typing mistakes. For example, the Luhn algorithm [Luh60] could be used by P to add a single digit checksum after each candidate's code in both columns A and B. Instead of entering κ digits for each candidate, V would enter $\kappa + 1$ digits. Before computing C^*, token H would check whether all checksum were correct; if not, H would ask V to retype them.

Error recovery: Several kinds of errors can arise, which could result in a voter's vote being rejected. Some errors arise because of the voter's mistakes (errors in inputs, for example); others could arise because of malicious behaviour by P, H or S. Another possible error could occur if during the search for the voter's input to H, the server S finds that there are more than one possible input (a clash due to the small number of digits in H's output). Unfortunately, it is difficult to put error recovery procedures in place without compromising the security properties. In particular, a situation in which the platform P starts again could allow it to cheat undetectably. Therefore, if any of these kinds of errors arise, the voter is forced to abandon the attempt to vote and use an alternative mean (such as polling station voting). We hope to improve this situation in further work.

DoS attacks by P: If P is malicious, it could refuse to cooperate in casting a vote for V, perhaps by not posting values on BB, or by posting incorrect values. V can observe this behaviour and vote using a different P. Moreover, V can inform election authorities about the possibility of election specific malware, which could be investigated later.

A more subtle attack by P would be to simulate following the Du-Vote protocol honestly, and show a faked bulletin board to V instead of the real BB. So V needs to check BB from another P, or convey sufficient information to a trusted third party so that it can perform the BB verification steps from Section 3.3.8 on V's behalf. As previously mentioned in Section 3.3.5, conveying this information does not harm V's privacy.

With either of these attacks, the worst effect P can have is denial of service. P cannot modify the votes.

Malicious S: One might think that a malicious S could ignore any malicious behaviour it detects from P. But S is expected to create a public audit trail on BB. If many voters or observers detect cheating behavior in that audit trail, the collusion between S and P will be detected. So if S wants to maintain its reputation as an honest voting service, it cannot engage in large-scale, detectable attacks.

Limitation on the number of candidates: Du-Vote does not place an upper bound on the number of candidates in an election. But, from a usability perspective, it would be tedious for voters to enter a long string of digits into H. This problem can be addressed by changing the set of codes that are typed into H. Currently, the user types all n codes of the audit column, plus one code for her vote. But alternative arrangements could require the user only to type some of the audit codes.

3.6 Comparison with the related work

In this section we briefly compare Du-Vote with some existing internet voting systems that makes efforts to overcome untrusted platform problem.

- The system proposed by Haenni and Koenig (explained in section 2.3.2) come closest to Du-Vote. Their system requires a trusted device for voting over the internet on an untrusted platform. Recall that in their system, the platform generates encryptions in the form of barcodes, the voter makes her selection on the trusted device by scanning chosen candidate's barcode and later transfers her vote to a trusted computer. That device requires a camera, matrix barcode reader, and uplink to a computer. Compared to Du-Vote's hardware token, that device is complicated and difficult to make trustworthy. Their hardware device learns the voter's vote. Moreover, a

camera built-in into an untrusted platform may record a voter's behaviour while she scans her chosen candidate's barcode. In Du-Vote voters do not need to trust the hardware token for privacy.

- Neumann et al. [NFVK13, NV12] extend JCJ/Civitas with a smart card to manage voter credentials, however, unlike Du-Vote that scheme does not protect the voter's privacy from an untrusted machine. For integrity smart card needs to be trusted and cut-and-choose mechanism is used to make sure that correct vote is encrypted. Several other variants of Civitas improve the usability of the aspects related to verifiability [SHKS11] and coercion-resistance [CH11], but similar to JCJ/Civitas they still rely on trusted machines.

- Chaum et al. [CFN+11] introduce a *computational assistant* to verify the vote cast on a voting machine inside a polling station. This system is designed for polling-place voting, whereas Du-Vote is suitable for remote voting.

- SureVote and Pretty Good Democracy (PGD) use the voter's computer as an untrusted communication channel. The computer is used only to send and receive codes from the sheet. Integrity depends on secrecy of the code sheets in both SureVote and PGD , but Du-Vote does not require code pages to be secret.

- Helios let voters audit/challenge their computer to prove it acted honestly in encrypting their candidate choice. However, the voters need trustworthy computers to verify those proofs. Du-Vote does not require a trustworthy computer. In Helios, the voter's computer automatically learns how the voter voted. Du-Vote guarantees privacy as long as either P is honest, or H and S are honest. This is better because voters who are technically competent can protect their privacy by making sure that their own computer is free from any election-specific malware. Voters who don't trust their computers can rely on trustworthiness of the devices (H and S) provided by the election authorities.

- Remotegrity assumes a secure means of distributing the ballots and authorization cards. Since Du-Vote needs to securely distribute a hardware token that is used to authenticate votes, Remotegrity could be seen as a paper-based analogue of Du-Vote. Both systems solve the problem of untrusted computing platforms, but Du-Vote does so without relying on paper, invisible ink and scratch-offs, or auditing of printed ballots (which is required by Scantegrity II and necessitates use of a random beacon and a distinguished group of auditors).

- In [KR16], authors provide an extensive security analysis of Du-Vote. They show some attacks and also propose changes to the system that would avoid such attacks.

3.7 Concluding remarks

Conventional wisdom used to hold that remote voting cannot be secure, because voters' computers themselves are not secure [JRSW04]. Research in the last decade, though, indicates that voters do not need fully trustworthy computers. With voter-initiated auditing [Ben06, Ben07], computers prove to voters that votes are properly encrypted—but voters are instructed to seek out many computers, so that at least one is likely to be honest, and computers must still be trusted for privacy.

In this chapter we proposed to use a dedicated hardware token to overcome the untrusted platform problem. By using this token, Du-Vote doesn't require voters to trust general-purpose computer for integrity, nor does it require voters to trust computers for privacy. We also discovered that the token need not be trusted for integrity provided that the adversary cannot communicate with it after the election opens. We are now optimistic, based on the techniques developed in this work, that secure internet voting systems can be deployed without requiring any trust in the computers used by voters.

Could we replace Du-Vote's special-purpose tokens with smartphones? Users might download a simple app that simulates the Du-Vote token. Preventing untrustworthy

smartphones from conspiring with other computers to violate verifiability and privacy—especially preventing communication with the adversary—would be challenging, as would be distributing the secret values stored by tokens. Du-Vote's security degrades gracefully when the token is malicious, so it might be possible to address these concerns. If so, smartphones could be profitably employed to improve usability.

CHAPTER 4

Trivitas: Improving verifiability for voters

4.1 Introduction

As mentioned in the introduction of this thesis, election verifiability in presence of a bulletin board is conveniently split in three notions i.e. Individual verifiability, Universal verifiability, and Eligibility verifiability.

One problem with the way election verifiability, especially *individual verifiability* is usually achieved is that the voter is not able to see her vote at the time she visits the bulletin board after casting a ballot. This is the case in Helios, Prêt à Voter, JCJ/Civitas, and others. The reason is in order to achieve the property of coercion-resistance, which asserts that the voter shouldn't be able to prove to a potential coercer how she voted. Therefore, individual verifiability is achieved by indirect means; the voter can check that the encrypted ballot is present, and has some other evidence (perhaps based on auditing) that the encrypted ballot really represents her vote. Moreover, after the ballots are anonymized, the voter even looses track of her encrypted ballot.

Voters are likely to find this indirect achievement of individual verifiability unsatisfactory. This feeling has arisen in the focus groups that were held as part of the EPSRC project *Trustworthy Voting Systems* [SLC+11]. Four hour-long managed discussions among groups of about 10 citizens were arranged by a professional facilitator, with the aim of soliciting people's opinions about Prêt à Voter. In at least two of the discussions, participants questioned the worthiness of checking the presence of their ballot on the bulletin board, given that they did not have any direct evidence that the ballot really contained their vote. The issue has also been mentioned by Adida and Neff [AN06], where the requirement that verifiability should be *direct* and *end to end* has been highlighted.

Our contribution. In this chapter we introduce Trivitas, a protocol based on JCJ/Civitas (described in Chapter 2 Section 2.3.1.1). We show how the credentials of JCJ/Civitas can be adapted to improve individual verifiability. In particular, we show how voters can see their own vote in plaintext present on bulletin board, making the verification experience direct, and more intuitive; without loosing coercion-resistance.

Our first idea is the notion of trial votes. A trial vote is a vote that is cast along with real votes, but will not be counted. It will be decrypted and exposed along the way, in a way that is traceable by the vote who has cast it. Since most of the system components are not able to distinguish trial and real votes, it gives confidence in the correct handling of all votes. This is an extension of the ideas of auditing in [CRS05, Adi08], with the crucial difference that the auditing process is performed not only in the phase of casting a ballot, but is spread throughout the whole election process. In other words, a trial credential will function as a marker whose sign is that the handling of this ballot should be made transparent, by e.g. decrypting and publishing its contents at every stage. There are a few technical difficulties with that, because trial ballots can fulfil their role only as long as they are not identified as such by possibly corrupted agents in the voting system. To avoid this problem, we make use of the fact that the decryption key is distributed among a set of tellers and propose to decrypt trial ballots by running a decryption mix [Cha04, CRS05] among the tellers.

Our second idea is based on the following observation: real credentials are indistinguishable for anyone (except for the voter) from trial credentials and fake credentials (an element that JCJ/Civitas introduces to enable coercion-resistance). Therefore, without compromising coercion-resistance, for each ballot (be it real, trial or fake) we can publish after anonymization (done by a re-encryption mixnet [JJR02]) its corresponding credential and the decrypted vote. This allows a voter to verify that all its votes have made it into the final tally: its real vote, its trial vote and its possible fake votes. There is again a technical difficulty, related to eligibility verifiability and coercion-resistance: trial votes and fake votes have to be eliminated from the final tally in a publicly verifiable way, hence a coercer could observe that the credential obtained from the voter is fake. To avoid this problem we again make use of a decrytion mix run by the tellers: there is no way to link the decrypted contents of a ballot to ballots that will be eliminated to enforce eligibility.

Outline of the chapter. In section 4.2 we review election verifiability in JCJ/Civitas. Then, we make specific our critique of individual verifiability, that can be extended to systems like Prêt à Voter and Helios. In section 4.3, we describe our proposal. In section 4.4, we show initial ideas about how trial credentials could be used to improve universal verifiability and recoverability. Finally, in section 4.5 we argue that changing JCJ/Civitas in the way that we propose does not compromise the coercion-resistance guarantees of the original system, and we give a hint of how the proof of [JCJ05] could be adapted to prove coercion-resistance for the new system.

4.2 Verifiability in JCJ/Civitas

We gave an overview of JCJ/Civitas in Chapter 2 in Section 2.3.1.1. In this section we explain how verifiability is ensured in JCJ/Civitas.

4.2.1 Election verifiability

Recall that in JCJ/Civitas the re-encryption mix, the plaintext equivalence tests and the decrypted votes are accompanied by zero-knowledge proofs to ensure the operations have been correctly performed. Universal verifiability and eligibility verifiability are achieved from these proofs by posting them on the bulletin board, this way allowing any external auditor to check their validity.

More specifically, **Universal verifiability** and **eligibility verifiability** in JCJ/Civitas are achieved as follows:

- Mix proofs allow auditors O to verify that a ballot coming out of the mix net Mix (i.e. belonging to the set BB_{mix}) corresponds to some recorded ballot (i.e. belonging to the set BB_{cast}), and also that no recorded ballot has been discarded.

- PET proofs and proofs $\mathsf{P}_{s\nu}$ from the ballot allow O to verify that only votes coming from eligible voters are kept on the bulletin board for final decryption.

- Decryption proofs allow O to verify that all countable votes have been correctly decrypted.

Individual verifiability in JCJ/Civitas is achieved as follows:

1. Voter must trust her voting platform P that her cast ballot correctly encodes her vote.

2. Voter can check the bulletin board BB to see that her cast ballot has been correctly recorded.

3. Universal verifiability of mix nets Mix ensures that all the recorded votes are correctly mixed, and therefore the vote cast by V is included in the set of mixed ballots.

4. Universal verifiability of plaintext equivalence tests (pet) ensures that at least one copy of V's ballot is kept after the elimination of duplicates. Moreover, the proof P_Q that the voter obtains during registration ensures that her private credential corresponds to a public credential in the electoral roll ER. Universal verifiability of mix nets ensures that a re-encryption of her public credential is also present in the anonymized electoral roll BB_{aer}. Universal verifiability of pets ensures that valid ballots are not eliminated when credentials are checked against the anonymised electoral roll BB_{aer}. Altogether, these give to V an assurance that her ballot is identified as coming from an eligible voter and is not eliminated during the enforcement of eligibility.

5. Finally, universal verifiability of distributed decryption ensures that V's ballot is correctly decrypted and tallied.

Some systems, e.g Prêt à Voter [CRS05] and Helios [Adi08], improve the first point with a *cut-and-choose* mechanism, that allows the voter to audit a ballot before casting a

vote.

Our critique of individual verifiability in JCJ/Civitas, Prêt à Voter and Helios refers more generally to systems that rely on universal verifiability to achieve end-to-end individual verifiability. The points 3, 4 and, 5 above to ensure individual verifiability require complex mathematical operations and the corresponding verification algorithms must be run on a trusted platform. Moreover, even if the corresponding zero-knowledge proofs are rigorously tailored to ensure the desired properties, the ordinary voter may be left wondering if her vote has actually been counted in the final tally. While auditors may be expected to have access to trusted platforms and to understand the concepts behind zero-knowledge proofs, we do not consider these assumptions satisfactory when individual verifiability is considered.

Let us also note the following limiting aspect of the cut-and-choose mechanism in [Ben06, CRS05] and [Adi08]. In all these systems, the audited ballot is handled as a real ballot, but only up to the point when the voter decides to audit it. Once audit is chosen, the ballot is discarded. Only one ballot gets to be cast and submitted to the bulletin board. In our proposal, the audited ballot, that we call a *trial* ballot, will be handled in the same way as a real ballot at each phase of the protocol, while additionally playing its role as an audit ballot. In particular, it will be posted on the bulletin board before mixing and handled subsequently in the mix and in the decryption phase. The traditional cut-and-choose guarantees are recovered in this setting by tracking the trial ballot and the corresponding decrypted vote on the bulletin board before the mix.

4.3 Trivitas: trial credentials and universal decryption

The first proposal of Trivitas is the notion of trial credentials. A trial credential is a credential that allows a voter to cast a vote that will not be counted in the final tally, but will appear on the bulletin board at several stages of the tabulation phase. Its purpose is

to allow the voter to gain confidence in the correct operation of the system. Trial ballots are identifiable as such only by a threshold set of tellers, thus any component of the system has to treat the set of all ballots in the same way. We show how trial credentials can be implemented in the context of JCJ/Civitas and show their immediate benefit for individual verifiability.

Moreover, we propose another addition to JCJ/Civitas, independent of trial credentials, that brings a further improvement to individual verifiability: we decrypt and publish the content of all ballots after the mix. Therefore, for every ballot that a voter has cast (real, trial or fake), she can verify that the corresponding credential and vote occur on the bulletin board after the mix. This gives a direct evidence to the voter that her ballots, and most importantly her real ballot, have been correctly constructed by the platform P and processed by the mix network.

4.3.1 Overview of proposed additions

In this section we explain the additions that we propose in each phase of JCJ/Civitas:

Registration. Recall that in JCJ/Civitas each voter receives private share of her real voting credentials $s_1, \ldots, s_n$. In Trivitas, mirroring the set of real credentials $s_1, \ldots, s_n$ we assume that registrars also generate a set of *trial credentials* $s_1^t, \ldots, s_n^t$. The set $s_1^t, \ldots, s_n^t$ is constructed and distributed to voters following the same protocols as for $s_1, \ldots, s_n$, thus we can assume the same security properties: in particular, trial credentials are indistinguishable from real credentials and fake credentials, for anyone but for the voter that receives its share. In addition to the electoral roll ER, now we have a *trial roll* TR, that contains the public parts of the trial credentials $\mathsf{enc}(s_1^t; pk_T), \ldots, \mathsf{enc}(s_n^t; pk_T)$. We omit randomness from this notation of encryption as it is clear from the context.

Voting. In addition to constructing a ballot $(\mathsf{enc}(s_v; pk_T), \mathsf{enc}(\nu; pk_T), \mathsf{P}_{\mathsf{corr}}, \mathsf{P}_{\mathsf{s}\nu})$ as in JCJ/-Civitas, voter V additionally constructs a trial ballot $(\mathsf{enc}(s_v^t; pk_T), \mathsf{enc}(\nu^t; pk_T), \mathsf{P}^t_{\mathsf{corr}}, \mathsf{P}^t_{\mathsf{s}\nu})$

and uploads both the real ballot and the trial ballot to the bulletin board (at implementation level, it could be decided if a ballot construction form would be used twice or if the system would allow the construction of both ballots at the same time). Both real and trial ballots are added to the set BB_{cast}.

Verification of proofs and mixing. At the time of ballot validation (i.e. just after the voting phase ends), the trial roll TR is anonymized with mix and mixed trial credentials are stored in a set BB_{atr}. This is done in addition to electoral roll ER anonymization and mixed credentials being stored in set BB_{aer}. All other verification operations are performed as done in verification phase of JCJ/Civitas.

Tabulation. The tellers T first start with performing plaintext equivalence test of ballots in BB_{valid} against BB_{atr} (Recall that BB_{valid} is the set of cast ballots with valid zero-knowledge proofs of correctness). For all ballots for which this pet returns true, the tellers decrypt the corresponding credential and the corresponding vote and publish them on the bulletin board: this is represented as a set BB_{TrialsA}. Formally, tellers publish on the bulletin board the result of $\mathsf{PETDecryptUnlink}(BB_{\mathsf{valid}}, BB_{\mathsf{atr}})$, where the motivation, specification and the algorithm for $\mathsf{PETDecryptUnlink}$ are discussed in section 4.3.3.

The set of all the valid ballots (that includes the trial ballots i.e. BB_{valid}) are again sent to the mixnet Mix for anonymization. Just after this mix and before the pet tests for eligibility enforcement, all ballots are decrypted and their corresponding decrypted credentials and decrypted votes are posted on the bulletin board as a set $BB_{\mathsf{DecBallots}}$. To preserve coercion-resistance of the system, this has to be done in a way that does not link the published credentials and votes to the corresponding ballot. We propose the use of a decryption mix $\mathsf{DecryptUnlink}(BB_{\mathsf{mix}})$, whose idea is discussed in section 4.3.3 as well.

At the time of eligibility enforcement, credentials in BB_{mix} are additionally tested against the trial roll BB_{atr}. If a ballot is identified as trial, it is not discarded but is labelled as such on the bulletin board. Finally, all ballots that remain on the bulletin

board after eligibility enforcement are decrypted and only votes that do not correspond to trial ballots are tallied. If a ballot is labelled as a trial ballot, the corresponding credential is also decrypted. The decrypted trial ballots after the tabulation form the set BB_{TrialsB}.

4.3.2 Individual verifiability in Trivitas

A voter can trace her trial vote in each phase of the system: it should be present on the bulletin board in the set BB_{TrialsA}, after the voting phase, and in the set BB_{TrialsB}, after the tabulation phase. Moreover, relying on the decryption of all the ballots after the mix, the result of which is the set $BB_{\mathsf{DecBallots}}$ on the bulletin board, the voter can check that the encryption and the mix has been correct for all of her cast ballots: the one with a real credential, the one with a trial credential and possibly the ones with fake credentials. Hence, fake credentials can also be used for the purpose of end-to-end individual verifiability. In summary, the tests that voter V_i perform and the properties that are assured are shown in table 4.1.

	Verifiability test	**Assured property**
$\mathcal{IV}_1$	The pair (s_i^t, ν_i^t) occurs in the set BB_{TrialsA} on the bulletin board	The platform has correctly encoded V_i's votes and V_i's ballots have been correctly recorded on the bulletin board
$\mathcal{IV}_2$	The pairs $(s_i, \nu_i), (s_i^t, \nu_i^t)$ and all (s_i^f, ν_i^f) occur in the set $BB_{\mathsf{DecBallots}}$ on the bulletin board	All of V_i's submitted ballots have been input in the mixnet Mix and have been correctly processed and output by Mix
$\mathcal{IV}_3$	The pair (s_i^t, ν_i^t) occurs in the set BB_{TrialsB} on the bulletin board	V_i's intended vote occurs in the final outcome

Table 4.1: Individual verifiability tests and assured properties in Trivitas

Let us argue why all these tests are valid, in the sense that, if they are satisfied for the voter V_i, then the claimed properties hold with high probability for all of V_i's ballots: trial, real and fake. We leave rigorous proofs along the lines of [JCJ05, KRS10] as future work,

and perform only an informal analysis below. Recall that JCJ/Civitas makes the trust assumption that: *at least one of tellers T is honest*. We make the same trust assumption for Trivitas.

