Automated Privacy Verification of Voting Systems

Murat Moran

A thesis submitted to the
University of Surrey for the degree of
Doctor of Philosophy

University of Surrey
Faculty of Engineering and Physical Sciences
Department of Computing

Declaration

This thesis and the work to which it refers are the results of my own efforts. Any ideas, data, images or text resulting from the work of others (whether published or unpublished) are fully identified as such within the work and attributed to their originator in the text, bibliography or in footnotes. This thesis has not been submitted in whole or in part for any other academic degree or professional qualification. I agree that the University has the right to submit my work to the plagiarism detection service TurnitinUK for originality checks. Whether or not drafts have been so-assessed, the University reserves the right to require an electronic version of the final document (as submitted) for assessment as above.

Murat Moran
Dec 2013

Abstract

Voting systems aim to provide trustworthiness in elections; however, they have always been a target of malicious behaviours due to difficulties in designing such complex systems and the enormous value of controlling the election results, causing unfair election outcome, loss of personal privacy and trust in democracy. This thesis aims to shed light on how voting systems, in particular, paper-based ones can be evaluated so as to provide a better level of confidence in their trustworthiness.

This thesis advances the evaluation of the paper-based voting systems using formal methods with automated analysis. In analysis of security protocols, the formal definitions of protocol requirements need to be constructed precisely. To this end, a formal framework regarding the anonymity requirement has been given and demonstrated to be appropriate for the analysis of voting systems. Similarly, it has been demonstrated that the assumptions under which voting systems are secure should be well-defined for a rigorous security analysis with the automated analysis of the ThreeBallot voting system. Moreover, a novel approach has been proposed to analyse cryptographic voting systems under a passive attacker model using the Prêt à Voter voting system as case study. Finally, an active powerful attacker has been adapted into the analysis of voting systems, and an automated formal analysis of vVote voting system has been conducted, which is under development for use in Victorian Electoral Commission (VEC) elections, Australia in 2014. With the analyses of voting systems performed in this thesis, the formal approach developed here has been demonstrated to be successful in the automated analysis of such complex systems using the process algebra, Communicating Sequential Processes (CSP), and the model checker, Failures-Divergence Refinement (FDR).

Acknowledgements

First, and foremost, I would like to thank my supervisors James Heather and Steve Schneider. James has dedicated a considerable amount of time and energy to the supervision of my research. He has helped me to explore new ideas by critically evaluating my work and giving practical advice on every problem encountered. Steve has provided the fundamental skills of our discipline as well as the inspiration and motivation via regular and irregular meetings. He never made me feel that he was just my second supervisor and provided a good effort in guidance of my research.

I appreciate the financial support provided by The Ministry of National Education, Republic of Turkey.

I would like to thank David Williams for his priceless help in understanding the modelling and analysis with CSP. I look forward to our future collaboration and continued friendship. I must also thank the members of the Trustworthy Voting Systems (TVS) at the University of Surrey, Chris Culnane, Zhe Xia (Joson), Sriram Srinivasan, Stathis Stathakidis and Harshana Liyanage, the SeRTVS Project in Luxembourg Peter Ryan, Hugo Jonker and Dalia Khader as well as Mark Ryan, Sergiu Bursuc and Myrto Arapinis from University of Birmingham for their presentations and discussions over project meetings, which helped me understand different ideas on formal methods and voting system analysis. In addition, I would also like to thank the CryptoForma members, especially Eerke Boiten, for funding me to attend those hugely beneficial meetings and workshops.

I owe a debt of gratitude to Peter Ryan and Helen Treharne, for their kindness in agreeing to examine this thesis.

I am truly grateful to Stefan Stafrace for his great friendship as an endless source of advice and for giving me motivational talks during my PhD, general chats on anything, and for the numerous hours spent in the Five and Lime, and in Legion and for all the laughter we had over a drink or nine. I would also like to thank my friends Emrah, Fani, Kenan, Yigit and my cousin Emrah, it can truly be said that they are never too far away. Maria-Eleni deserves particular consideration

for the unconditional and incomparable support, understanding and care shown over the last two years.

I am also grateful to my mum and dad—Gültaze and Selman Moran—for their enormous unconditional support on every decision I made and care even though I am far away home; sisters Serpil and Semiha, and brother in law Erkan and Yaşar, respectively, for their remote support taking many forms and with their delicious food, which kept me going and finally my nieces and nephew, Ceren, Cansu, Ceyda and Caner for their love and laughter over Skype and msn talks and for giving me a good reason to go back home.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the main aims of voting systems is to provide trustworthiness in elections; however, they have always been a target of malicious behaviours due to difficulties in designing such complex systems and the enormous value of controlling the election results. This thesis aims to shed light on how voting systems, in particular paper-based ones, can be evaluated so as to provide a better level of confidence in their trustworthiness.

In the early stages of democracy, the voting procedure took the form of a show-of-hands, i.e., voters simply put forward their preference in public. However, as this method allowed voters to be influenced by others, a new system based on ballot papers was introduced in Australia for presidential elections in 1858 in which voters voted in the private environment of a booth. This was the election to introduce the concept of privacy and it caught on across the world's democracies. However, although voters obtained a better level of secrecy, this meant a shift in verifiability of an election to establishing whether the contents of the ballot box had been interfered with in any way. Subsequently, through mechanization and most recently, computerization of elections voting systems have evolved with (e.g., lever machines, punch cards, optical scans and Direct Recording by Electronics (DRE) machines). Nonetheless, from the literature, despite these new technologies, it is apparent that there are problems with voting systems that have yet to be resolved [DSS03, Sch04, Gum05, Goo08, Bla09].

In 2000, in the United States presidential elections, Al Gore, the Democrat candidate was given a negative vote count of $-16,022$ votes in Florida, Volusia County due to equipment failure and subsequently this controversial election was won by the Republican, George W. Bush, with a narrow margin of just 537 votes. Similarly, in the Florida 2004 presidential election, $58,000$ postal ballots went missing [BBC04]. These are only two examples among many others of election frauds or failures in the history of democracies. As a result of such problems,

a number of countries have either abandoned or suspended their e-voting systems (e.g., California and Florida states in USA in 2006, Netherlands in 2008, Germany in 2009 and Ireland in 2009).

Many voting systems have been proposed over the last two decades, which aim to ensure strict requirements so as to guarantee their trustworthiness [Cha81, FOO92, Nef01, BG02, CRS05, Riv06, CEC+08, CCM08, Adi08]. The convention in all except [Riv06] is to use cryptography to provide voter privacy, anonymity, integrity and so forth. However, encryption does not always mean guaranteed privacy in security protocols. For example, Lowe [Low95] demonstrates that the Needham-Schroeder public-key protocol [NS78] is indeed insecure 17 years since it was first