We explain below why $\mathcal{IV}_1$, $\mathcal{IV}_2$, and $\mathcal{IV}_3$ are valid:

$\mathcal{IV}_1$. The assumption that at least one T is honest ensures that the decryption of trials is correct: the published trial pair is indeed the content of V_i's trial ballot, that is present on the bulletin board. Then, the fact that a trial credential is indistinguishable from a real credential ensures that a cheating voting platform or a cheating bulletin board has to make a random guess if it wants to substitute a voter's ballot, thus having at least a 50% probability of being detected.

$\mathcal{IV}_2$. Similarly, the assumption that at least one T is honest ensures that the set of published pairs ($BB_{\mathsf{DecBallots}}$) corresponds to the decryption of ballots output by Mix (BB_{mix}). Therefore, $\mathcal{IV}_2$ assures that all of V_i's ballots are correctly output by the Mix. Moreover, note that $\mathcal{IV}_2$ increases the assurance offered by $\mathcal{IV}_1$, because V_i can check the correct construction and transmission of all her ballots. Still, $\mathcal{IV}_1$ is useful to detect a potential problem as early as possible and also to identify more precisely the elements of the system that have caused the problem. For instance, we will see in section 4.4 how $\mathcal{IV}_1$ allows for recoverability when a problem is detected before the mix.

$\mathcal{IV}_3$. The parallel decryption of trial votes gives some evidence to the voter that votes are not arbitrarily eliminated during eligibility enforcement. This is again formally ensured by assumption that at least one T is honest.

4.3.3 Anonymous PETs and distributed decryption with ciphertext-plaintext unlinkability

We now come back to two cryptographic components of the proposed system that have been left out in section 4.3.1: PETDecryptUnlink and DecryptUnlink. PETDecryptUnlink is used to decrypt trial ballots while keeping them indistinguishable from other ballots.

This is necessary for being able to rely on trial ballots to audit the system even after they are decrypted. DecryptUnlink is used to decrypt all ballots, without revealing the link between individual ballots and their content. This is necessary to preserve coercion-resistance: otherwise, a coercer could detect that a ballot cast with a fake credential has been eliminated before the final tally.

Recall that votes and credentials are encrypted with public key pk_T, whose corresponding private part sk_T is distributed among T. In the following, we assume $T = \{T_1, \ldots, T_n\}$. For simplicity, we denote $\mathsf{enc}(s; pk_T) =$ es,we denote $\mathsf{enc}(\nu; pk_T) = \mathsf{e}\nu$, and the encrypted credentials on TR (i.e. BB_{atr}) as $\mathsf{es}'_1, \ldots, \mathsf{es}'_k$. The specification for PETDecryptUnlink and DecryptUnlink is as follows:

PETDecryptUnlink

Input: $\mathcal{S} = (\mathsf{es}_1, \mathsf{e}\nu_1), \ldots, (\mathsf{es}_m, \mathsf{e}\nu_m)$ and $\mathsf{TR} = \mathsf{es}'_1, \ldots, \mathsf{es}'_k$

Output: $\mathcal{O} = \{(s, \nu) \mid \exists i_{\mathcal{S}} \in \{1, \ldots, m\},\ \exists i_{TR} \in \{1, \ldots, k\},$
$\mathsf{dec}(\mathsf{es}_{i_S}) = \mathsf{dec}(\mathsf{es}'_{i_{TR}}) = s\ \&\ \mathsf{dec}(\mathsf{e}\nu_{i_S}) = \nu\}$

Unlinkability: for all $(s, \nu) \in \mathcal{O}$, the index i_S of $(\mathsf{enc}(s; pk_T), \mathsf{enc}(\nu; pk_T))$ in S is indistinguishable from a random number in $\{1, \ldots, m\}$.

DecryptUnlink

Input: $\mathcal{S} = (\mathsf{es}_1, \mathsf{e}\nu_1), \ldots, (\mathsf{es}_m, \mathsf{e}\nu_m)$

Output: $\mathcal{O} = \{(s, \nu) \mid \exists i_S \in \{1, \ldots, m\}.\ \mathsf{dec}(\mathsf{es}_{i_S}) = s\ \&\ \mathsf{dec}(\mathsf{e}\nu_{i_S}) = \nu\}$

Unlinkability: for all $(s, \nu) \in \mathcal{O}$, the index i_S of $(\mathsf{enc}(s; pk_T), \mathsf{enc}(\nu; pk_T))$ in S is indistinguishable from a random number in $\{1, \ldots, m\}$.

Our proposed implementation for PETDecryptUnlink and DecryptUnlink is an adaptation of the decryption mix idea present in [Cha04, CRS05] to the case of distributed El-Gamal. This setting has already been studied in e.g. [FMM+02, Fur04], that show how the shuffle can be made verifiable. However, since we do not require a verifiable shuffle for our application (a misbehaviour during decryption would be detected by the voter by simply observing the trial credentials) our algorithms are more straightforward and do not provide

zero-knowledge proofs. We only describe the algorithm for PETDecryptUnlink, the second algorithm being similar and more simple. We describe the algorithm for one trial credential et from the set TR and the algorithm is applied iteratively to every credential. (Recall the shape of $\mathsf{enc}(m; pk_T)$ is a pair $(g^r, m \cdot y^r)$.)

PETDecryptUnlink. For all public trial credentials et in TR, the parties $T_1, \ldots, T_n$ (holding private key shares $x_1, \ldots, x_n$) run the following protocol:

Initial phase (can be run publicly by any party).

Assume $\mathsf{et} = (a, b)$ and, for all $1 \leq i \leq m$, assume $\mathsf{es}_i = (a_i, b_i)$. Compute and publish $p_1 = (\frac{a_1}{a}, \frac{b_1}{b}), \ldots, p_m = (\frac{a_m}{a}, \frac{b_m}{b})$. The input for T_1 in the next phase is $(p_1, \mathsf{es}_1, \mathsf{e}\nu_1), \ldots, (p_m, \mathsf{es}_m, \mathsf{e}\nu_m)$.

PET phase (being run privately and consequently by each of $T_1, \ldots, T_n$).

Let $(p_1, \mathsf{es}_1, \mathsf{e}\nu_1), \ldots, (p_m, \mathsf{es}_m, \mathsf{e}\nu_m)$ be the input for T_i. Create new random numbers $r_1^p, \ldots, r_m^p \in \mathbb{Z}_q^*, r_1^s, \ldots, r_m^s \in \mathbb{Z}_q^*, r_1^\nu, \ldots, r_m^\nu \in \mathbb{Z}_q^*$ and compute

- $(c_1, d_1) = \mathsf{renc}(p_1, r_1^p), \ldots, (c_m, d_m) = \mathsf{renc}(p_m, r_m^p)$
- $\mathsf{es}'_1 = \mathsf{renc}(\mathsf{es}_1, r_1^s), \ldots, \mathsf{es}'_m = \mathsf{renc}(\mathsf{es}_m, r_m^s)$
- $\mathsf{e}\nu'_1 = \mathsf{renc}(\mathsf{e}\nu_1, r_1^\nu), \ldots, \mathsf{e}\nu'_m = \mathsf{renc}(\mathsf{e}\nu_m, r_m^\nu)$

Partially decrypt $(c_1, d_1), \ldots, (c_m, d_m)$, i.e. compute $d'_1 = \frac{d_1}{c_1^{x_i}}, \ldots, d'_m = \frac{d_m}{c_m^{x_i}}$.

Choose a permutation σ of $\{1, \ldots, m\}$ and publish

$$((c_{\sigma(1)}, d'_{\sigma(1)}), \mathsf{es}'_{\sigma(1)}, \mathsf{e}\nu'_{\sigma(1)}), \ldots, ((c_{\sigma(m)}, d'_{\sigma(m)}), \mathsf{es}'_{\sigma(m)}, \mathsf{e}\nu'_{\sigma(m)})$$

This is the input for T_{i+1}.

Decryption phase (run jointly by $T_1, \ldots, T_n$). For all $(p, \mathsf{es}, \mathsf{e}\nu)$ in the output of T_n: if $p = 1$, perform a distributed decryption of es and of $\mathsf{e}\nu$ and make the result part of the output set: $\mathcal{O} := \mathcal{O} \cup \{(\mathsf{dec}(\mathsf{es}), \mathsf{dec}(\mathsf{e}\nu))\}$.

If at least one of $T_1, \ldots, T_n$ behaves honestly, PETDecryptUnlink satisfies also the unlinkability requirement, as formalized and proved in [Fur04].

Intuitively, the values $p_1, \ldots, p_m$ after the initial phase represents an encryption of a number that record whether the credential encrypted in et is equal to the credential encrypted in es: if that number is 1, then it is equal, otherwise not.

Then, the goal of the next phase is to decrypt $p_1, \ldots, p_m$ in order to determine the encrypted ballots that correspond to trial credentials. These elements are passed from one party to the next party in turn, in order to partially decrypt them with each share of the secret key. As output of T_n, we will get either 1 or a random number.

In addition, we have re-encryptions and secret permutations of ballots. This is performed in order to ensure the additional requirement of unlinkability. So, if one of the p_i at the end is 1, we do not know where that ballot came from, i.e. we don't know the link between the ballot to be decrypted and the corresponding input ballot

4.4 Other properties

In this section we discuss other possible applications of trial credentials.

4.4.1 Universal verifiability

We propose the following universal verifiability test for Trivitas:

	Verifiability test	**Assured property**
$\mathcal{UV}$	All the trials published before the mix are in the set of decrypted ballots after the mix, i.e. $BB_{\mathsf{TrialsA}} \subseteq BB_{\mathsf{DecBallots}}$, and they have the same number of occurrences	The mixnet Mix is correctly processing all the ballots

Table 4.2: UV test and assured property

We propose this test as an addition to the current universal verifiability proofs, not as a replacement: it is more efficient, but probably offers less assurance than traditional

zero-knowledge proofs. On the other hand, this test could also be combined with other tests that offer lesser guarantees of correctness but better performance [BG02], in order to improve their assurance while preserving their efficiency.

Now we show the validity of $\mathcal{UV}$. Because $\mathsf{PETDecryptUnlink}(BB_{\mathsf{cast}}, BB_{\mathsf{atr}})$ does not give away what ballots among BB_{cast} are trials, Mix has to treat all the cast ballots in BB_{cast} uniformly. In particular, if it chooses to cheat on a subset of ballots in BB_{cast}, this subset is random. Therefore, if there are enough trial ballots (this could be ensured for instance by letting observers insert any number of trials), a dishonest behaviour of Mix would be detected with high probability by the test $\mathcal{UV}$.

These arguments hold only when the voting platforms P is trusted. Otherwise, if P and Mix are both controlled by the same attacker the a corrupt Mix could differentiate trial ballots from other ballots when they are decrypted. We address this problem by a variant of Trivitas that does not let the platform learn which ballots are trials, even when they are decrypted (section 4.4.3).

4.4.2 Recoverability from failed verification

What happens when individual verifiability fails, e.g. an incorrect trial vote is published along a voter's trial credential? In general, this issue is quite complex, because it requires procedures to determine who is telling the truth: the voter or the voting system. For Trivitas, our proposed recoverability technique is straightforward and requires only a slight modification to the system: trials are decrypted and published in short time after the ballots are cast and voter V does not have to wait for the end of the voting phase to verify a trial. In this case, if a voter observes a problem with her trial vote on the bulletin board, she should simply re-vote, using a potentially safer platform. The policy for handling duplicate votes would then be to consider only the last vote as being valid, because it is the vote in which the voter has the highest confidence.

However, like in the case of universal verifiability, this solution is not ideal, because an untrusted platform P could make a distinction between trial credentials and valid

credentials, after trial ballots are decrypted. The variant of Trivitas in the next section addresses this issue.

4.4.3 The case of an untrusted voting platform

In Chapter 3, we show how to achieve both privacy and verifiability in the presence of an untrusted voting platform by using a dedicated hardware token. Here we propose a variant of Trivitas whose aim is to allow universal verifiability and recoverability, as discussed in previous section, even in presence of an untrusted voting platform. The main property of this variant is that it preserves the secrecy of the trial credential, while still allowing the voter to verify a trial vote relying on that credential. The cost is a slightly more complicated voting and vote verification experience:

- along with credentials s and s^t, the voter additionally receives (or constructs) two numbers: one corresponding to a random number r and one to $\mathsf{enc}(r; pk_T)$. We may assume that the same protocol is run for obtaining s, s^t and r and hence that the value of r is secret and known only to the voter.

- when constructing a ballot, the voter inputs not only the credential and the vote, but also $\mathsf{enc}(r; pk_T)$.

- when decrypting trial ballots, the tellers T do not decrypt directly the credential $\mathsf{enc}(s^t; pk_T)$ but instead multiply it with $\mathsf{enc}(r; pk_T)$, to obtain $\mathsf{enc}(s^t \cdot r; pk_T)$ (relying on the homomorphic properties of ElGamal) and decrypt it to $s^t \cdot r$. Hence, instead of looking for a pair (s^t, ν^t) on the bulletin board (like in the basic version of Trivitas), the voter would look for $(s^t \cdot r, \nu^t)$ (for usability, one can see that an additive homomorphism, also possible with ElGamal, would be better here).

In this variation of Trivitas, even if the voting platform is compromised, it can not be used to identify which ballots are trials. Hence, trial ballots can also be used for universal verifiability. For recoverability, a trial credential could be used multiple times and the

platform would still be forced to take a 50% chance of getting caught each time when it is cheating. However, compared to our work proposed in Chapter 3, it is more complicated.

4.5 Coercion-resistance in Trivitas

In this section we discuss why Trivitas offers the same coercion-resistance guarantees as JCJ/Civitas. Coercion-resistance in JCJ/Civitas relies on the ability of the voter to create a fake credential s' and a fake proof P'_Q that satisfy the following properties:

- given a pair (s^i, P^i_Q), a coercer can not determine if the pair represents a voter's real credential and proof (s, P_Q) or if it represents a fake pair (s', P'_Q). This is due to the fact that at least one registrar is assumed to be honest and the communication channel used with that registrar is assumed to be untappable.

- if a ballot with a fake credential is submitted, it will be eliminated from the final tally in the tabulation phase, during eligibility enforcement. Crucially, all ballots have been mixed and re-encrypted and at least one member of the mix network is assumed to be honest. This ensures that a coercer can not observe to what credentials correspond the ballots that have not been included in the final tally.

As usual, we also have to assume that there are enough votes for each candidate, so that the coercer can not observe that the voter did not follow her instructions from the mere outcome of the election.

The first addition of Trivitas, trial credentials, does not affect the way in which real ballots and fake ballots are handled by the election system. The only observable difference for the coercer is the presence of decrypted trial ballots at every phase and this does not give any information about real ballots or fake ballots. In particular, the two properties mentioned above remain true in presence of trial ballots.

The second addition of Trivitas, universal decryption, is potentially more problematic for coercion-resistance, since it concerns all the recorded ballots. However, coercion-resistance

is preserved by two crucial points:

- all ballots are decrypted, without making a difference between real credentials, fake credentials and trial credentials. This ensures that, in Trivitas as in JCJ/Civitas, the coercer can not determine if a credential is valid or not.

- the algorithm applied to decrypt all ballots is a decryption mix, i.e. we apply DecryptUnlink(MixedBallots). It may be surprising that a set of anonymized ballots is decrypted with a decryption mix. However, this is needed because tellers must eliminate fake ballots in a publicly verifiable way. In that case, if the coercer could additionally see the contents of all ballots, he could determine what credentials were invalid.

Intution towards a formal proof. A formal proof is out of scope of this thesis but we expect that the coercion-resistance proof for JCJ/Civitas [JCJ05] could be extended to cover Trivitas. To define coercion-resistance for an election system $\mathcal{EVS}$ in a computational model, [JCJ05] considers an ideal system $\mathcal{EVS}_{\mathcal{I}}$ where the outcome of the election is "magically" computed: the adversary can observe only the final outcome and is not able to influence anything more than vote choices for the compromised voters. Then, a system is said to satisfy coercion-resistance if the probability of a polynomial time adversary being able to determine if it has been cheated is roughly the same when the election is run by $\mathcal{EVS}$ as in the case when the election is run by $\mathcal{EVS}_{\mathcal{I}}$.

The proof of coercion-resistance is a reduction to (a variant of) the Decisional-Diffie Hellman (DDH) assumption. The proof relies on a simulator $\mathcal{S}$ that behaves either as $\mathcal{EVS}$ or as $\mathcal{EVS}_{\mathcal{I}}$, depending on whether its input is a Diffie-Hellman tuple or not. If there would be a polynomial time adversary that breaks coercion-resistance, i.e. it has better chances of coercion when $\mathcal{EVS}$ is used instead of $\mathcal{EVS}_{\mathcal{I}}$, then that adversary could be used by the simulator $\mathcal{S}$ to determine if its input is a Diffie-Hellman tuple in polynomial time. This would break the DDH assumption.

To extend this proof to Trivitas all we have to do is to show that the simulator $\mathcal{S}$ of

[JCJ05] can also execute the additional operations, i.e. the management of trial credentials and the universal decryption of all ballots after the mix. This is possible because the simulator of [JCJ05] holds the private key sk_T and can therefore decrypt ballots at any time. This makes it possible to simulate both the audit of trial ballots and the universal decryption. Moreover, the same simulator can easily create trial credentials and trial ballots, this process being similar to the creation of real credentials and ballots.

4.6 Concluding remarks

In this chapter we proposed Trivitas; a protocol that achieves direct and end-to-end individual verifiability, while at the same time preserving coercion-resistance. We propose the use of trial credentials, as a way to track and audit the handling of a ballot from one end of the election system to the other end, without increased complexity on the voter end. Secondly, due to indistinguishability of credentials from random values, we observe that the association between any credential and its corresponding vote can be made public at the end of the election process, without compromising coercion-resistance. The voter has more intuitive and direct evidence that her intended vote has not been changed and will be counted in the final tally.

The idea of trial ballots is not necessarily specific to JCJ/Civitas. We believe it could be implemented in other electronic voting systems as well. The universal decryption of ballots after the mix relies on the notion of credentials, to allow voters to identify their votes, and of fake credentials, to allow coercion-resistance. Credentials are also interesting for eligibility verifiability, possible in JCJ/Civitas but generally not possible in other systems. Hence, it would be interesting to investigate the possibility of adding a credential infrastructure on top of other E-voting protocols.

CHAPTER 5

Caveat Coercitor: Coercion evidence in internet voting

5.1 Introduction

5.1.1 Background and motivation

Current research in electronic voting focuses on trying to design usable voting systems that satisfy the properties of *coercion-resistance* and *verifiability*. Coercion-resistance is a strong fundamental property of electronic voting systems. It states that a voter should be able to cast her vote as intended, even in presence of a coercer that may try to force her to cast a different vote. On the other hand, verifiability allows to verify that the outcome of the election reflects correctly the voters choice. A voter can verify that the ballot has been correctly recorded on the bulletin board, and anyone can verify that the recorded ballots from eligible voters are tallied correctly.

Especially in internet voting, coercion-resistance and verifiability are extremely hard to achieve together and this leads to schemes of questionable usability or to trust assumptions that are difficult to meet in practice. Perhaps the system that comes closest to satisfying both those properties in their strongest form is JCJ/Civitas (see section 2.3.1.1). However, it does so at the cost of usability. In JCJ/Civitas and in our work in Chapter 4, to achieve coercion-resistance and verifiability:

- the voters are required to run a multi-party protocol with registrars in order to construct their credentials.
- to avoid coercion, the voters must be capable of creating and passing off fake credentials and fake proofs.
- a trusted registrar and an untappable channel for that registrar are necessary.

Helios voting system, however, is highly usable but it has very weak methods to resist coercion. For example, in some variants of Helios (and also in the Estonian system), only the last vote counts. This is supposed to help a voter resist coercion by re-voting. However, if the voting credentials have been leaked, an attacker can always override a legitimate vote.

Moreover, in any e-voting system, there may be cases where the voter is subject to *silent coercion*, which means being coerced without noticing. This can happen when the voting credentials are leaked for various reasons: dishonest registrars, insecure communication channels, malware on the voting platform. In that case, the voter can be subject to an impersonation attack and the intended vote may be replaced by a vote chosen by the attacker. This is a more general form of coercion which is not addressed even in a strong system like JCJ/Civitas, because there is no way for the voter to find out that the voting credentials have been compromised. Some variants of JCJ/Civitas [CH11, AFT10, AT13] aim to improve the usability of voter strategies for achieving coercion-resistance, but they do not consider the problem of silent coercion explicitly. In Helios as well, if the credentials of a voter have been leaked, they can be used to cast a vote on voter's behalf.

5.1.2 Coercion-evidence

This chapter takes as its starting point the observation that it has been impossible to achieve all the requirements simultaneously. Since result verifiability and usability may be considered "non-negotiable", we consider a setting in which coercion-resistance may be relaxed. Nevertheless, to defend against coercion (both explicit coercion and silent coercion), we propose the property of *coercion-evidence*. This means that unforgeable evidence about the degree of coercion that took place is included in the election output. Authorities, observers and voters can examine this evidence and use it to determine whether the election result carries a mandate for the winning candidate. Thus, election authorities can decide to consider the election as valid or not, leading to disincentivisation of coercion.

In some situations, dishonest voters could try to disrupt the election by faking being coerced. Since an individual voter can only fake her own coercion, this strategy would be effective only if the winning margin is low. Although this doesn't prevent coercion evidence, it could lower the attractiveness of our system. We will discuss practical mitigations as well as research perspectives to address this challenge.

5.1.3 Caveat Coercitor

As an example of a system satisfying coercion-evidence, we propose Caveat Coercitor, an internet voting scheme intended to be practically deployable. By shifting away from the conventional wisdom of coercion-resistance in favour of coercion-evidence, it tries to find a "sweet spot" between security and usability. Coercion-evidence also means that the system disincentivises coercion. In Caveat Coercitor, the most the coercer can achieve is to cancel a voter's vote. This is worst for the attacker than the usual notion of forced abstention (where an attacker can force a voter to abstain), because the number of such cancelled votes is revealed as part of the election output. Since the election result will be considered valid only if those cancelled votes would not have affected it, this means that the coercer cannot significantly affect the outcome of the election.

Caveat Coercitor borrows some ideas from JCJ/Civitas, in particular the notion of private and public credentials, but at the same time it is designed such that, unlike JCJ/Civitas:

- The generation of credentials does not have to be distributed, making it simpler and more usable.
- The registration phase needs to be secured only with "best effort" confidentiality of credentials. If this security is broken, coercion may be possible, but it will be evident. Credentials can be sent by post, SMS or email. Effort should be made to keep the credentials confidential, but even if this does not succeed the core property of coercion evidence is not broken.
- This property does not rely on trustworthiness of any registrar, and does not require untappable channels during registration. We require the channel to be *resilient*: voters will receive their credentials, perhaps after multiple attempts.
- Voters can directly verify that their private credential matches the public one on the bulletin board (voters can be given the randoms used in the encryption). There is no need for zero-knowledge proofs.

- The system does not require the use of fake credentials. If a voter is coerced, she can give away her real credential and vote normally. Coercion will be evident in that case.

- The system addresses the problem of silent coercion. If a voting credential has been leaked and misused by an intruder, this will be evident to any external observer.

Our contributions. 1) We introduce the property of *coercion-evidence* and we propose a general formal definition (section 5.2). 2) We propose a new internet voting scheme, *Caveat Coercitor*, that achieves coercion-evidence (section 5.3). 3) We weaken the trust assumptions for internet voting and we discuss how coercion-evidence is useful in practice. We also hint how Caveat Coercitor could be adapted to address the problem of an untrusted voting platform (section 5.4). 4) We perform a rigorous analysis of the fact that Caveat Coercitor satisfies coercion-evidence (section 5.5).

5.2 Coercion-evidence

In this section we do not fix the model that is used to specify security protocols. The definition can be instantiated in any computational or symbolic model. We assume that the model defines the notion of a run and of a bulletin board for an e-voting system. As usual, we assume that there is an attacker (or coercer, intruder) that controls the communication network. When a message is sent to the environment, it is assumed to be in the control of the attacker.

We assume that each eligible voter V has a unique voting credential s_V. There is a registration phase where the voters can obtain their correct voting credentials. In other words, we assume a registration protocol that ensures availability and integrity of voting credentials. We do not assume that this protocol ensures the secrecy of voting credentials. In particular, the voting credentials of several voters may have been leaked during registration.

The voting phase allows a voter with voting credentials s and the intended vote ν to execute a program $\mathcal{V}(s, \nu)$, whose definition depends on the protocol. $\mathcal{V}(s, \nu)$ may allow a voter to cast one or multiple ballots, abstain or report coercion. We say that a voter V, with credential s_V and intended vote ν_V, follows the specification of the protocol if its interaction with the protocol consists in executing $\mathcal{V}(s_V, \nu_V)$.

Definition 1 (coerced, dishonest, free voters)**.** *Let τ be a run of an electronic voting system and V be a voter with credential s_V who intends to vote ν_V. We say that the voter V is:*

- coerced *(to vote for ν_C), if*
  - *a ballot with credential s_V and vote ν_C, with $\nu_C \neq \nu_V$, is present on the bulletin board in the voting phase*
  - *V follows the specification of the protocol*

  in the run τ.
- dishonest*, if V does not follow the specification of the protocol in the run τ.*
- free*, if V is not coerced in the run τ*

From our definitions, it follows that honest voters always cast a vote for their intended choice, and they do not cast a vote for a different candidate. If a voter V with voting credentials s_V is coerced (resp. dishonest) in a run τ, we let $s_V \in \delta_c(\tau)$ (resp. $s_V \in \delta_d(\tau)$) and we will sometimes say that s_V is a coerced (resp. dishonest) credential. Thus for a run τ,

- $\delta_c(\tau)$ is the set of credentials for coerced voters in τ.
- $\delta_d(\tau)$ is the set of credentials for dishonest voters in τ.

The number $|\delta_c(\tau)|$ is called the degree of coercion in the run τ.

A coerced voter is one whose credentials have been leaked and misused by an intruder. This may have happened by explicit coercion or by silent coercion. We assume that an honest voter would nevertheless obtain the voting credentials and cast a vote normally for the intended choice. We consider voters to be coerced only if their credentials have been used to cast a vote for a candidate that is different from their intended choice.

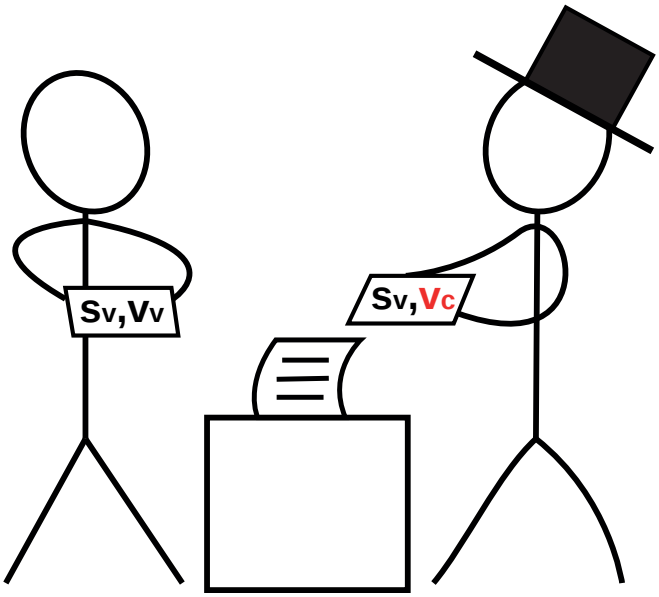

Figure 5.1: A voter with credential s_V intending to vote ν_V is coerced to vote $\nu_\mathcal{C}$ if a ballot with credential s_V and vote $\nu_\mathcal{C}$ with $\nu_V \neq \nu_\mathcal{C}$ is present on the BB; and the voter follows the specification of the protocol.

A dishonest voter can misbehave in arbitrary ways, and appear to be coerced even when she is not. It is also possible that a voter did not follow the protocol due to a genuine mistake, but for simplicity we will consider such voters dishonest. Note that a free voter can also be dishonest.

We consider an equivalence relation on runs, called *indistinguishability*, that models the inability of an observer to tell the difference between two runs that differ on some private data. For two runs τ and τ', we denote by $\tau \sim \tau'$ if they are in the indistinguishability relation (defined in section 5.5). A run τ is by definition *complete* if its corresponding bulletin board contains the outcome of the election.

Definition 2 (coercion-evidence)**.** *An electronic voting system $\mathcal{EVS}$ satisfies* coercion-evidence *if:*

1. ***Coercion-evidence test:*** *there exists a test* $\mathsf{ce}(\_)$ *that takes as input data on the bulletin board and outputs a number such that, for every complete run τ of $\mathcal{EVS}$, we have*

$$|\delta_c(\tau)| \leq \mathsf{ce}(\tau) \leq |\delta_c(\tau)| + |\delta_d(\tau)|$$

2. ***Coercer independence:*** *Let V be a voter with credential s_V who intends to vote for ν_V. Then, for every run τ of $\mathcal{EVS}$ where*

   - *the voter V follows the protocol by executing $\mathcal{V}(s_V, \nu_V)$*
   - *for every candidate ν_i, there is a free honest voter V_i that follows the protocol by executing $\mathcal{V}(s_{V_i}, \nu_i)$*

   there exists a run τ', where V does not execute $\mathcal{V}(s_V, \nu_V)$ but rather follows the coercer's instructions, such that $\tau \sim \tau'$.

The first part of coercion-evidence requires the existence of a test that can be applied to data that is available on the bulletin board in order to determine the degree of coercion in a given run. Note however that we do not require the test to return the exact degree of coercion, but allow for an over-approximation. Indeed, for an external party, there is no way of telling the difference between a coerced voter and a dishonest voter who simulates being coerced. That is why we have to allow an upper bound of $|\delta_c(\tau)| + |\delta_d(\tau)|$.

The second part of coercion-evidence requires that the coercer can not make a distinction between a run τ where the voter has followed the protocol normally and has cast a vote for the desired candidate, in spite of being coerced, and a run τ' where the voter has followed the coercer's instructions. So a voting system may require that for each coerced voter there exists a free honest voter, so that in each case there is at least one vote for the

coerced choice, otherwise the coercer would trivially detect that the coerced voter did not obey the instructions.

Note that our definition also counters vote buying and achieves effective receipt-freeness. A receipt for a ballot with valid credentials and a certain candidate, even if that ballot is cast, is not useful to a coercer because it does not guarantee that the vote will be counted.

One might imagine a system that allows voters to declare that they are being coerced, perhaps with a tick-box on the ballot form, or a separate channel by which to report coercion later.

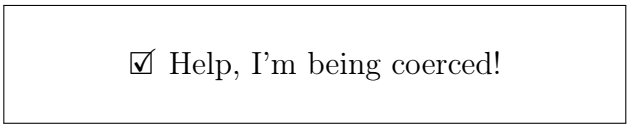

Figure 5.2: Straw-man proposal for coercion evidence

This may work if voters know they are being coerced. But such a system does not satisfy coercion-evidence according to our definition, since our definition requires evidence even in the case of silent coercion.

5.3 Caveat Coercitor

Caveat Coercitor is inspired from JCJ/Civitas [JCJ05, CCM08]. JCJ/Civitas is described in Chapter 2 in Section 2.3.1.1, here we describe how Caveat Coercitor deviates from it.

5.3.1 Caveat Coercitor: registration, voting and mixing

Caveat Coercitor follows the same scheme as JCJ/Civitas (c.f. Chapter 2 in Section 2.3.1.1). It has a set R of registrars, set T of tellers, set $V_1, \ldots, V_n$ of eligible voters, a re-encryption mix net Mix, and a bulletin board BB. Here we describe it and point out how it deviates from JCJ/Civitas.

Trust assumptions: We assume that one mix server in Mix, one teller in T and the voting platforms are honest. However, to achieve coercion-evidence, Caveat Coercitor does not have to assume that any of the registrars are trusted, and neither does it require any untappable channel between voters and registrars. The registration channel is assumed only to allow the voters to obtain their credentials, and it may allow the coercer to read honest messages and inject fake messages. We discuss our trust assumptions in more detail in section 5.4.

Initialisation: The initialisation phase and the structure of the voting ballots are exactly the same as in JCJ/Civitas. The tellers T have the public key pk_T and private key sk_T. For a given private credential s, the corresponding public credential $\mathsf{enc}(s; pk_T; r)$ is published on the electoral roll (r is a fresh random number in $\mathbb{Z}_q^*$).

Registration: The voters can simply receive their private credentials s by email or by post. Their creation does not have to be distributed (Recall in JCJ/Civitas, voter V runs a separate protocol with each of the registrars and obtains a private share s_V^i. The private voting credential, in JCJ/Civitas, is the sum of all private credential shares, i.e. $s_V = s_V^1 + \ldots + s_V^m$).

Validity of voter credentials: Because the voter is not required to keep the private credential secret at all cost, the proofs that the public part $\mathsf{enc}(s; pk_T; r)$ of the credential is a correct encryption of the private part s can be significantly simplified. In fact, the voter can simply obtain from the registrar the random number r that has been used in the encryption algorithm. Then, the voter can use any platform to verify that the encryption has been correctly performed.

Resisting coercion: In Caveat Coercitor voters do not have to "resist" coercion. To make coercion "evident" a voter does not need to perform any additional task other than to cast

a vote for the desired candidate. She can give away the voting credential to the coercer, if asked.

Voting: The construction of voting ballots is similar to the one used in JCJ/Civitas. The ballot $(\mathsf{enc}(s_V; pk_T; r_s), \mathsf{enc}(\nu; pk_T; r_\nu), \mathsf{P}_{\mathsf{s}\nu}, \mathsf{P}_{\mathsf{corr}})$, from the voter V, contains the encryption of the private credential s_V, the encryption of the intended vote ν, and zero-knowledge proofs $\mathsf{P}_{\mathsf{s}\nu}$ and $\mathsf{P}_{\mathsf{corr}}$ as used in JCJ/Civitas (recall that the zero-knowledge proof $\mathsf{P}_{\mathsf{s}\nu}$ proves that its creator knows both s_V and ν, and zero-knowledge proof $\mathsf{P}_{\mathsf{corr}}$ proves that ν is a valid vote, according to the specification decided by election authorities.). The fundamental difference in this phase as compared to in JCJ/Civitas is how Caveat Coercitor handles multiple ballots from the same voter: a voter may submit multiple ballots, constructed on possibly different voting devices, provided they are all for the same candidate. At most one vote for one candidate will be counted for each credential. Precisely one vote will be counted if all the ballots for a given credential contain a vote for the same candidate. No votes will be counted for a credential if there are ballots for that credential corresponding to at least two distinct candidates. Instead, that credential will be detected by the coercion-evidence test.

Verification of proofs and mixing: The zero-knowledge proofs of cast ballots are verified and the ballots with invalid proofs are eliminated. The valid ballots (without the proofs) and the electoral roll are then sent to the re-encryption mix net Mix for anonymization.

5.3.2 Caveat Coercitor: coercion-evidence and tallying

We have at this stage a set of anonymized ballots to be tallied and an anonymized electoral roll. In addition to eliminating ballots with fake credentials, now we need to determine the coerced credentials, remove the corresponding ballots from the ballots to be tallied and output evidence of coercion.

Coercion-evidence: Let τ be a run of Caveat Coercitor up to this phase. We let

- $BB_{\mathsf{aer}}(\tau)$ be the anonymized electoral roll with each element of the form $\mathsf{enc}(s; pk_T; r)$, for some private credential s. $\mathsf{enc}(s; pk_T; r)$ are re-encryptions of public credentials and we sometimes denote them by es.

- $BB_{\mathsf{cast}}(\tau)$ be the set of cast ballots with valid zero-knowledge proofs of correctness. Each element of $BB_{\mathsf{cast}}(\tau)$ is of the form $(\mathsf{enc}(s; pk_T; r1), \mathsf{enc}(\nu; pk_T; r1'), \mathsf{P}_{\mathsf{s}\nu}, \mathsf{P}_{\mathsf{corr}})$, for some private credential s and vote ν. The encrypted votes are sometimes denoted by $\mathsf{e}\nu$.

- $BB_{\mathsf{tally}}(\tau)$ be the set of anonymized ballots to be authorized relying on $BB_{\mathsf{aer}}(\tau)$ and then tallied. Each element of $BB_{\mathsf{tally}}(\tau)$ is of the form $(\mathsf{enc}(s; pk_T; r2), \mathsf{enc}(v; pk_T; r2'))$, for some private credential s and vote ν.

The tellers will execute an algorithm whose output can be used by any external observer to determine the amount of coercion, while at the same time allowing coercer-independence. Specifically, the tellers will compute the set of credentials in $BB_{\mathsf{aer}}(\tau)$ for which there are at least two ballots with two different votes in $BB_{\mathsf{tally}}(\tau)$, i.e. the set

$$BB_{\mathsf{ce}}(\tau) = \{\mathsf{es} \in BB_{\mathsf{aer}}(\tau) |\ \exists (\mathsf{es}_1, \mathsf{e}\nu_1), (\mathsf{es}_2, \mathsf{e}\nu_2) \in BB_{\mathsf{tally}}(\tau).$$
$$\mathsf{pet}(\mathsf{es}, \mathsf{es}_1) = \mathsf{pet}(\mathsf{es}, \mathsf{es}_2) = \mathsf{ok}, \mathsf{pet}(\mathsf{e}\nu_1, \mathsf{e}\nu_2) \neq \mathsf{ok}\}$$

There is a simple way to compute $BB_{\mathsf{ce}}(\tau)$, shown in Algorithm 1.

The idea is to first group together all the ballots that correspond to a given credential, relying on plaintext equivalence tests, and then detect if among these there are two ballots representing distinct votes. At the end of algorithm 1, the set $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau)$ should be equal to $BB_{\mathsf{ce}}(\tau)$ (we will prove this in section 5.5) and the zero-knowledge proofs in $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau)$ should allow anyone to verify that the set $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau)$ has been correctly computed.

Now, the *coercion-evidence test* $\mathsf{ce}_{\mathsf{cc}}(\tau)$ in Caveat Coercitor, described in Algorithm 2, consists in verifying that $\mathsf{cc}_{\mathsf{ce}}^{\mathsf{cred}}(\tau)$ has been correctly computed (basically, this means

Algorithm 1 Coercion-evidence: algorithm that reveals too much

Input: $BB_{\mathsf{aer}}(\tau), BB_{\mathsf{tally}}(\tau)$ (taken from the bulletin board)
Output: $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau), \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau)$
$\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau) := \emptyset; \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) := \emptyset$
for $\mathsf{es} \in BB_{\mathsf{aer}}(\tau)$ **do**
  $\mathcal{B}_{\mathsf{es}} := \emptyset$ // $\mathcal{B}_{\mathsf{es}}$ is the set of ballots corresponding to es
  for $(\mathsf{es}', \mathsf{e}\nu) \in BB_{\mathsf{tally}}(\tau)$ **do**
    if $\mathsf{pet}(\mathsf{es}, \mathsf{es}') = \mathsf{ok}$ **then**
      $\mathcal{B}_{\mathsf{es}} := \mathcal{B}_{\mathsf{es}} \cup \{(\mathsf{es}', \mathsf{e}\nu)\}$
      $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) := \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) \cup \mathsf{petproof}(\mathsf{es}, \mathsf{es}', \mathsf{yes})$
  for $(\mathsf{es}_1, \mathsf{e}\nu_1) \in \mathcal{B}_{\mathsf{es}}, (\mathsf{es}_2, \mathsf{e}\nu_2) \in \mathcal{B}_{\mathsf{es}}$ **do**
    if $\mathsf{pet}(\mathsf{e}\nu_1, \mathsf{e}\nu_2) \neq \mathsf{ok}$ **then**
      $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau) := \mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau) \cup \{\mathsf{es}\}$
      $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) := \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) \cup \mathsf{petproof}(\mathsf{e}\nu_1, \mathsf{e}\nu_2, \mathsf{no})$
return $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau), \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau)$ (to the bulletin board)

verifying the zero-knowledge proofs for the plaintext equivalence tests) and outputting $|\mathsf{cc}_{\mathsf{ce}}^{\mathsf{cred}}(\tau)|$ as the observed degree of coercion.

Algorithm 2 Coercion-evidence test

Input: $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau), \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau)$ (taken from the bulletin board)
Output: $|\mathsf{ce}_{\mathsf{cc}}(\tau)|$
for $\mathsf{zkp} \in \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau)$ **do**
  if $\mathsf{verify}\ (\mathsf{zkp}) \neq ok$ **then**
    fail
$\mathsf{ce}_{\mathsf{cc}}(\tau) := \mathsf{count}(\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau))$
return $\mathsf{ce}_{\mathsf{cc}}(\tau)$ (to the bulletin board)

The algorithm 1 provides data for the coercion-evidence test and allows coercer independence, but let us see at what cost coercer independence is achieved. Indeed, note that the number of ballots that correspond to every credential in $BB_{\mathsf{aer}}(\tau)$ is revealed. Now, assume the following coercion strategy: the coercer instructs the voter V to abstain and then casts a very large number n_V of ballots using s_V. Therefore, if the voter disobeyed and has cast a vote using s_V, the output of the algorithm 1 will allow the coercer to observe that for some credential $n_V + 1$ ballots have been cast. Now, to obtain coercer-independence, there should be some free voter V' that has cast n_V ballots using s'_V: the coercer will see n_V ballots for some credential, and will not be able to tell whether that credential is s_V or

not.

Although in theory coercer independence can be achieved in this way, the practical aspect of this approach is questionable. Therefore, we propose another algorithm to compute $BB_{\mathsf{ce}}(\tau)$, which outputs less information about the set of tallied ballots (Algorithm 3). Instead of computing the set of all ballots that have been cast with every credential s_V, we carefully guide the computation to determine if there are two distinct votes for s_V. In particular, if there are multiple ballots with credential s_V for the same candidate, only one of them will be determined. We do the minimal amount of computations required to determine if a credential is coerced or not, and then go to the next credential.

Algorithm 3 Coercion-evidence: algorithm that reveals enough

Input: $BB_{\mathsf{aer}}(\tau), BB_{\mathsf{tally}}(\tau), \mathsf{can}_1 \ldots \mathsf{can}_l$
(taken from the bulletin board)
Output: $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau), \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau)$
$\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau) := \emptyset; \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) := \emptyset$
// Step 1: group ballots according to the vote that they encode
for $i := 1$ to l **do**
  $\mathcal{B}_i := \emptyset$
  for $(\mathsf{es}, \mathsf{e}\nu) \in BB_{\mathsf{tally}}(\tau)$ **do**
    if $\mathsf{pet}(\mathsf{can}_i, \mathsf{e}\nu) = \mathsf{ok}$ **then**
      $\mathcal{B}_i := \mathcal{B}_i \cup \{(\mathsf{es}, \mathsf{e}\nu)\}$
      $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) := \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) \cup \mathsf{petproof}(\mathsf{can}_i, \mathsf{e}\nu, \mathsf{yes})$
// Step 2: for each es, check if it occurs in $\mathcal{B}_i, \mathcal{B}_j$, $i \neq j$
for $\mathsf{es} \in BB_{\mathsf{aer}}(\tau)$ (***) **do**
  for $i := 1$ to l **do**
    for $(\mathsf{es}_1, \mathsf{e}\nu_1) \in \mathcal{B}_i$ **do**
      if $\mathsf{pet}(\mathsf{es}, \mathsf{es}_1) = \mathsf{ok}$ **then**
        $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) := \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) \cup \mathsf{petproof}(\mathsf{es}, \mathsf{es}_1, \mathsf{yes})$
        for $j := i + 1$ to l **do**
          for $(\mathsf{es}_2, \mathsf{e}\nu_2) \in \mathcal{B}_j$ **do**
            if $\mathsf{pet}(\mathsf{es}, \mathsf{es}_2) = \mathsf{ok}$ **then**
              $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau) := \mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau) \cup \{es\}$
              $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) := \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau) \cup \mathsf{petproof}(\mathsf{es}, \mathsf{es}_2, \mathsf{yes})$
              go to next credential in $BB_{\mathsf{aer}}(\tau)$ at (***)
return $\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau), \mathsf{ce}_{\mathsf{cc}}^{\mathsf{zkp}}(\tau)$ (to the bulletin board)

We assume there are l candidates and that a list $\mathsf{can}_1, \ldots, \mathsf{can}_l$ of candidate names encrypted with pk_T has been computed and passed through a mixnet. These encrypted

candidate names are part of the input for Algorithm 3. The algorithm first groups the ballots according to the candidate that they represent and then for every credential it checks whether it belongs to at least two groups.

To achieve coercer independence for a voter V in the context of Algorithm 3, all we need is the presence of a free voter V' that can either cast a normal vote or cancel her ballots by casting two different votes: the coercer can not tell the difference between the ballots of V and those of V'. If the coercer has cast a large number of ballots n_V with s_V and the voter V has cast a different vote with s_V, the free voter V' just needs to cast a vote for the candidate desired by the coercer: there is no way for the coercer to detect that there are n_V+1 ballots with credential s_V and that they have been discarded as coercion-evidence.

Tallying: For a ciphertext c and a set of ciphertexts M, we will denote by $c \in_{\mathsf{pet}} M$ the fact $\exists c' \in M.\mathsf{pet}(c, c') = \mathsf{ok}$. Let $BB_{\mathsf{valid}}(\tau) = BB_{\mathsf{aer}}(\tau) \setminus BB_{\mathsf{ce}}(\tau)$ be the set of anonymized credentials that are not coerced. Now, for every $(\mathsf{es}, \mathsf{e}\nu) \in BB_{\mathsf{tally}}(\tau)$,

- either $\mathsf{es} \in_{\mathsf{pet}} BB_{\mathsf{valid}}(\tau)$. In this case, since $\mathsf{es} \notin_{\mathsf{pet}} BB_{\mathsf{ce}}(\tau)$, we are certain that all ballots that correspond to es contain a vote for the same candidate. Only one of them should be counted.

- or else $\mathsf{es} \in_{\mathsf{pet}} BB_{\mathsf{ce}}(\tau)$. In this case, es corresponds to a coerced credential and no ballot should be counted for es.

- or else $\mathsf{es} \notin_{\mathsf{pet}} BB_{\mathsf{aer}}(\tau)$. In this case, es corresponds to a non-eligible credential and no ballot should be counted for es

Therefore, we have the following simple algorithm for tallying: for all credential $\mathsf{es} \in BB_{\mathsf{valid}}(\tau)$, find the first ballot $(\mathsf{es}', \mathsf{e}\nu) \in BB_{\mathsf{tally}}(\tau)$ such that $\mathsf{pet}(\mathsf{es}, \mathsf{es}') = \mathsf{ok}$; then, decrypt $\mathsf{e}\nu$.

5.4 Coercion-evidence in Caveat Coercitor

5.4.1 Trust assumptions

We will show in section 5.5 that coercion-evidence holds in caveat coercitor under the following trust assumptions:

- The voting platforms of coerced voters and of some free voters are not compromised.
- The registration channel allows voters to obtain their voting credentials, and voters verify them to ensure that they are correct. In particular, this implies that voters can cast a valid ballot for their intended choice.
- There is an anonymous channel that allow voters to cast their vote as intended.
- There exists at least one honest mix server Mix
- There exists at least one honest teller T

The trust assumptions about one honest mix server and one honest teller are standard and probably necessary in any system based on mixnets and on public-key cryptography. We explain later in this section how Caveat Coercitor may be adapted to relax the first assumption about the trusted voting platform.

A big novelty of our trust assumptions, in contrast with Helios and JCJ/Civitas, is that voting credentials need to be secured only with "best effort". Thus, even if the voting channel where the voter obtains her credential is compromised, or if the voter is coerced to reveal the private credential, coercion-evidence still holds. (However, if the security is insufficient, a lot of coercion will be detected.) This also means we do not have to trust the registrars to keep the credential secret. More generally, this addresses the problem of silent coercion discussed in the introduction of this chapter, which is common to Helios, JCJ/Civitas and the Estonian internet voting system.

Informally, coercion-evidence holds given these trust assumptions because:

- The coerced voters follow the instruction to cast a vote for their intended candidate. Since they have access to a trusted voting platform, this ensures that, for all coerced voters, coercion-evidence is input in the system during the voting phase. It will be detected later.

- Additionally, a trusted voting platform means that the coercer can not observe that a voter has cast a ballot, and this is essential for coercer-independence.

- The voter verifies that her ballot has reached the bulletin board and zero-knowledge proofs for mixnets and plaintext equivalence tests are publicly checked. This ensures that coercion-evidence that is input in the system during the voting phase is not lost on the way from the voters to the tellers. The role of zero-knowledge proofs is also to ensure that coercion-evidence in ballots to be tallied corresponds indeed to ballots cast in the voting phase, and not introduced in the system during mixing.

- Voters receive and verify that their credentials are valid. This ensures that their ballots are not discarded before the final tally, and will result either in a counted vote, or in coercion-evidence.

- The honest mixer and the honest teller will help in achieving coercer-independence: the honest mixer will ensure that the ballots cast by the coercer or the coerced voter can not be tracked to determine how they have been handled in the tallying phase; the honest teller ensures that the ballots are decrypted and plaintext equivalence tests are performed only as specified by the protocol.

5.4.2 Example

Suppose that events unfold as depicted in Table 5.1. Among 48M voters, about 44M voters submit ballots with vote for only one candidate, while about 4.2M voters (or their coercers) submit ballots containing votes for different candidates. Thus, most voters have not been coerced, and have cast possibly multiple ballots for a single candidate. They account for

	No. of credentials	%age of credentials
Total number of eligible credentials	48,783,530	100%
Number of credentials that have voted for a single candidate	44,539,363	91.3%
Number of credentials that have voted for at least two candidates	4,244,167	8.7%

Table 5.1: An example distribution of ballots. A table in this format is output by the system at the end of the election.

91.3% of voters. The rest 8.7% of voters have cast ballots for different candidates. These may be voters that have been forced to release their credentials, forced to cast vote to a candidate but they also managed to cast vote to their intended candidate or they may simply have decided to submit two ballots for different candidates. So either they were coerced or they did not follow the specifications of the protocol and thus were dishonest. The ballots from these 8.7% of voters will all be discarded and 91.3% of the ballots are taken to determine the outcome. The system counts one vote for the (single) candidate represented by each of the set of ballots that corresponds to each credential. The margins by which the winner beats the other candidates can be considered in order to determine whether the election result should carry.

To understand the reason for this way of counting, let us distinguish the following two cases:

- If a credential has been used once or more times, but each time to vote for the same candidate, a vote for that candidate is counted. Indeed, there are three subcases:
  - The voter has cast multiple ballots for the same candidate;
  - The attacker obtained voter's credentials and has cast a ballot for the same candidate as the voter; or
  - The voter knowingly abstained and the attacker obtained her credentials and cast ballots for one candidate.

The output of the system does not allow any voter, coercer or observer to distinguish between these sub-cases, but if the voter behaved correctly and cast a ballot (the third case is eliminated), then her vote is counted.

- If a credential has been used to cast votes for multiple candidates, none of its votes is counted. The following sub-cases are indistinguishable:
  - The voter has cast multiple ballots for several different candidates;
  - The voter has cast multiple ballots for one candidate, the attacker obtained her credentials and has cast a vote for a different candidate;
  - The voter and the attacker each have cast votes for several different candidates; or
  - The voter knowingly abstained and the attacker cast votes on her behalf for multiple different candidates.

In each of these cases, either the voter is dishonest or the voter is coerced. The corresponding votes are therefore not counted and the evidence is recorded for the authorities to make a decision.

5.4.3 Discussion

Disincentivisation of coercion: As demonstrated, a coercer that wishes to achieve a particular outcome is faced with a dilemma. Assuming the security of the cryptography, and assuming that the voter is not dissuaded from casting her own ballot by invalid threats, the best the coercer can do is try to force a large number of annulled votes. However, the number of annulled votes he forces will be detected and announced, and an analysis will be performed to check whether those votes will materially affect the outcome. If not, the election will be considered a success and the declared outcome will have been shown to be robust against the degree of coercion attempted.

Since annulled votes in Caveat Coercitor are evident, they do not have the power that forced abstentions have in other systems. If the coercer has a strategy of annulling votes that he believes would be votes for a particular candidate, it won't work, because the final results will be interpreted as meaning 'there was a lot of coercion' rather than 'there were a lot of abstentions'.

Disruption of the election. In the definition of coercion-evidence, we have taken into account the fact that dishonest voters may pretend to be coerced, and therefore the coercion-evidence test may overestimate the actual degree of coercion. In some situations, a set of dishonest voters could rely on this in order to challenge the validity of the election. Especially when the difference between the winner and the runner-up of the election is small, a minority of voters would suffice to cause a disruption.

To deter voters from disrupting the election in practice, one option could be making voluntarily voting for two different candidates an offence. Both technical and administrative measures should be carefully designed to find an acceptable balance between discouraging dishonest behaviour and encouraging resistance to coercion.

Another option is to have a probabilistic interpretation of coercion-evidence, depending on various data that can be gathered about the election (voter intentions, audit logs, etc.). This approach would allow to separate coercion-evidence into different threads, one of which would be evidence of actual coercion. More research on this idea is needed.

Individual and universal verifiability: Individuals can verify that their ballot is present on the bulletin board, and will be counted. If the voter was coerced, their ballot will be included in the coercion evidence test. Observers can perform universal and eligibility verification, because the computations of algorithm 3 are publicly verifiable. Additionally, observers can verify the figures given in the table.

Towards untrusted voting platforms: Although in this chapter we assume, for

simplicity, that a voter has access to a trusted voting platform, Caveat Coercitor has the potential to be adapted to provide coercion evidence in the context of untrusted platforms. For instance, assume a voter has access to n voting platforms and the voter is unsure which one can be trusted for integrity. That is not a problem for Caveat Coercitor: the instruction for the voter is to simply cast her vote on any number of available voting platforms. As long as *one out of n* platform is not integrity-compromised (the voter does not need to know which one):

- either the vote will be counted for the preferred candidate;
- or coercion will be recorded.

This deals with integrity. For voting platforms that are privacy-corrupted, the situation is more difficult. A voter needs to take whatever steps are necessary to prevent the voting platform communicating with the coercer. This may involve using the voting platform in a Faraday cage, and resetting it or even destroying it afterwards, depending on the coercer's power.

Systems like JCJ/Civitas and Helios are not as well adapted to the untrusted platform, because they are not designed to be as tolerant of coercion as Caveat Coercer. Our system allows coercion (but makes it evident).

5.5 Caveat Coercitor satisfies coercion-evidence

According to definition 2, we prove that the coercion-evidence test of Caveat Coercitor is adequate (section 5.5.1) and that Caveat Coercitor allows coercer independence (section 5.5.2). For coercer independence we give proof sketches in this section and complete proofs in appendix B.

5.5.1 Coercion-evidence test

If $\mathsf{ce}_{\mathsf{cc}}(\tau)$ is the result of applying the coercion-evidence test in Caveat Coercitor for a run τ, we have to show that $|\delta_c(\tau)| \leq \mathsf{ce}_{\mathsf{cc}}(\tau) \leq |\delta_c(\tau)| + |\delta_d(\tau)|$.

For a credential s and a vote ν, we will denote by $\mathcal{B}(s,\nu)$ the set of possible ballots (including the zero-knowledge proofs) with credential s and vote ν. Thus, $\mathcal{B}(s,\nu)$ is the set of tuples of the form $(\mathsf{enc}(s;pk_T;r), \mathsf{enc}(\nu;pk_T;r'), \mathsf{P}_{s\nu}, \mathsf{P}_{\mathsf{corr}})$, where r, r' are random numbers and $\mathsf{P}_{s\nu}$, $\mathsf{P}_{\mathsf{corr}}$ are the corresponding zero-knowledge proofs for this ballot. We denote by $\mathcal{B}_0(s,\nu)$ the set of possible ballots with credential s and vote ν, without the zero-knowledge proofs.

For a run τ of Caveat Coercitor, recall that $BB_{\mathsf{cast}}(\tau)$ is the set of cast ballots with valid zero-knowledge proofs and $BB_{\mathsf{tally}}(\tau)$ is the set of ballots to be tallied in the run τ. Let us denote by $\mathcal{E}(\tau)$ the set of private credentials for eligible voters in a run τ. We define

$$\begin{aligned}
\mathcal{I}_{\mathsf{cast}}(\tau) = \quad & \{s \mid \exists \nu, \nu'.\nu \neq \nu', \mathcal{B}(s,\nu) \cap BB_{\mathsf{cast}}(\tau) \neq \emptyset \\
& \ \& \ \mathcal{B}(s,\nu') \cap BB_{\mathsf{cast}}(\tau) \neq \emptyset\} \\
\mathcal{I}_{\mathsf{tally}}(\tau) = \quad & \{s \mid \exists \nu, \nu'.\nu \neq \nu', \mathcal{B}_0(s,\nu) \cap BB_{\mathsf{tally}}(\tau) \neq \emptyset \\
& \ \& \ \mathcal{B}_0(s,\nu') \cap BB_{\mathsf{tally}}(\tau) \neq \emptyset\} \\
\mathcal{F}_{\mathsf{cast}}(\tau) = \quad & \{s \mid \exists \nu. \mathcal{B}(s,\nu) \cap BB_{\mathsf{cast}}(\tau) \neq \emptyset\} \setminus \mathcal{E}(\tau) \\
\mathcal{F}_{\mathsf{tally}}(\tau) = \quad & \{s \mid \exists \nu. \mathcal{B}_0(s,\nu) \cap BB_{\mathsf{tally}}(\tau) \neq \emptyset\} \setminus \mathcal{E}(\tau)
\end{aligned}$$

In words, $\mathcal{I}_{\mathsf{cast}}(\tau)$ and respectively $\mathcal{I}_{\mathsf{tally}}(\tau)$ represent the set of (inconsistent) credentials that have at least two corresponding ballots with different votes on the bulletin board in the voting phase and in the tallying phase respectively. The set $\mathcal{F}_{\mathsf{cast}}(\tau)$ represents fake (i.e. invalid) credentials that have been used to cast a vote, while the set $\mathcal{F}_{\mathsf{tally}}(\tau)$ is formed of fake credentials that correspond to ballots that are to be tallied.

A first lemma shows that all the coerced credentials are contained in the set $\mathcal{I}_{\mathsf{cast}}(\tau)$ and that, additionally, the set $\mathcal{I}_{\mathsf{cast}}(\tau)$ may only contain fake or dishonest credentials:

Lemma 1. *For all run τ of Caveat Coercitor, we have*

$$\delta_c(\tau) \subseteq \mathcal{I}_{\mathsf{cast}}(\tau) \subseteq \delta_c(\tau) \cup \delta_d(\tau) \cup \mathcal{F}_{\mathsf{cast}}(\tau)$$

The proof of the first inclusion relies on the definition of coerced voters and on the trust assumption that the coerced voters have access to an honest platform to cast their vote. Thus, at least two different votes are present for each coerced credential on the bulletin board: one coming from the voter and another one coming from the coercer. The second inclusion can be shown by a simple case analysis.

Proof. First we prove that $\delta_c(\tau) \subseteq \mathcal{I}_{\mathsf{cast}}(\tau)$. Let $s \in \delta_c(\tau)$ be the credential of coerced voter V in a run τ. By definition of coerced voters and since by our assumptions V has access to a trusted voting device, there exist two ballots in $BB_{\mathsf{cast}}(\tau)$ corresponding to s and two different votes ν, ν': $\mathcal{B}(s, \nu) \cap BB_{\mathsf{cast}}(\tau)$, $\mathcal{B}(s, \nu') \cap BB_{\mathsf{cast}}(\tau)$ and $\nu \neq \nu'$. Therefore, we have $s \in \mathcal{I}_{\mathsf{cast}}(\tau)$.

Let us now prove that $\mathcal{I}_{\mathsf{cast}}(\tau) \subseteq \delta_c(\tau) \cup \delta_d(\tau) \cup \mathcal{F}_{\mathsf{cast}}(\tau)$. Let $s \in \mathcal{I}_{\mathsf{cast}}(\tau)$ and assume that $s \notin \mathcal{F}_{\mathsf{cast}}(\tau)$. We show that $s \in \delta_c(\tau) \cup \delta_d(\tau)$. By definition of $\mathcal{I}_{\mathsf{cast}}(\tau)$, there exist $\nu_1, \ldots, \nu_n$, such that $n > 1$, $\nu_i \neq \nu_j$ and $\mathcal{B}(s, \nu_i) \cap BB_{\mathsf{cast}}(\tau) \neq \emptyset$, for all $1 \leq i, j \leq n$. By definition of coerced and dishonest voters, we have

- if all ballots that are cast by V belong to $\mathcal{B}(s, \nu_i)$, for a unique i, $1 \leq i \leq n$, then V is coerced and $s \in \delta_c(\tau)$.

- otherwise, either V has cast two different votes or V did not cast a vote, and we have $s \in \delta_d(\tau)$.

Hence we can conclude that $s \in \delta_c(\tau) \cup \delta_d(\tau) \cup \mathcal{F}_{\mathsf{cast}}(\tau)$. □

Secondly, we show that the set of fake credentials and the set of inconsistent credentials can not be changed in the mixnet.

Lemma 2. *For all run τ of Caveat Coercitor in which the mixnet proofs are valid, we have*

$$\mathcal{I}_{\mathsf{tally}}(\tau) = \mathcal{I}_{\mathsf{cast}}(\tau) \text{ and } \mathcal{F}_{\mathsf{tally}}(\tau) = \mathcal{F}_{\mathsf{cast}}(\tau)$$

The proof is an easy consequence of the fact that verifiable mixnets can not modify the content of ballots.

Proof. From definitions, we observe that to conclude the lemma it is sufficient to show that for all credential s and all vote ν, we have $|\mathcal{B}(s,\nu) \cap BB_{\mathsf{cast}}(\tau)| = |\mathcal{B}_0(s,\nu) \cap BB_{\mathsf{tally}}(\tau)|$. In words, this means that no ballots have been lost or added to the tally during the mixing phase. This follows immediately from the validity of mixnet proofs.

□

Another easy observation is that the set $BB_{\mathsf{ce}}(\tau)$, defined in section 5.3.2, has the same cardinality as the set of inconsistent credentials that are not fake.

Lemma 3. *For all run τ of Caveat Coercitor, we have*

$$|BB_{\mathsf{ce}}(\tau)| = |\mathcal{I}_{\mathsf{tally}}(\tau) \setminus \mathcal{F}_{\mathsf{tally}}(\tau)|$$

Proof. Let $BB^{priv}_{\mathsf{ce}}(\tau)$ be the set of private credentials that corresponds to the public credentials in $BB_{\mathsf{ce}}(\tau)$. We note that $|BB^{priv}_{\mathsf{ce}}(\tau)| = |BB_{\mathsf{ce}}(\tau)|$ and therefore it is sufficient to show that $BB^{priv}_{\mathsf{ce}}(\tau) = \mathcal{I}_{\mathsf{tally}}(\tau) \setminus \mathcal{F}_{\mathsf{tally}}(\tau)$.

We have $s \in BB^{priv}_{\mathsf{ce}}(\tau) \Leftrightarrow \exists es \in BB_{\mathsf{ce}}(\tau)$, es is the public part of s

$\exists \{s\}^r_{pk} \in BB_{\mathsf{aer}}(\tau)$ and $(\{s\}^{r_1}_{pk}, e\nu_1), (\{s\}^{r_2}_{pk}, e\nu_2) \in BB_{\mathsf{tally}}(\tau)$ such that $\mathsf{pet}(e\nu_1, e\nu_2) \neq \mathsf{ok} \Leftrightarrow s \in \mathcal{E}(\tau)$ and $\exists \nu_1 \neq \nu_2$ such that $\mathcal{B}_0(s,\nu_1) \cap BB_{\mathsf{tally}}(\tau) \neq \emptyset$ and $\mathcal{B}_0(s,\nu_2) \cap BB_{\mathsf{tally}}(\tau) \neq \emptyset \Leftrightarrow s \in \mathcal{I}_{\mathsf{tally}}(\tau) \setminus \mathcal{F}(\tau)$. □

Let $\mathsf{ce}^{\mathsf{cred}}_{\mathsf{cc}}(\tau)$ be the set of credentials output by the Algorithm 3 for a run τ.

Finally, we show that the coercion-evidence test provided by Caveat Coercitor captures exactly $|BB_{\mathsf{ce}}(\tau)|$, i.e. we show that the Algorithm 3 from section 5.3.2 is correct.

Lemma 4. *For all run τ of Caveat Coercitor, we have*

$$\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau) = BB_{\mathsf{ce}}(\tau)$$

To prove this lemma, we show that every credential in $BB_{\mathsf{ce}}(\tau)$ will be detected in the step 2 of the Algorithm 3 and added to the set $\mathsf{ce}_{\mathsf{cc}}^{\tau}$. Moreover, we observe that every credential that has been used to cast a vote for a single candidate will not be part of $\mathsf{ce}_{\mathsf{cc}}(\tau)$, and therefore $\mathsf{ce}_{\mathsf{cc}}(\tau) \subseteq BB_{\mathsf{ce}}(\tau)$.

Proof. We prove first that $BB_{\mathsf{ce}}(\tau) \subseteq \mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau)$. If $es^0 \in BB_{\mathsf{ce}}(\tau)$, then there exist $(es_1^0, e\nu_1^0), (es_2^0, e\nu_2^0) \in BB_{\mathsf{tally}}(\tau)$ such that $\mathsf{pet}(es^0, es_1^0) = \mathsf{pet}(es^0, es_2^0) = \mathsf{ok}, \mathsf{pet}(e\nu_1^0, e\nu_2^0) \neq \mathsf{ok}$. As $\mathsf{pet}(e\nu_1^0, e\nu_2^0)) \neq \mathsf{ok}$ there exist $\mathsf{can}_i, \mathsf{can}_j$ with $1 \leq i, j \leq$ such that $\mathsf{pet}(\mathsf{can}_i, e\nu_1^0) = \mathsf{ok}$ and $\mathsf{pet}(\mathsf{can}_j, e\nu_2^0) = \mathsf{ok}$. After the Step 1 of the Algorithm 3, we have $(es_1^0, e\nu_1^0) \in \mathcal{B}_i$ and $(es_2^0, e\nu_2^0) \in \mathcal{B}_j$. Without loss of generality, let us suppose that $i < j$ and that i, j are the smallest indices with these properties.

Therefore, during the step 2 of the Algorithm 3, when es^0 is considered for the role of es, $(es_1^0, e\nu_1^0) \in \mathcal{B}_i$ and $(es_2^0, e\nu_2^0) \in \mathcal{B}_j$ can be considered for the role of $(es_1, e\nu_1)$ and $(es_2, e\nu_2)$. Thus, we have $es \in \mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau)$.

To prove the inverse inclusion, consider $es \in \mathsf{cc}_{\mathsf{ce}}^{\mathsf{cred}}(\tau)$. This means there exist $\mathcal{B}_i, \mathcal{B}_j$ such that there are $(es_1, e\nu_1) \in \mathcal{B}_i$, $(es_2, e\nu_2) \in \mathcal{B}_j$ and $\mathsf{pet}(\mathsf{can}_i, e\nu_1) = \mathsf{ok}$, $\mathsf{pet}(\mathsf{can}_j, e\nu_2) = \mathsf{ok}$ and $\mathsf{pet}(es_1, es) = \mathsf{ok} = \mathsf{pet}(es, es_2)$. Therefore, we can conclude $es \in BB_{\mathsf{ce}}(\tau)$. □

Recall that, by definition, we have $\mathsf{ce}_{\mathsf{cc}}(\tau) = |\mathsf{ce}_{\mathsf{cc}}^{\mathsf{cred}}(\tau)|$. Putting all the lemmas together, we obtain the first part of coercion-evidence according to definition 2. Corollary 1 shows that the output of the coercion-evidence test in Caveat Coercitor is a correct estimate of the degree of coercion in any run, up to the number of dishonest voters:

Corollary 1. *For all run τ of Caveat Coercitor, we have*

$$|\delta_c(\tau)| \leq \mathsf{ce}_{\mathsf{cc}}(\tau) \leq |\delta_c(\tau)| + |\delta_d(\tau)|$$

Proof. From lemma 1 and lemma 2, we have $\delta_c(\tau) \subseteq \mathcal{I}_{\mathsf{cast}}(\tau)$ and $\mathcal{I}_{\mathsf{cast}}(\tau) = \mathcal{I}_{\mathsf{tally}}(\tau)$. Therefore, we deduce $\delta_c(\tau) \subseteq \mathcal{I}_{\mathsf{tally}}(\tau)$. By definition, all credentials in $\delta_c(\tau)$ are valid, i.e. $\delta_c(\tau) \subseteq \mathcal{E}(\tau)$, and thus $\delta_c(\tau) \cap \mathcal{F}_{\mathsf{tally}}(\tau) = \emptyset$. Hence, we obtain $\delta_c(\tau) \subseteq \mathcal{I}_{\mathsf{tally}}(\tau) \setminus \mathcal{F}_{\mathsf{tally}}(\tau)$. Considering the cardinality of these sets, we obtain $|\delta_c(\tau)| \leq |\mathcal{I}_{\mathsf{tally}}(\tau) \setminus \mathcal{F}_{\mathsf{tally}}(\tau)| = |BB_{\mathsf{ce}}(\tau)| = |\mathsf{ce}^{\mathsf{cred}}_{\mathsf{cc}}(\tau)| = \mathsf{ce}_{\mathsf{cc}}(\tau)$, where we have used lemma 3 for the first equality and lemma 4 for the second equality. This way we get $|\delta_c(\tau)| \leq \mathsf{ce}_{\mathsf{cc}}(\tau)$.

From lemma 1, we have $\mathcal{I}_{\mathsf{cast}}(\tau) \subseteq \delta_c(\tau) \cup \delta_d(\tau) \cup \mathcal{F}_{\mathsf{cast}}(\tau)$. Using lemma 2, we get $\mathcal{I}_{\mathsf{tally}}(\tau) \subseteq \delta_c(\tau) \cup \delta_d(\tau) \cup \mathcal{F}_{\mathsf{tally}}(\tau)$. Subtracting $\mathcal{F}_{\mathsf{tally}}(\tau)$ from both sides, we have $\mathcal{I}_{\mathsf{tally}}(\tau) \setminus \mathcal{F}_{\mathsf{tally}}(\tau) \subseteq \delta_c(\tau) \cup \delta_d(\tau) \setminus \mathcal{F}_{\mathsf{tally}}(\tau)$. Furthermore, by definition we have $\delta_c(\tau) \cup \delta_d(\tau) \subseteq \mathcal{E}(\tau)$ and therefore $\delta_c(\tau) \cup \delta_d(\tau) \setminus \mathcal{F}_{\mathsf{tally}}(\tau) = \delta_c(\tau) \cup \delta_d(\tau)$. Thus, we have $\mathcal{I}_{\mathsf{tally}}(\tau) \setminus \mathcal{F}_{\mathsf{tally}}(\tau) \subseteq \delta_c(\tau) \cup \delta_d(\tau)$. Considering the cardinality of these sets and applying lemma 3 and lemma 4, we deduce $\mathsf{ce}_{\mathsf{cc}}(\tau) \leq |\delta_c(\tau)| + |\delta_d(\tau)|$ and we can conclude the proof. □

5.5.2 Coercer independence

Let V be a voter with credential s_V who intends to vote for ν_V. We have to show that, for every run τ of Caveat Coercitor where

- the voter V follows the protocol by executing $\mathcal{V}(s_V, \nu_V)$
- for every candidate ν_i, there is a free honest voter V_i that follows the protocol by executing $\mathcal{V}(s_{V_i}, \nu_i)$

there exists a run τ', where V does not execute $\mathcal{V}(s_V, \nu_V)$, such that $\tau \sim \tau'$.

First we show how τ' can be constructed given τ and then we show that $\tau \sim \tau'$.

A run τ can be characterized by the sequence of messages that have been sent over the network in τ. For every message in a run τ, either the message is sent by the attacker (or by a party under the control of the attacker), or else it is sent by an honest agent, and all the attacker can do is to learn that message. A run τ can be seen as a sequence of (partial) runs $\tau = \tau_1 \dots \tau_n$. In Caveat Coercitor, we split a run in a sequence corresponding to the

phases of the system: $\tau = \tau_{\mathsf{vote}}.\tau_{\mathsf{er}}.\tau_{\mathsf{mix}}.\tau_{\mathsf{tally}}$, where τ_{vote} is the list of ballots with valid zero-knowledge proofs that are cast on the bulletin board in the voting phase, τ_{er} is the electoral roll, τ_{mix} is the output of the re-encryption mix net and τ_{tally} is the output that tellers compute in the tallying phase, which includes data for the coercion-evidence test and the final outcome

Construction of the run τ'. There are four possible cases for the run τ:

- the coercer cast a vote with credential s_V for a candidate $\nu_{\mathcal{C}}$, with $\nu_{\mathcal{C}} \neq \nu_V$
- the coercer did not cast a vote with credential s_V
- the coercer cast a vote with credential s_V for the candidate ν_V
- the coercer cast at least two different votes with credential s_V

For simplicity, we only consider the first case, which is also the most likely. The other three cases can be handled in a similar way. We sketch the main ideas of the construction in the following and we give a complete description in the appendix.

Assume the coercer has cast a ballot $b_{s_V,\nu_{\mathcal{C}}} \in \mathcal{B}(s_V, \nu_{\mathcal{C}})$. By definition of the run τ, we know that the voter V has cast a ballot $b_{s_V,\nu_V} \in \mathcal{B}(s_V, \nu_V)$ and that there is a free honest voter V' that has cast a ballot $b_{s_{V'},\nu_{\mathcal{C}}} \in \mathcal{B}(s_{V'}, \nu_{\mathcal{C}})$. Thus, the ballots corresponding to s_V are cancelled because they are counted as coercion-evidence, while there is a vote counted for $\nu_{\mathcal{C}}$ that corresponds to $s_{V'}$.

On the other hand, a requirement of the definition is that the voter V obeys coercer in the run τ'. In our case this means that the voter V abstains from voting in the run τ'. Therefore, a vote for $\nu_{\mathcal{C}}$ corresponding to s_V is counted in the final tally. In order to obtain $\tau \sim \tau'$, we need at least to have the same outcome in the runs τ and τ'. Hence, we need the free voter V' to cancel her vote in the run τ': we assume V' casts a ballot $b_{s_{V'},\nu_{\mathcal{C}}} \in \mathcal{B}(s_{V'}, \nu_{\mathcal{C}})$, as in the run τ, and in addition a ballot $b_{s_{V'},\nu_V} \in \mathcal{B}(s_{V'}, \nu_V)$. The rest of the cast ballots in τ' are assumed to be exactly the same as in τ.

Thus, the amount of coercion-evidence and the final outcome of τ' are exactly the same as in τ. What we need in addition in order to achieve coercer independence is that:

- the coercer is not able to trace any of the cast ballots and detect that they where handled as coercion evidence

- the coercer is not able to decrypt the ballots before they are mixed

The first point is achieved by our trust assumptions that the voter V can cast her ballot on an anonymous channel and that at least one mix server is honest. The anonymous channel ensures that the coercer can not distinguish the two runs during the voting phase. Later, in the run τ', the honest mix server will permute the ballots so that coercion-evidence and the votes for $\nu_{\mathcal{C}}$ show up in the same places as in τ. The second point is achieved by out trust assumption that at least one teller is honest. Therefore, the ballots will be decrypted only as specified by the protocol, after the mixing occurs.

In conclusion, if $\tau = \tau_{\mathsf{vote}}.\tau_{\mathsf{er}}.\tau_{\mathsf{mix}}.\tau_{\mathsf{tally}}$, then we let $\tau' = \tau'_{\mathsf{vote}}.\tau'_{\mathsf{er}}.\tau'_{\mathsf{mix}}.\tau'_{\mathsf{tally}}$, where

- τ'_{vote} is as τ_{vote}, with the difference that the voter V abstains from voting and the free voter V' cancels her vote.

- $\tau'_{\mathsf{er}} = \tau_{\mathsf{er}}$

- τ'_{mix} is as τ_{mix}, with the difference that the honest mixer swaps some ballots corresponding to s_V with some ballots corresponding to $s_{V'}$, and also swaps the credentials of V and of V' while anonymizing the electoral roll.

- $\tau'_{\mathsf{tally}} = \tau_{\mathsf{tally}}$. This is possible because, by construction of the runs τ_{vote} and τ'_{vote}, it is the case that they determine the same outcome. Since there is one teller that is not under the control of the coercer, all the coercer can do is to observe the final outcome computed according to the specification of the protocol.

Indistinguishability of τ and τ'. Although the actions performed and observed by the coercer are the same in τ and in τ', it may be the case that the coercer could distinguish τ from τ' by performing various computations on the messages that where sent in τ and τ': encryption, re-encryption, checking zero-knowledge proofs, combining

messages, etc. In order to reason about the knowledge of the attacker, we therefore need a formal model of messages and of all possible computations that an attacker may perform. We adopt a variant of the applied pi-calculus [AF01], used by the protocol verifier ProVerif [Bla04, BAF08].

We specify for ProVerif the messages that can be built in Caveat Coercitor, the cryptographic primitives and the possible actions of the attacker. Then, ProVerif allows us to verify that all the pairs of runs of a *bi-process* are in *observational equivalence*. Observational equivalence models the indistinguishability relation between runs that we are interested in: $\tau \sim \tau'$ if and only if any computations applied to τ and to τ' lead to the same observations. A bi-process is a pair of applied pi-calculus processes (P, P'), see e.g. [AF01], that share the same structure and differ only on the messages that they handle. A bi-process (P, P') generates pairs of runs (τ, τ'), where τ is a run of P, τ' is a run of P' and τ and τ' have been generated by executing the same actions in P and in P'.

We observe that the runs τ and τ' constructed in section 5.5.1 are of the form $\tau = \tau_0.\tau_{\mathsf{tally}}$ and $\tau' = \tau'_0.\tau_{\mathsf{tally}}$, i.e. the tally in the two runs is exactly the same. Therefore, in order to show that $\tau \sim \tau'$, it is sufficient to show that $\tau_0 \sim \tau'_0$, where $\tau_0 = \tau_{\mathsf{vote}}.\tau_{\mathsf{er}}.\tau_{\mathsf{mix}}$ and $\tau'_0 = \tau'_{\mathsf{vote}}.\tau'_{\mathsf{er}}.\tau'_{\mathsf{mix}}$. Accordingly, we specify two process P_{cc} and $\mathsf{P}'_{\mathsf{cc}}$ such that the bi-process $(\mathsf{P}_{\mathsf{cc}}, \mathsf{P}'_{\mathsf{cc}})$ generates all the pairs of runs of the form $(\tau_{\mathsf{vote}}.\tau_{\mathsf{er}}.\tau_{\mathsf{mix}})$ and $(\tau'_{\mathsf{vote}}.\tau'_{\mathsf{er}}.\tau'_{\mathsf{mix}})$ as constructed in section 5.5.1. We show with ProVerif that observational equivalence holds for the given bi-process $(\mathsf{P}_{\mathsf{cc}}, \mathsf{P}'_{\mathsf{cc}})$ (the ProVerif code is attached in Appendix B and is also available online [1]), and therefore we can conclude the indistinguishability of any pair of runs (τ, τ'):

Corollary 2. *For any pair of runs (τ, τ') of Caveat Coercitor constructed as described in section 5.5.1, we have $\tau \sim \tau'$.*

From corollary 1 and corollary 2, we can conclude:

Theorem 1. *Caveat Coercitor satisfies coercion-evidence under the trust assumptions from section 5.4.1.*

[1] markryan.eu/research/caveat-coercitor/

5.6 Comparison with the related work

Caveat Coercitor can readily be compared with other systems that allow internet voting, such as JCJ/Civitas and Helios.

- **JCJ/Civitas** makes a strong assumption that there is an untappable channel for registration. It also assumes that the voter can run a multiparty protocols and keep real credentials secret all the time. Under these assumptions, it is strongly resistant to coercion, and is fully verifiable by voters and observers. Several variants of JCJ/Civitas improve the usability of the aspects related to verifiability [BGR11, SHKS11] and coercion-resistance [CH11].

  On the other hard, Caveat Coercitor makes more realistic assumptions about the voter's computation environment. It may be broken by the attacker, and if so, the attacker can get hold of voters credentials. However, coercion-evidence in Caveat Coercitor is not dependent on the secrecy of voting credentials, leading to more realistic assumption about the distribution of credentials. The registration phase, the voting phase and the resistance to coercion become simpler and more usable. For example, credentials can even be sent to voters by post. If credentials are leaked by anyone, e.g. registrars, postman or computer, coercion evidence will still be recorded. A limited form of coercion can take place, namely, the voter's real vote can be cancelled by a coercer (neither of their votes will be counted). If this happens, it cannot be hidden by the election authorities. Anyone can check if such coercion would make a material difference to the outcome. Caveat Coercitor inherits the same high degree of verifiability as JCJ/Civitas (that is, individual, universal and eligibility verifiability all hold).

- **Helios**, as explained earlier, is designed for low-coercion elections. Recall that, this system makes a few efforts to resist potential coercion, for example by keeping secret from voters the randoms in their ballots, but these efforts are easily defeated. The

most interesting feature of Helios is its high usability. Caveat Coercitor is designed to be as usable as Helios (indeed, it can have the same front end and voter experience). Moreover, Caveat Coercitor does not have the restriction to low-coercion elections of Helios. Although a weak form of coercion is possible in Caveat Coercitor, the incentive to coerce a voter is curtailed by two facts: the coercer can only achieve an annulled vote; and all attempted coercion is publicly verifiable by observers. Any observer can check if the actual coercion could have made a difference to the election outcome.

5.7 Concluding remarks

In this chapter, we proposed replacing the requirement of coercion-resistance with a new requirement of coercion-evidence to simplify the user's interaction with the voting systems. We also proposed Caveat Coercitor, an internet voting system to show how coercion-evidence could be achieved. To the best of our knowledge, our work is the first to propose an internet voting system with incoercibility properties that does not rely on an untappable channel. It is also the first work to realise the importance of considering "silent" coercion (coercion unknown to the voter being coerced, for example by leaked credentials) together with coercion that is known to the voter. Internet voting system like Helios that provide high level of usability to the voters could potentially be adapted to provide coercion-evidence. It would also be interesting to explore how the concept of coercion evidence could be applied in other contexts than E-voting. For example, it might be possible to detect malicious behaviour in Bitcoin and other electronic currencies using techniques similar to coercion-evidence.

CHAPTER 6

Discussion, further work, and conclusion

6.1 Discussion

In Chapter 1 section 1.2 we stated objectives of this thesis. Here we discuss what we have achieved with respect to those objectives.

Objective 1: Improve verifiability and privacy in the presence of untrusted computers.

Decades of research have concentrated on securing the back end of voting systems, while assuming that the front end—the voter's computer—is trusted. Unfortunately, the computers used by the voters to cast their votes are usually untrustworthy. So in Chapter 3, we set out to design a voting system, called Du-Vote, that would distribute trust between a voter's computer and a minimal token that could be made trustworthy. We eschewed designs that would have required tokens to have cameras, GUIs, or general-purpose operating systems. One of the challenge we consequently encountered was how to minimize the amount of information a human would have to convey between the token and computer. Typing short codes presented an attractive tradeoff between security and usability. We mentioned that such hardware tokens are standard in internet banking. However, the way we use these token in internet voting is novel. Moreover, we improve on the usage in banking in two ways: [a] the tokens provide a privacy-preserving (as well as integrity-preserving) channel; [b] we do not require the assumption that the tokens are trustworthy—we even permit the adversary to manufacture them.

Objective 2: Improve individual verifiability for voters.

In Chapter 4, we argued that the way individual verifiability is achieved in electronic voting is not intuitive and some voters are not convinced by the security guarantees provided by the cryptography (e.g. zero-knowledge proofs). So we proposed the use of trial votes to allow voters to cast votes and audit them at different stages of the election process. To show how this can be achieved we proposed several additions to JCJ/Civitas that improve its individual verifiability. We introduced trial credentials that offer to voters

the ability to audit the election process at any stage: creation of ballots, their transmission to the bulletin board, their processing by the mixnet and their final decryption. Moreover, we observed that we can rely on the presence of fake ballots and trial ballots on the bulletin board after the mix to decrypt all ballots without compromising coercion-resistance. This certainly improves individual verifiability: to our knowledge, this is the first mixnet based system where a voter can directly verify that her actual vote is correctly recorded in the system after the mix. Not only that, but she can also cast as many votes as she likes with fake credentials and check that they are all correctly output after the mix.

Objective 3: Improve usability by relaxing coercion-resistance.

We argued in the introduction of this thesis and in chapter 5 as well that the balance between coercion-resistance, election verifiability and, usability remains unresolved in internet voting despite significant research over the last few years. Therefore, we proposed coercion-evidence – a new property for e-voting systems that can replace coercion-resistance and improve usability of electronic voting systems. Our formal definition of coercion-evidence is general enough to permit the verification of coercion-evidence in various systems and different security models. To show how coercion-evidence can actually be achieved in practice and to show how it improves usability we proposed Caveat Coercitor – a voting system in which coercion is only weakly resisted, but made evident. The ability of a coercer in this system is limited, because the best the coercer can achieve is the annulment of a voter's vote. A major feature of the system (due to coercion-evidence) is that the degree of coercion that actually took place is publicly verifiable, provided the coerced voters follow the instructions of the protocol and cast votes for their desired candidate. This means that the coercer has only little incentive to coerce. Any observer can check if the coercion (that is, the annulled votes) could have made a material difference to the outcome.

6.2 Further work

In this section we will discuss possible further research work and identify open questions that have emerged from this thesis.

6.2.1 Could we integrate the schemes presented in this thesis?

Integrating Du-Vote and Trivitas: If we integrate Du-Vote and Trivitas then a voter will need to acquire both the hardware token and the credentials from the election authorities. She would receive her token by post, and then run a separate JCJ/Civitas like protocol to get her credentials (both real and trial) from distributed registration tellers. Then she will interact with her voting platform to generate a code page and follow instructions similar to the ones in Du-Vote (Chapter 3 Fig 3.4). Once the code for the desired candidate is chosen the voter will submit it to the server along her trial or real credentials depending on whether she wants to cast her real or trial vote. Theoretically an integration of these systems would be possible but the integrated system will be highly unusable. The amount of work that voters will have to do will be very tedious. Moreover, due to the presence of an untrusted machine the system will not satisfy coercion-resistance property. Therefore we do not recommend this integration.

Integrating Du-Vote and Caveat Coercitor: If we integrate Du-Vote and Caveat Coercitor then a voter would receive both the hardware token and credentials by post (perhaps in two different posts). At the time of voting, the voter will use the voting platform to generate the code page and follow her instructions to cast her vote. While submitting her vote to the server, she submits her credentials as well. This setup would make the final integrated system coercion-evident, provide eligibility verifiability, and work on untrusted platform without voters seeking multiple devices. As compared to the individual systems, the integrated system will be reasonably usable as well.

Integrating Caveat Coercitor and Trivitas: Integrating Caveat Coercitor and Trivitas would be quite straightforward as both systems are based on JCJ/Civitas. Voters in the integrated system will need to acquire their credentials from the distributed registration tellers. This is needed to maintain the indistingushability of trial and real credentials. From usability point of view, voters will have the same level of usability as in Trivitas. However, the requirement of an untappable channel, distributed credential distribution, and voters needing to keep their credentials secret makes this integrated system less appealing.

It should be noted that more research will be needed before trying to integrate any of the above mentioned systems as these analysis are informal. More careful and formal analysis will be required to verify the properties satisfied by these systems.

6.2.2 Usability studies

In this thesis, with our contributions in Chapter 3-5 we briefly described how a voter will interact with each system and we argued how it would be better than the existing voting systems. In the future we would like to conduct usability studies to identify user behaviour while interacting with these systems. That will give us more detailed view about the user behaviour and help us gather requirements to improve the user experience even further. We would also like to test different user interfaces for the systems presented in this thesis.

6.2.3 Towards digital democracy

The main focus of this thesis was about internet voting. However, the techniques presented can also enable direct participation in the democratic process, without relying on representatives. This would enable the next version of our democracy where citizens will be fully involved in the proposal, development and creation of laws. If well-designed and implemented mechanisms for internet voting and digital democracy are built, then that

would greatly increase societal inclusiveness and cohesion, as well as lowering the costs of making democracy work.

6.3 Concluding remarks

We discussed some of the practical problems that any deployable internet voting system would face. We identified three main issues: (a) computers used by the voters are untrusted (b) the security guarantees provided by cryptography are not intuitive and (c) coercion-resistance and verifiability can't be achieved along with usability. We went on to propose solutions for these problems that in the future will help overcome the above mentioned problems. The Digital Democracy Commission in the UK has recommended that the election should include internet voting in the year 2020. With the solutions proposed in this thesis we believe that we have taken secure internet voting a step towards being possible in the future.

Appendices

APPENDIX A

Appendices related to Chapter 3

This appendix contains proofs supporting Chapter 3.

A.1 Proofs used by S to prove its honesty

The server S creates universally-verifiable proofs to show that it correctly selected and reencrypted the ciphertext that was chosen by the voter. These proofs are posted on BB to allow anyone to check it. We briefly discussed these proofs in Chapter 3. Here we provide these proofs with their technical details and the cryptographic primitives that are required to understand them.

A.1.1 Ring signatures and OR-proofs

Consider a set $\{P_1, \ldots, P_n\}$ of participants, and suppose that each participant P_i has a private key sk_i and a corresponding public key pk_i. We suppose P_i possesses his own private key sk_i and all of the public keys $\{pk_j\}_{1 \leq j \leq n}$. A *ring signature* is a cryptographic signature

that such a participant P_i can make, and has the property that any verifier in possession of the public keys $\{pk_j\}_{1\leq j\leq n}$ can be assured that one of the participants $\{P_1, \ldots, P_n\}$ made the signature, without being able to tell which one it was [AOS02, CLMS08]. In a ring signature, there is one "actual" signer, in our example P_i; the signing process uses his secret key sk_i and the public keys $\{pk_j\}_{1\leq j\leq n,\, j\neq i}$ of the other ring members. The actual signer does not need the permission or involvement of the other ring members.

A zero-knowledge proof (ZKP) is a proof between a prover and a verifier that demonstrates to the verifier that a certain statement is true, without revealing any information except the validity of the statement [GMR89]. Non-interactive zero knowledge proofs (NIZKP) are a variant of ZKP that do not require interaction between the prover and the verifier. If the statement being proved asserts that the prover knows a secret signing key, this type of NIZKP can be a signature on some arbitrary value using the signing key, and is called a signature-based proof of knowledge (SPK). The message being signed is not important, it can be a constant value or even blank. A Schnorr signature [Sch91] is commonly used for this purpose.

If the statement being proved asserts that the prover knows a secret value satisfying a disjunction of properties, then a ring signature can be used. For example, given a finite set $\Omega = \{x_1, \ldots, x_n\}$, a group generator g, and a function f, we may prove in zero knowledge that we know a d such that $g^d = f(x_1)$ or $\ldots$ or $g^d = f(x_n)$. Such a proof is a ring signature on an arbitrary value, where the actual signing key is d and the public keys are $\{f(x) \mid x \in \Omega\}$. We write such a signature proof of knowledge as

$$SPK\{d \mid \exists x : x \in \Omega \wedge g^d = f(x)\}.$$

It is useful to generalise this to a *double signature*. Let Ω be a set of pairs, and g_1, g_2 be two group generators, and f_1, f_2 be two functions. We write

$$SPK\{d \mid \exists x_1, x_2 : (x_1, x_2) \in \Omega \wedge g_1^d = f_1(x_1) \wedge g_2^d = f_2(x_2)\}$$

to mean a proof of knowledge of d satisfying $g_1^d = f_1(x_1)$ and $g_2^d = f_2(x_2)$ for some $(x_1, x_2) \in \Omega$. The signer's private key is d, and the ring of public key pairs is $\{(f_1(x_1), f_2(x_2) \mid (x_1, x_2) \in \Omega\}$.

A.1.2 Proof of selection and re-encryption of the voter's chosen ciphertext

Using these ideas, we show how the server S produces a zero-knowledge proof that it correctly computed the voter's chosen ciphertext $x = (x_1, x_2)$ from the inputs C^* and x^* that S receives from P; and that it outputs a correct re-encryption of that x, all without revealing what x is. The proof is divided into three parts.

Proof of selection. S computes a pair $D = (D_1, D_2) = (g^d, y^d h^{x_2})$, for some value d, where x_2 is the second component of the voter's ciphertext; and proves that D is of this form.

To see how this is achieved, we assume that V's coin flip yielded tails, and V enters $B_1^* \ldots B_n^* x^*$ into H, where $x^* \in \{A_1^* \ldots A_n^*\}$. S starts as follows:

- retrieve $C = y^k h^{(B_1^* || \ldots || B_n^* || x^*)}$ and publish it. The last κ digits of C is C^*, which can be checked by everybody.
- compute $C_{x^*} = C / h^{(10^\kappa B_1^* || \ldots || B_n^*)} = y^k h^{x^*}$. This value is publicly known as anybody can compute it from C and $(B_1, \ldots, B_n)$.
- compute x' by truncating the last κ digits of x_2. Thus, $x_2 = x' || x^*$.
- compute $C_{x'} = y^{r'} h^{10^\kappa x'}$, and publish it, where r' is chosen randomly.
- prove knowledge of x' and r' in $C_{x'}$. We call this proof $P_{x'}$.
- compute $D = (D_1, D_2) = (g^d, C_{x^*} C_{x'}) = (g^d, y^d h^{x_2})$, where $d = k + r'$; anyone can compute D_2 from C_{x^*} and $C_{x'}$.

These computations demonstrate the relationship between D and C: they show that $D_2 = y^d h^{x_2}$ and $C = y^k h^{B_1^* || \ldots || B_n^* || x^*}$ share the same x^*.

To prove the structure of D, S must prove knowledge of d such that $D = (D_1, D_2) = (g^d, y^d h^{x_2})$, for some $x_2 \in \{A_{1,2}, \ldots, A_{n,2}\}$ (recall that each ElGamal ciphertext A_i is a pair $(A_{i,1}, A_{i,2})$). This proof can be written as

$$P_D = SPK\{d \mid \exists x_2 \in \Omega : \; g^d = D_1 \wedge y^d h^{x_2} = D_2\},$$

where $\Omega = \{A_{1,2}, \ldots, A_{n,2}\}$. We show that such a proof can be achieved, by writing it equivalently in the following form to match the shape of ring signature proofs above:

$$P_D = SPK\{d \mid \exists x_2 \in \Omega : \; (gy)^d = D_1 D_2 / h^{x_2}\}.$$

Proof of re-encryption of a ciphertext. As previously mentioned, the ballot x can't directly go to the bulletin board BB, because P knows the plaintext and could later use BB to see how the voter voted. To address this, S now computes a re-encryption $E = (E_1, E_2) = (g^e x_1, y^e x_2)$ of x (where e is chosen at random).

The proof of the structure of E can be shown using double ring signature:

$$P_E = SPK\{e \mid \exists x_1, x_2 : (x_1, x_2) \in \Omega \wedge g^e = E_1/x_1 \wedge y^e = E_2/x_2\},$$

where $\Omega = \{A_1, \ldots, A_n\}$.

Proof that the re-encryption is of the selected ciphertext. The proof P_E shows that E is the re-encryption of one of the values in $\{A_1, \ldots, A_n\}$, but not that it is the one chosen by the voter. To prove that the (x_1, x_2) used in the computation of E matches the code chosen by the voter, S has to show that the x_2 in E is the same as the x_2 in D. To demonstrate this, S computes $F = (F_1, F_2) = (E_1 D_1, E_2 D_2) = (g^f x_1, y^f x_2 h^{x_2})$

where $f = d + e$, and proves that F indeed has this form. The proof that F has the correct form is another double ring signature:

$$P_F = SPK\{f \mid \exists x_1, x_2 : (x_1, x_2) \in \Omega \wedge g^f = F_1/x_1 \wedge y^f = F_2/x_2 h^{x_2}\}$$

where $\Omega = \{A_1, \ldots, A_n\}$.

Note that by putting $P_{x'}, P_D, P_E$ and P_F together, we have actually proved that E is a re-encryption of $A_i = (A_{i,1}, A_{i,2})$ where $A_{i,2} = x_2$ and x_2^* is involved in C^*. The proofs $P_{x'}, P_D, P_E$ and P_F are bound together as a single Schnorr signature proof P_{total}. This proof P_{total} is also posted on the bulletin board BB.

Our constructions are novel, but they make use of the standard signature-based proof of knowledge (SPK) technique. By using the Fiat-Shamir [FS86] heuristic, we are assured that the proof is sound and complete, and has the computational zero-knowledge property.

A.2 Proof that the encryption scheme is IND-CPA

Du-Vote requires the platform to find ciphertexts for which, when written in decimal form, the last κ digits are equal to certain codes. One might be concerned that adding such a requirement may imply that the encryption scheme is insecure. To alleviate such fears, in this section we prove that the encryption scheme is IND-CPA.

Definitions. Let Encrypt0 be standard ElGamal, and Encrypt1 be our variant. Formally, they are defined as follows:

Encrypt0(m , y) = // standard ElGamal
random r;
$(c_1 , c_2) = (g^r , m.y^r)$;
output (c_1 , c_2)

Encrypt1(m , y , *code*) = // our variant

new r;

$(c_1 , c_2) = (g^r , m.y^r)$;

while c_2^* != *code* {

$(c_1 , c_2) = (c_1.g , c_2.y)$

}

output (c_1 , c_2)

Recall that Encrypt0(m, y) satisfies IND-CPA. Thus, we may consider a typical CPA game between a simulator and an attacker. In such a game, the attacker receives the public key y from the simulator; the attacker sends a pair of challenge messages (m_0, m_1) to the simulator; the simulator chooses a random bit b, encrypts m_b and returns the ciphertext; the attacker outputs b'. If $b' = b$, the attacker wins the game.

Assumption. Let m and y be fixed, and i be a randomly chosen q-bit value. Then $(m.y^i)^*$ is uniformly distributed in the space of κ-digit decimal numbers.

Theorem 2. *Under our assumption, Encrypt1*$(m, y, code)$ *satisfies IND-CPA.*

Proof: Suppose Adversary A has a target of breaking Encrypt0(m, y). If there exists Adversary B who is able to break Encrypt1$(m, y, code)$, then A can use B to break Encrypt0(m, y).

We set up two games. Game 1 is between a simulator S and the adversary A (playing as an attacker), and Game 2 is between A (playing as a simulator) and B (playing as an attacker).

- In Game 1, A gets the ElGamal public key y from S.
- In Game 2, A forwards y to B.

- In Game 2, A receives a pair of challenge messages (m_0, m_1) from B.
- In Game 1, A forwards (m_0, m_1) to S.
- In Game 1, A receives $(c_1, c_2) = (g^r, m_b \cdot y^r)$, where $b \in \{0, 1\}$, from S.
  - A searches for a value i satisfying $(c'_1, c'_2) = (c_1.g^i, c_2.y^i)$ and $c'^*_2 = code$.
  - A terminates the games if such a value doesn't exist.
- In Game 2, A sends (c'_1, c'_2) to B.
- In Game 2, B returns b' to A.
- In Game 1, A returns b' to S.

If A succeeds in finding the value i, the probability of A winning Game 1 is equal to the probability of B winning Game 2.

Let us now calculate the probability that A succeeds in finding the value i. For a given value of r and i, the probability that $(m.y^{r+i})^* = code$ is $1/10^\kappa$ (here we rely on the assumption). The probability that $(m.y^{r+i})^* \neq code$ is, therefore, $1 - 1/10^\kappa$. We have $2^{|q|}$ possible values of i, so the probability of none of them satisfying the equation is $(1 - 1/10^\kappa)^{2^{|q|}}$. Therefore the probability that there is at least one i that satisfies the equation is $\pi = 1 - (1 - 1/10^\kappa)^{2^{|q|}})$.

The probability that A wins Game 1 is π times the probability of B winning Game 2. The value $1 - \pi$ is negligible in q, so the theorem follows.

APPENDIX B

Appendices related to Chapter 5

This appendix contains proofs supporting Chapter 5

B.1 Proof for coercer independence 5.5.2

To prove that Caveat Coercitor satisfies coercer independence, we have to show that for every run τ of Caveat Coercitor where

- the voter V follows the protocol by executing $\mathcal{V}(s_V, \nu_V)$
- for every candidate ν_i, there is a free honest voter V_i that follows the protocol by executing $\mathcal{V}(s_{V_i}, \nu_i)$

there exists a run τ', where V does not execute $\mathcal{V}(s_V, \nu_V)$, such that $\tau \sim \tau'$.

We gave a sketch of the main ideas of the construction of run τ' in chapter 5 section 5.5.2. Here we give a complete description.

Construction of the run τ'. The main idea in the construction of τ' is to rely on the free voter V' in the voting phase so that the ballots of V' in the run τ' look like the ballots of voter V in the run τ, and the other way around. Furthermore, we rely on an honest re-encryption mix server to ensure that the coercer can not track the ballots back to V or to V'. The honest mix server swaps the ballots of V and V' and also their credentials in the anonymized electoral roll. An honest tabulation teller ensures that the tallying is performed according to the specification of the protocol, both in τ and in τ'.

For a run $\tau = \tau_{\mathsf{vote}}.\tau_{\mathsf{er}}.\tau_{\mathsf{mix}}.\tau_{\mathsf{tally}}$ as the one in section 5.5.2, we let $\tau' = \tau'_{\mathsf{vote}}.\tau_{\mathsf{er}}.\tau'_{\mathsf{mix}}.\tau'_{\mathsf{tally}}$, where τ'_{vote}, τ'_{mix} and τ'_{tally} are defined in the following.

Voting phase. We have by definition at least three particular ballots present in τ_{vote} (recall that we consider the case when the coercer has cast a vote for $\nu_{\mathcal{C}}$, $\nu_{\mathcal{C}} \neq \nu_V$, using the credential s_V): two with credential s_V and one with credential $s_{V'}$. Without loss of generality, we fix an order for these three ballots on the bulletin board. Then, we have:

$$\tau_{\mathsf{vote}} = L_1.b_{s_V,v_{\mathcal{C}}}.L_2.b_{s_V,v_V}.L_3.b_{s_{V'},v_{\mathcal{C}}}.L_4$$

where $b_{s_V,\nu_{\mathcal{C}}} \in \mathcal{B}(s_V, \nu_{\mathcal{C}})$, $b_{s_V,\nu_V} \in \mathcal{B}(s_V, \nu_V)$, $b_{s_{V'},\nu_{\mathcal{C}}} \in \mathcal{B}(s_{V'}, \nu_{\mathcal{C}})$ and L_1,L_2,L_3,L_4 are lists of ballots. Furthermore, for any $1 \leq \ell \leq 4$, if there is a ballot in L_i that corresponds to the credential $s_{V'}$, then the corresponding vote is $\nu_{\mathcal{C}}$.

Let c_V and respectively $c_{V'}$ be the public credentials that correspond to the private credentials s_V and $s_{V'}$. Then, without loss of generality, we can assume that the electoral roll is

$$\tau_{\mathsf{er}} = S_1.c_V.S_2.c_{V'}.S_3$$

where S_1, S_2 and S_3 are the public credentials of other voters. Now, for some $b_{s_{V'},\nu_V} \in \mathcal{B}_{s_{V'},\nu_V}$, let us consider τ'_{vote} defined as follows

$$\tau'_{\mathsf{vote}} = L_1.b_{s_V,\nu_{\mathcal{C}}}.L_2.b_{s_{V'},\nu_V}.L_3.b_{s_{V'},\nu_{\mathcal{C}}}.L_4$$

Hence, τ'_{vote} is the same as τ_{vote}, with the difference that V now follows the instructions of the coercer and does not cast a vote for the intended candidate, while the free voter V' becomes dishonest, and casts an additional ballot for the candidate intended by V.

Mixing phase. According to our trust assumptions, there is at least one mix server that is not controlled by the coercer. Let $\tau_{\mathsf{mix}} = \tau^0_{\mathsf{mix}}.\tau^1_{\mathsf{mix}}.\tau^h_{\mathsf{mix}}.\tau^2_{\mathsf{mix}}$, where:

• τ^0_{mix} is the input to the mix network, i.e. $\tau^0_{\mathsf{mix}} = \tau^0_{\mathsf{vote}}.\tau_{\mathsf{er}}.\tau_{\mathsf{cand}}$, where τ^0_{vote} contains the ballots from τ_{vote}, but not their corresponding zero-knowledge proofs, and τ_{cand} is a list of encrypted candidate names (to be used in the coercion-evidence test, cf Algorithm 2).

• τ^1_{mix} is the output of (dishonest) mix servers that are present in the mixnet before the honest mix server.

• τ^h_{mix} is the output of the honest mix server.

• τ^2_{mix} is the output of (dishonest) mix servers that are present in the mixnet after the honest mix server.

Now, we consider $\tau'_{\mathsf{mix}} = \tau'^0_{\mathsf{mix}}.\tau'^1_{\mathsf{mix}}.\tau'^h_{\mathsf{mix}}.\tau'^2_{\mathsf{mix}}$, where:

• $\tau'^0_{\mathsf{mix}} = \tau'^0_{\mathsf{vote}}.\tau'_{\mathsf{er}}.\tau'_{\mathsf{cand}}$, where τ'^0_{vote} contains the ballots from τ'_{vote}, but not their corresponding zero-knowledge proofs,, and $\tau'_{\mathsf{er}} = \tau_{\mathsf{er}}, \tau'_{\mathsf{cand}} = \tau_{\mathsf{cand}}$.

• τ'^1_{mix} is constructed from τ'^0_{mix} by applying exactly the same operations as for constructing τ^1_{mix} from τ^0_{mix}. This possible because these operations consist only in performing re-encryptions and applying permutations to the set of ballots: they do not depend on the content of ballots.

• τ'^h_{mix} is constructed as follows. Recall that $\tau^0_{\mathsf{mix}} = \tau^0_{\mathsf{vote}}.\tau_{\mathsf{er}}.\tau_{\mathsf{cand}}$, where τ^0_{vote} is τ_{vote} without the zero-knowledge proofs of correctness, i.e. we have

$$\tau^0_{\mathsf{vote}} = L^0_1.b^0_{s_V,\nu_\mathcal{C}}.L^0_2.b^0_{s_V,\nu_V}.L^0_3.b^0_{s'_V,\nu_\mathcal{C}}.L^0_4$$

where we denote by b_0 (resp. L^0) a ballot b (resp. a list of ballots L) stripped of its proofs. Let τ^{ih}_{mix} be the input for the honest mix server in τ_{mix}, i.e. the output of the last dishonest server in τ^1_{mix}. Relying on the zero-knowledge proofs that have to be provided by any

mix server (including the dishonest ones), we deduce that $\tau^{ih}_{\mathsf{mix}} = \rho_1.\rho_2.\rho_3$ where ρ_1, ρ_2 and respectively ρ_3 are re-encryption mixes of τ^0_{vote}, τ_{er} and respectively τ_{cand}, with some permutations chosen by the coercer. Thus, we have

$$\begin{aligned} \rho_1 &= L_1.w_1.L_2.w_2.L_3.w_3.L_4 \\ \rho_2 &= S_1.t_1.S_2.t_2.S_3 \end{aligned}$$

where $w_1.w_2.w_3$ is a re-encryption mix of $b^0_{s_V,\nu_\mathcal{C}}.b^0_{s_V,\nu_V}.b^0_{s'_V,\nu_\mathcal{C}}$ and $t_1.t_2$ is a re-encryption mix of $c_V.c_{V'}$. Furthermore, $\tau^h_{\mathsf{mix}} = \rho^h_1.\rho^h_2.\rho^h_3$, where, for every $1 \leq i \leq 3$, ρ^h_i is a re-encryption mix of ρ_i, with some permutation chosen by the honest mix server. Let:

$$\begin{aligned} \rho^h_1 &= L^h_1.w^h_1.L^h_2.w^h_2.L^h_3.w^h_3.L^h_4 \\ \rho^h_2 &= S^h_1.t^h_1.S^h_2.t^h_2.S^h_3 \end{aligned}$$

where $w^h_1.w^h_2.w^h_3$ is a re-encryption mix of $w_1.w_2.w_3$ and $t^h_1.t^h_2$ is a re-encryption mix of $t_1.t_2$.

On the other hand, according to our construction of ${\tau'}^1_{\mathsf{mix}}$, the input to the honest mix server in τ'_{mix} is ${\tau'}^{ih}_{\mathsf{mix}} = \rho'_1.\rho_2.\rho_3$, where

$$\rho'_1 = L_1.w'_1.L_2.w'_2.L_3.w'_3.L_4$$

and $w'_1.w'_2.w'_3$ is a re-encryption mix of $b^0_{s_V,\nu_\mathcal{C}}.b^0_{s_{V'},\nu_V}.b^0_{s_{V'},\nu_\mathcal{C}}$ such that, for every $1 \leq i \leq 3$, if $w'_i \in \mathcal{B}_0(s_V, \nu_\mathcal{C})$ or $w'_i \in \mathcal{B}_0(s_{V'}, \nu_\mathcal{C})$, then $w'_i = w_i$. Thus, as in the case of the pair $(\tau_{\mathsf{vote}}, \tau'_{\mathsf{vote}})$, the only difference between ${\tau'}^{ih}_{\mathsf{mix}}$ and τ^{ih}_{mix} is that, for some $1 \leq i \leq 3$, we have $w'_i \in \mathcal{B}_0(s_{V'}, \nu_V)$ and $w_i \in \mathcal{B}_0(s_V, \nu_V)$.

In the run τ' the output of the honest mix server must be ${\tau'}^h_{\mathsf{mix}} = {\rho'_1}^h.{\rho'_2}^h.{\rho'_3}^h$, corresponding to re-encryption mixes of ρ'_1, ρ_2 and ρ_3. Relying on the fact that the mix server producing ${\tau'}^h_{\mathsf{mix}}$ is honest, we can choose particular permutations and particular randoms for the re-encryption mixes that produce ${\tau'}^h_{\mathsf{mix}}$. We choose these such that we have the following:

$$\begin{aligned} \rho'^h_1 &= L^h_1.w'^h_1.L^h_2.w'^h_2.L^h_3.w'^h_3.L \\ \rho'^h_2 &= S^h_1.t'^h_1.S^h_2.t'^h_2.S^h_3 \end{aligned}$$

where $w'^h_1.w'^h_2.w'^h_3$ is a re-encryption mix of $w'_1.w'_2.w'_3$ and $t'^h_1.t'^h_2$ is a re-encryption mix of $t_1.t_2$ such that:

- $w'^h_i \in \mathcal{B}_0(s_{V'}, \nu_{\mathcal{C}}) \Leftrightarrow w^h_i \in \mathcal{B}_0(s_V, \nu_{\mathcal{C}})$.
- $w'^h_i \in \mathcal{B}_0(s_{V'}, \nu_V) \Leftrightarrow w^h_i \in \mathcal{B}_0(s_V, \nu_V)$
- $w'^h_i \in \mathcal{B}_0(s_V, \nu_{\mathcal{C}}) \Leftrightarrow w^h_i \in \mathcal{B}_0(s_{V'}, \nu_{\mathcal{C}})$
- $\mathsf{pet}(t'^h_1, c_V) = \mathsf{ok} \Leftrightarrow \mathsf{pet}(t^h_1, c_{V'}) = \mathsf{ok}$
- $\mathsf{pet}(t'^h_2, c_{V'}) = \mathsf{ok} \Leftrightarrow \mathsf{pet}(t^h_2, c_V) = \mathsf{ok}$

Intuitively, the two ballots from V' and the ballot from V are re-arranged in the run τ'_{mix} by the honest mix server so that the plaintext equivalence tests performed for coercion-evidence will succeed for the same positions in τ' as in τ, and also the vote for $v_{\mathcal{C}}$ is revealed in the same position. Furthermore, the credentials of V and V' are swapped on the electoral roll, so that the credential that is marked as coerced is at the same position in both runs.

- τ'^2_{mix} is constructed from τ'^h_{mix} by performing exactly the same operations as for constructing τ^2_{mix} from $\tau_m ix^h$. This is possible because the operations performed by a re-encryption mix server do not depend on the content of ballots.

Tallying phase. Let $\tau^{\mathsf{out}}_{\mathsf{mix}}$ and respectively $\tau'^{\mathsf{out}}_{\mathsf{mix}}$ be the outcome of the re-encryption mix net after the (partial) runs $\tau_{\mathsf{vote}}.\tau_{\mathsf{mix}}$ and $\tau'_{\mathsf{vote}}.\tau'_{\mathsf{mix}}$. Note that $\tau^{\mathsf{out}}_{\mathsf{mix}} = \tau^{\mathsf{out}}_{\mathsf{vote}}.\tau^{\mathsf{out}}_{\mathsf{er}}.\tau^{\mathsf{out}}_{\mathsf{cand}}$ and $\tau'^{\mathsf{out}}_{\mathsf{mix}} = \tau'^{\mathsf{out}}_{\mathsf{vote}}.\tau'^{\mathsf{out}}_{\mathsf{er}}.\tau'^{\mathsf{out}}_{\mathsf{cand}}$, where for all $a \in \{\mathsf{vote}, \mathsf{er}, \mathsf{out}\}$, τ^{out}_a and respectively τ'^{out}_a is a re-enncryption mix of τ_a and respectively τ'_a. Furthermore, by construction we have:

- $\tau^{\mathsf{out}}_{\mathsf{cand}} = \tau'^{\mathsf{out}}_{\mathsf{cand}}$
- $\tau^{\mathsf{out}}_{\mathsf{er}} = t_1 \ldots t_n$ and $\tau'^{\mathsf{out}}_{\mathsf{er}} = t'_1 \ldots t'_n$ such that for every $1 \leq i \leq n$, we have

  - either $t'_i = t_i$
  - or else $\mathsf{pet}(t'_i, c_V) = \mathsf{ok}$ and $\mathsf{pet}(t_i, c_{V'}) = \mathsf{ok}$
  - or else $\mathsf{pet}(t'_i, c_V) = \mathsf{ok}$ and $\mathsf{pet}(t_i, c_{V'}) = \mathsf{ok}$

- $\tau^{\mathsf{out}}_{\mathsf{vote}} = w_1 \dots w_m$ and $\tau'^{\mathsf{out}}_{\mathsf{vote}} = w'_1 \dots w'_m$ such that for every $1 \leq i \leq m$, we have
  - either $w'_i = w_i$
  - or else $w'_i \in \mathcal{B}_0(s_{V'}, \nu_{\mathcal{C}})$ and $w_i \in \mathcal{B}_0(s_V, \nu_{\mathcal{C}})$
  - or else $w'_i \in \mathcal{B}_0(s_{V'}, \nu_V)$ and $w_i \in \mathcal{B}_0(s_V, \nu_V)$
  - or else $w'_i \in \mathcal{B}_0(s_V, \nu_{\mathcal{C}})$ and $w_i \in \mathcal{B}_0(s_{V'}, \nu_{\mathcal{C}})$

The trust assumptions for Caveat Coercitor ensure that there is at least one honest teller. The honest teller will follow the protocol and will not reveal his share of the decryption key to the coercer. Therefore, the only way to complete any run in the tallying phase for the coercer is to obey the specification of the protocol. Thus, we have $\tau_{\mathsf{tally}} = \tau_{\mathsf{valid}}.\tau_{\mathsf{buckets}}.\tau_c^1 \dots \tau_c^n.\tau_{\mathsf{dec}}$, where

- τ_{valid} are the ballots from $\tau^{\mathsf{out}}_{\mathsf{vote}}$ with eligible credentials
- τ_{buckets} is the outcome of plaintext equivalence tests that are performed to group the ballots according to the vote that they contain (step 1 of algorithm 2)
- for every $1 \leq i \leq n$, τ_c^i represents the outcome of plaintext equivalence tests performed to determine if the voter with credential t_i is coerced (step 2 of algorithm 2)
- τ_{dec} represents the result of decrypting ballots with valid credentials whose corresponding voters have not been coerced or dishonest.

Then, we let $\tau'_{\mathsf{tally}} = \tau'_{\mathsf{valid}}.\tau'_{\mathsf{buckets}}.\tau'^1_c \dots \tau'^n_c.\tau'_{\mathsf{dec}}$, where:

- because s_V and $s_{V'}$ are valid credentials by assumption, we note that the set of ballots with ineligible credentials is exactly the same in $\tau^{\mathsf{out}}_{\mathsf{vote}}$ and $\tau'^{\mathsf{out}}_{\mathsf{vote}}$. Therefore, if $\tau_{\mathsf{valid}} = \tau^{\mathsf{out}}_{\mathsf{vote}} \setminus S$, then we can choose $\tau'_{\mathsf{valid}} = \tau'^{\mathsf{out}}_{\mathsf{vote}} \setminus S$.

- from our observation about $(\tau^{\mathsf{out}}_{\mathsf{vote}}, \tau'^{\mathsf{out}}_{\mathsf{vote}})$ and $(\tau^{\mathsf{out}}_{\mathsf{er}}, \tau'^{\mathsf{out}}_{\mathsf{er}})$, it follows that the plaintext equivalence tests from algorithm 2 and the decryption of the final outcome give exactly the same results for $(\tau_{\mathsf{valid}}, \tau^{\mathsf{out}}_{\mathsf{er}})$ and $(\tau'_{\mathsf{valid}}, \tau'^{\mathsf{out}}_{\mathsf{er}})$. Therefore, we can choose $\tau'_{\mathsf{buckets}} = \tau_{\mathsf{buckets}}, \tau'^{i}_{c} = \tau^{i}_{c}$, for all $1 \leq i \leq n$, and $\tau'_{\mathsf{dec}} = \tau_{\mathsf{dec}}$.

This way we finish the construction of τ'.

Indistinguishability of τ and τ'.

The ProVerif code below (which is also available online at *markryan.eu/research/caveat-coercitor*) verifies the observational equivalence of a bi-process $(P_{\mathsf{cc}}, P'_{\mathsf{cc}})$, which generates pairs of runs of the form (τ, τ'), as described in section 5.5.2. P_{cc} and P'_{cc} share the same structure and differ only on some (internal) messages. This difference can be expressed in ProVerif with the construct $\mathsf{choice}[u, v]$: the process P_{cc} (also called the left process) uses the term u, whereas the process P'_{cc} (the right process) uses the term v. To model the difference between the runs τ and τ' from Caveat Coercitor all we need is to make sure that the ballots that are cast are as expected. Then, ProVerif explores all the possible runs starting from that.

```
(* CANDIDATES  *)
fun can1/0. fun can2/0.

(* PUBLIC KEY ENCRYPTION & RE-ENCRYPTION *)
fun pub/1. fun enc/3. fun dec/2.
fun renc/2. fun f/2.

equation dec(enc(x, pub(y), z), y) = x.
(*equation renc(enc(x,y,z),z') = enc(x,y,f(z,z')).*)

(*ZERO KNOWLEDGE PROOFS*)
```

```
fun ok/0.

(*PROOF OF VALID DECRYPTION*)
fun decProof/3. fun checkDec/4.
equation checkDec(decProof(enc(x,pub(y),z),x,y),
enc(x,pub(y),z),x, pub(y))=ok.

(*PROOF OF KNOWLEDGE OF PLAINTEXT*)
fun knowsProof/3. fun checkKnows/2.
equation checkKnows(knowsProof(enc(x,y,z),x,z),
enc(x,y,z)) = ok.

(*PROOF THAT THE VOTE IS IN VALID RANGE*)
fun voteProof/3. fun checkVote/2.
equation checkVote(voteProof(enc(can1,y,z),can1,z),
enc(can1,y,z)) = ok.

equation checkVote(voteProof(enc(can2,y,z),can2,z),
enc(can2,y,z)) = ok.

(*CHANNELS*)
free bb_ballots.
free bb_credentials.

(*SECRET KEY OF THE ELECTION*)
private free sk.

(*PRIVATE CREDENTIALS OF VOTERS A AND B*)
```

```
private free priv_a,priv_b.
free credentials.

let Credentials =
(*PUBLIC CREDENTIALS OF VOTER A AND VOTER B*)
new ra; new rb;
let pub_a = enc(priv_a,pub(sk),ra) in
        let pub_b = enc(priv_b,pub(sk),rb) in
out(bb_credentials,choice[pub_a,pub_b]);
out(bb_credentials,choice[pub_b,pub_a]);

(*PUBLIC CREDENTIALS OF ANY VOTER*)
in(credentials, priv_c);
new rc;
let pub_c = enc(priv_c,pub(sk),rc) in
out(bb_credentials,pub_c).

(*CANDIDATES*)
free candidates.
let Candidates = ! (out(candidates,can1) | out(candidates, can2)).

(*VOTERS*)
(*THE PROCESS VoterAB AND VoterB REPRESENT SOME BALLOTS THAT SHOULD BE
PRESENT ON THE BULLETIN BOARD ACCORDING TO THE SPECIFICATION OF RUNS TAU
 AND TAU'.  *)

(*IN THE RUN TAU (THE FIRST COMPONENT OF choice): VOTER A CASTS A VOTE
FOR THE INTENDED CANDIDATE can1 AND VOTER B CASTS A VOTE FOR THE
```

```
CANDIDATE DESIRED BY THE COERCER can2*)

(*IN THE RUN TAU' (THE SECOND COMPONENT OF choice): VOTER A ABSTAINS
AND THE VOTER B CASTS TWO BALLOTS: ONE FOR can1 AND ONE FOR can2*)

let VoterAB = new r1;new r2;

(*BALLOT FOR THE VOTER A IN THE RUN TAU*)
let enc_voteA = enc(can1,pub(sk),r1) in
let enc_credA = enc(priv_a,pub(sk),r2) in
let knows_proofA = (knowsProof(enc_voteA,can1,r1), knowsProof(enc_credA,
priv_a,r2)) in
let vote_proofA =  voteProof(enc_voteA,can1,r1) in
let ballotA = (enc_voteA,enc_credA, knows_proofA, vote_proofA) in

(*BALLOT FOR THE FREE VOTER B IN THE RUN TAU'*)
let enc_voteB = enc(can1,pub(sk),r1) in
let enc_credB = enc(priv_b,pub(sk),r2) in
let knows_proofB = (knowsProof(enc_voteB,can1,r1),
    knowsProof(enc_credB, priv_b,r2)) in
let vote_proofB =  voteProof(enc_voteB,can1,r1) in
let ballotB = (enc_voteB,enc_credB, knows_proofB, vote_proofB) in

(*OUTPUT OF ballotA IN THE RUN TAU AND OF ballotB IN THE RUN TAU'*)
out(bb_ballots,choice[ballotA,ballotB]).

(*THE PROCESS VoterB REPRESENTS A BALLOT FROM THE FREE VOTER B
FOR THE CANDIDATE can2 DESIRED BY THE COERCER *)
```

```
let VoterB = new r1; new r2;
let enc_vote = enc(can2,pub(sk),r1) in
let enc_cred = enc(priv_b,pub(sk),r2) in
let knows_proof = (knowsProof(enc_vote,can2,r1),
   knowsProof(enc_cred, priv_b,r2)) in
let vote_proof =  voteProof(enc_vote,can2,r1) in
let ballot = (enc_vote,enc_cred, knows_proof, vote_proof) in

out(bb_ballots,ballot).

let VoterGeneral =
     new r1; new r2;
     in(candidates,can);
     in(credentials,priv_c);
     let enc_vote = enc(can,pub(sk),r1) in
             let enc_cred = enc(priv_c,pub(sk),r2) in
     let knows_proof = (knowsProof(enc_vote,can,r1),
     knowsProof(enc_cred, priv_c,r2)) in
     let vote_proof =  voteProof(enc_vote,can,r1) in
     let ballot = (enc_vote,enc_cred, knows_proof, vote_proof) in
     out(bb_ballots,ballot).

let BallotsToMixnet = ! ( in(bb_ballots,(x,y,z,w)); out(bb_ballots,(x,y)) ).

let Mixnet = BallotsToMixnet | !MixBallots | !MixCredentials.

let MixBallots = in(bb_ballots,(x,y));
 let vote = dec(x,sk) in
```

```
 let cred = dec(y,sk) in
 new rx;new ry;
 let newx = enc(vote, pub(sk),rx) in
 let newy = enc(cred, pub(sk),ry) in
 out(bb_ballots, (newx,newy)).

let MixCredentials =
 in(bb_credentials,x);
 let cred = dec(x,sk) in
 new rx;
 let newx = enc(cred, pub(sk),rx) in
 out(bb_credentials, newx).

free roc.
let ReencryptionOracle =
 in(roc,ciph);
 let plain = dec(ciph,sk) in
 new random;
 let renc_ciph = enc(plain,pub(sk),random) in
 out(roc,renc_ciph).

process
out(bb,priv_a); (*** THE VOTER A IS COERCED: HIS CREDENTIAL
 IS KNOWN TO THE ATTACKER ***)

out(bb,pub(sk));  Candidates | Credentials | VoterAB
| VoterB | VoterGeneral | Mixnet | ReencryptionOracle
```

References

[Adi06] Ben Adida. *Advances in Cryptographic Voting Systems*. PhD thesis, Massachusetts Institute of Technology, August 2006.

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 335–348. USENIX Association, 2008.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 104–115. ACM, 2001.

[AFT07] Roberto Araújo, Sébastien Foulle, and Jacques Traoré. A practical and secure coercion-resistant scheme for remote elections. In *Proceedings of Workshop on Frontiers in Electronic Election*, 2007.

[AFT10] Roberto Araújo, Sébastien Foulle, and Jacques Traoré. A practical and secure coercion-resistant scheme for internet voting. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, volume 6000 of *Lecture Notes in Computer Science*, pages 330–342. Springer, 2010.

[AHL+09] Arne Ansper, Sven Heiberg, Helger Lipmaa, Tom André Øverland, and Filip van Laenen. Security and trust for the Norwegian e-voting pilot project *E-valg 2011*. In *Identity and Privacy in the Internet Age, 14th Nordic Conference on Secure IT Systems, NordSec 2009, Oslo, Norway, 14-16 October 2009. Proceedings*, volume 5838 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2009.

[Aka09] Adi Akavia. Solving hidden number problem with one bit oracle and advice. In *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 337–354. Springer, 2009.

[AN06] Ben Adida and C. Andrew Neff. Ballot casting assurance. In *2006 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06, Vancouver, BC, Canada, August 1, 2006*. USENIX Association, 2006.

[AOS02] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. In *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, volume 2501 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2002.

[AT13] Roberto Araújo and Jacques Traoré. A practical coercion resistant voting scheme revisited. In *E-Voting and Identify - 4th International Conference, Vote-ID 2013, Guildford, UK, July 17-19, 2013. Proceedings*, volume 7985 of *Lecture Notes in Computer Science*, pages 193–209. Springer, 2013.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[BCE+07] Matt Blaze, Arel Cordero, Sophie Engle, Chris Karlof, Naveen Sastry, Micah Sherr, Till Stegers, and Ka-Ping Yee. Source code review of the Sequoia voting system. In *Report commissioned as part of the California Secretary of State's Top-To-Bottom Review of California voting systems*, July 20, 2007.

[BCG+15] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy, (S&P 2015), San Jose, CA, USA, May 17-21, 2015*, pages 499–516. IEEE Computer Society, 2015.

[BCGG99] Sebastiano Battiato, Dario Catalano, Giovanni Gallo, and Rosario Gennaro. Robust watermarking for images based on color manipulation. In *Information Hiding, Third International Workshop, IH'99, Dresden, Germany, September*

29 - October 1, 1999, Proceedings, volume 1768 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 1999.

[BCH+12] Craig Burton, Chris Culnane, James Heather, Thea Peacock, Peter Y. A. Ryan, Steve Schneider, Vanessa Teague, Roland Wen, Zhe Xia, and Sriramkrishnan Srinivasan. Using prêt à voter in victoria state elections. In *2012 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '12, Bellevue, WA, USA, August 6-7, 2012*. USENIX Association, 2012.

[Ben06] Josh Benaloh. Simple verifiable elections. In *2006 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06, Vancouver, BC, Canada, August 1, 2006*. USENIX Association, 2006.

[Ben07] Josh Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *2007 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07, Boston, MA, USA, August 6, 2007*. USENIX Association, 2007.

[BG02] Dan Boneh and Philippe Golle. Almost entirely correct mixing with applications to voting. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 68–77. ACM, 2002.

[BGR11] Sergiu Bursuc, Gurchetan S. Grewal, and Mark D. Ryan. Trivitas: Voters directly verifying votes. In *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers*, volume 7187 of *Lecture Notes in Computer Science*, pages 190–207. Springer, 2011.

[BHM08] Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-Calculus. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, 23-25 June 2008*, pages 195–209. IEEE Computer Society, 2008.

[Bla04] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *2004 IEEE Symposium on Security and Privacy (S&P 2004), 9-12 May 2004, Berkeley, CA, USA*, pages 86–100. IEEE Computer Society, 2004.

[Bra05] Felix Brandt. Efficient cryptographic protocol design based on distributed El Gamal encryption. In *Information Security and Cryptology - ICISC 2005, 8th*

International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers, volume 3935 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2005.

[BT94] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 544–553. ACM, 1994.

[BVQ10] Josh Benaloh, Serge Vaudenay, and Jean-Jacques Quisquater. Final report of IACR Electronic Voting Committee. International Association for Cryptologic Research, 2010.

[CCC+09] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II: End-to-end verifiability by voters of optical scan elections through confirmation codes. *IEEE Transactions on Information Forensics and Security*, 4(4):611–627, 2009.

[CCC+10] Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Herrnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II municipal election at Takoma Park: The first E2E binding governmental election with ballot privacy. In *Proceedings of the 19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010*, pages 291–306. USENIX Association, 2010.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a Secure Voting System. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 354–368. IEEE Computer Society, 2008.

[CEA07] Jeremy Clark, Aleksander Essex, and Carlisle Adams. Secure and observable auditing of electronic voting systems using stock indices. In *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on*, pages 788–791. IEEE, 2007.

[CFH+07] Joseph A. Calandrino, Ariel J. Feldman, J. Alex Halderman, David Wagner, Harlan Yu, and William P. Zeller. Source code review of the Diebold voting system. In *Report commissioned as part of the California Secretary of State's Top-To-Bottom Review of California voting systems*, July 20, 2007.

[CFN+11] David Chaum, Alex Florescu, Mridul Nandi, Stefan Popoveniuc, Jan Rubio, Poorvi L. Vora, and Filip Zagórski. Paperless Independently-Verifiable Voting. In *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers*, volume 7187 of *Lecture Notes in Computer Science*, pages 140–157. Springer, 2011.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 1997.

[CH10] Jeremy Clark and Urs Hengartner. On the use of financial data as a random beacon. In *2010 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '10, Washington, D.C., USA, August 9-10, 2010*. USENIX Association, 2010.

[CH11] Jeremy Clark and Urs Hengartner. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In *Financial Cryptography and Data Security - 15th International Conference, FC 2011, Gros Islet, St. Lucia, February 28 - March 4, 2011, Revised Selected Papers*, volume 7035 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2011.

[Cha01] David Chaum. Surevote: Technical overview. In *Proceedings of the Workshop on Trustworthy Elections (WOTE'01)*, 2001.

[Cha03] David Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. In *Secure Electronic Voting*, volume 7 of *Advances in Information Security*, pages 211–219. Springer, 2003.

[Cha04] David Chaum. Secret-Ballot Receipts: True Voter-Verifiable Elections. *IEEE Security & Privacy*, 2(1):38–47, 2004.

[Cla11] Jeremy Clark. *Democracy Enhancing Technologies: Toward deployable and incoercible E2E elections*. PhD thesis, University of Waterloo, 2011.

[CLMS08] Liqun Chen, Hans Löhr, Mark Manulis, and Ahmad-Reza Sadeghi. Property-Based Attestation without a Trusted Third Party. In *Information Security, 11th International Conference, ISC 2008, Taipei, Taiwan, September 15-18, 2008. Proceedings*, volume 5222 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2008.

[CRS05] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. In *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139. Springer, 2005.

[CS11] Véronique Cortier and Ben Smyth. Attacking and Fixing Helios: An Analysis of Ballot Secrecy. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*, pages 297–311. IEEE Computer Society, 2011.

[CS14] Chris Culnane and Steve A. Schneider. A peered bulletin board for robust use in verifiable voting systems. In *Proceedings of the 27th IEEE Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 169–183. IEEE, 2014.

[Dan10] Dancho Danchev. Report: 48% of 22 million scanned computers infected with malware. http://www.zdnet.com/blog/security/report-48-of-22-million-scanned-computers-infected-with-malware/5365, 2010. Last accessed on 15 Aug 2014.

[DF89] Yvo Desmedt and Yair Frankel. Threshold Cryptosystems. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, 1989.

[DKR06] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006), 5-7 July 2006, Venice, Italy*, pages 28–42. IEEE Computer Society, 2006.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.

[dMPQ09] Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios. In *2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '09, Montreal, Canada, August 10-11, 2009*. USENIX Association, 2009.

[ED10] Saghar Estehghari and Yvo Desmedt. Exploiting the Client Vulnerabilities in Internet E-voting Systems: Hacking Helios 2.0 as an Example. In *2010 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '10, Washington, D.C., USA, August 9-10, 2010*, pages 1–9. USENIX Association, 2010.

[Ess12] Aleksander Essex. *Cryptographic End-to-end Verification for Real-world Elections*. PhD thesis, University of Waterloo, 2012.

[FMM+02] Jun Furukawa, Hiroshi Miyauchi, Kengo Mori, Satoshi Obana, and Kazue Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. In *Financial Cryptography, 6th International Conference, FC 2002, Southampton, Bermuda, March 11-14, 2002, Revised Papers*, volume 2357 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2002.

[FS86] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[FS01] Jun Furukawa and Kazue Sako. An Efficient Scheme for Proving a Shuffle. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. Springer, 2001.

[Fur04] Jun Furukawa. Efficient, verifiable shuffle decryption and its requirement of unlinkability. In *Public Key Cryptography - PKC 2004, 7th International Workshop on Theory and Practice in Public Key Cryptography, Singapore, March 1-4, 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 319–332. Springer, 2004.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[Gam85] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Gjø11] Kristian Gjøsteen. The Norwegian internet voting protocol. In *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers*, volume 7187 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal of Computing*, 18(1):186–208, 1989.

[GRBR13] Gurchetan S. Grewal, Mark D. Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. Caveat Coercitor: Coercion-Evidence in electronic voting. In *2013 IEEE Symposium on Security and Privacy, (S&P 2013), Berkeley, CA, USA, May 19-22, 2013*, pages 367–381. IEEE Computer Society, 2013.

[GRCC15] Gurchetan S. Grewal, Mark D. Ryan, Liqun Chen, and Michael R. Clarkson. Du-Vote: Remote electronic voting with untrusted computers. In *Proceedings of the 28th IEEE Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 155–169. IEEE, 2015.

[HBH10] Stuart Haber, Josh Benaloh, and Shai Halevi. The Helios E-voting demo for the IACR, 2010. Available at `http://www.iacr.org/elections/eVoting/heliosDemo.pdf`.

[HK13] Rolf Haenni and Reto E. Koenig. Voting over the Internet on an Insecure Platform. In *Design, Development and use of Secure Electronic Voting Systems*, pages 62–75. IGI Global, 2013.

[HLW11] Sven Heiberg, Peeter Laud, and Jan Willemson. The Application of I-Voting for Estonian Parliamentary Elections of 2011. In *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers*, volume 7187 of *Lecture Notes in Computer Science*, pages 208–223. Springer, 2011.

[HS00] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 539–556. Springer, 2000.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the*

Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005, pages 61–70. ACM, 2005.

[JJ00] Markus Jakobsson and Ari Juels. Mix and Match: Secure Function Evaluation via Ciphertexts. In *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2000.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making Mix Nets Robust for Electronic Voting by Randomized Partial checking. In *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*, pages 339–353. USENIX, 2002.

[JRSW04] David Jefferson, Aviel D. Rubin, Barbara Simons, and David Wagner. A security analysis of the secure electronic registration and voting experiment (SERVE), January 2004. Available at `http://www.servesecurityreport.org/`.

[KR05] Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the Applied Pi Calculus. In *Programming Languages and Systems, 14th European Symposium on Programming,ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200. Springer, 2005.

[KR16] Steve Kremer and Peter B. Rønne. To Du or not to Du: A Security Analysis of Du-Vote. In *2016 IEEE European Symposium on Security and Privacy, (EuroS&P 2016)*, 2016.

[KRS10] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 2010.

[KSRW04] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *2004 IEEE Symposium on Security and Privacy (S&P 2004), 9-12 May 2004, Berkeley, CA, USA*, pages 27–40. IEEE Computer Society, 2004.

[KTV12] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash attacks on the verifiability of e-voting systems. In *IEEE Symposium on Security and Privacy, S&P 2012, 21-23 May 2012, San Francisco, California, USA*, pages 395–409. IEEE Computer Society, 2012.

[LBD+03] Byoungcheon Lee, Colin Boyd, Ed Dawson, Kwangjo Kim, Jeongmo Yang, and Seungjae Yoo. Providing receipt-freeness in mixnet-based voting protocols. In *Information Security and Cryptology - ICISC 2003, 6th International Conference, Seoul, Korea, November 27-28, 2003, Revised Papers*, volume 2971 of *Lecture Notes in Computer Science*, pages 245–258. Springer, 2003.

[Luh60] P. Luhn. Computer for verifying numbers, August 23, 1960. US Patent 2,950,048.

[MN06] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 373–392. Springer, 2006.

[Nat14] Survation Surveying The Nation. Londestone Political Survey. http://survation.com/wp-content/uploads/2014/04/Full-Voters-NonVoters-Report.pdf, 2014. Last accessed on 15 Aug 2014.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8,CCS 2001.*, pages 116–125. ACM, 2001.

[Nef04] C. Andrew Neff. Practical high certainty intent verification for encrypted votes, 2004.

[NFVK13] Stephan Neumann, Christian Feier, Melanie Volkamer, and Reto E. Koenig. Towards a practical JCJ/Civitas implementation. In *Informatik 2013, 43. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Informatik angepasst an Mensch, Organisation und Umwelt, 16.-20. September 2013, Koblenz*, volume 220 of *LNI*, pages 804–818. GI, 2013.

[Nor14] Internet Voting Pilot to Be Discontinued. `https://www.regjeringen.no/en/aktuelt/Internet-voting-pilot-to-be-discontinued/id764300/`, 2014. Government.no. Ministry of Local Government and Modernisation [Online; accessed 04-Jan-2015].

[NV12] Stephan Neumann and Melanie Volkamer. Civitas and the Real World: Problems and Solutions from a Practical Point of View. In *Seventh International Conference on Availability, Reliability and Security, Prague, ARES 2012, Czech Republic, August 20-24, 2012*, pages 180–185. IEEE Computer Society, 2012.

[Oka96] Tatsuaki Okamoto. An electronic voting scheme. In *IFIP World Conference on IT Tools*, pages 21–30, 1996.

[Oka97] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings*, volume 1361 of *Lecture Notes in Computer Science*, pages 25–35. Springer, 1997.

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 316–337. Springer, 2003.

[Ped91a] Torben P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[Ped91b] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer, 1991.

[PIK93] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer, 1993.

[RBH+09] Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à Voter: a voter-verifiable voting system. *IEEE Transactions on Information Forensics and Security*, 4(4):662–673, 2009.

[RG13] Mark D. Ryan and Gurchetan S. Grewal. Digital Democracy lets you write your own laws. http://www.theconversation.com/digital-democracy-lets-you-write-your-own-laws-21483, 2013. The Conversation website [Online; accessed 04-Jan-2015].

[RG14] Mark D. Ryan and Gurchetan S. Grewal. Internet voting: coming to a computer near you, though more research is needed to eliminate the risks. http://www.democraticaudit.com/?p=3149, 2014. Democratic Audit blog [Online; accessed 04-Jan-2015].

[RG15] Mark D. Ryan and Gurchetan S. Grewal. Online voting is convenient, but if the results aren't verifiable it's not worth the risk. https://theconversation.com/online-voting-is-convenient-but-if-the-results-arent-verifiable-its-not-worth-the-risk-41277, 2015. The Conversation website [Online; accessed 21-July-2015].

[RS06] Peter Y. A. Ryan and Steve A. Schneider. Prêt à Voter with re-encryption mixes. In *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings*, volume 4189 of *Lecture Notes in Computer Science*, pages 313–326. Springer, 2006.

[RT09] Peter Y. A. Ryan and Vanessa Teague. Pretty Good Democracy. In *Security Protocols XVII, 17th International Workshop, Cambridge, UK, April 1-3, 2009. Revised Selected Papers*, volume 7028 of *Lecture Notes in Computer Science*, pages 111–130. Springer, 2009.

[Sam12] Ted Samson. Malware infects 30 percent of computers in U.S. http://www.infoworld.com/t/cyber-crime/malware-infects-30-percent-of-computers-in-us-199598, 2012. Last accessed on 15 Aug 2014.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[SFD+14] Drew Sprinall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. Security analysis of the Estonian internet voting system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 703–715. ACM, 2014.

[SHD10] Oliver Spycher, Rolf Haenni, and Eric Dubuis. Coercion-resistant hybrid voting systems. In *Electronic Voting 2010, EVOTE 2010, 4th International*

Conference, Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting.CC, July 21st - 24th, 2010, in Castle Hofen, Bregenz, Austria, volume 167 of *LNI*, pages 269–282. GI, 2010.

[SHKS11] Michael Schläpfer, Rolf Haenni, Reto E. Koenig, and Oliver Spycher. Efficient vote authorization in coercion-resistant internet voting. In *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers*, volume 7187 of *Lecture Notes in Computer Science*, pages 71–88. Springer, 2011.

[SKHS11] Oliver Spycher, Reto E. Koenig, Rolf Haenni, and Michael Schläpfer. A new approach towards coercion-resistant remote e-voting in linear time. In *Financial Cryptography and Data Security - 15th International Conference, FC 2011, Gros Islet, St. Lucia, February 28 - March 4, 2011, Revised Selected Papers*, volume 7035 of *Lecture Notes in Computer Science*, pages 182–189. Springer, 2011.

[SLC+11] Steve Schneider, Morgan Llewellyn, Chris Culnane, James Heather, Sriramkrishnan Srinivasan, and Zhe Xia. Focus group views on Prêt à Voter 1.0. In *2011 International Workshop on Requirements Engineering for Electronic Voting Systems, REVOTE 2011, Trento, Italy, August 29, 2011*, pages 56–65. IEEE, 2011.

[Smi05] Warren D Smith. New cryptographic election protocol with best-known theoretical properties. In *Proceedings of Workshop on Frontiers in Electronic Elections*, 2005.

[Uni15] Princeton University. Helios Princeton Undergraduate Elections. `https://princeton.heliosvoting.org`, 2015. [Online; accessed 21 July 2015].

[WAB07] Stefan G. Weber, Roberto Araujo, and Johannes A. Buchmann. On coercion-resistant electronic elections with linear work. In *Proceedings of the the Second International Conference on Availability, Reliability and Security, ARES 2007, The International Dependability Conference - Bridging Theory and Practice, April 10-13 2007, Vienna, Austria*, pages 908–916. IEEE Computer Society, 2007.

[WWH+10] Scott Wolchok, Eric Wustrow, J. Alex Halderman, Hari K. Prasad, Arun Kankipati, Sai Krishna Sakhamuri, Vasavya Yagati, and Rop Gonggrijp. Security analysis of India's electronic voting machines. In *Proceedings of the*

17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010, pages 1–14. ACM, 2010.

[WWIH12] Scott Wolchok, Eric Wustrow, Dawn Isabel, and J. Alex Halderman. Attacking the Washington, D.C. internet voting system. In *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, volume 7397 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2012.

[ZCC+13] Filip Zagórski, Richard Carback, David Chaum, Jeremy Clark, Aleksander Essex, and Poorvi L. Vora. Remotegrity: Design and use of an end-to-end verifiable remote voting system. In *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, volume 7954 of *Lecture Notes in Computer Science*, pages 441–457. Springer, 2013.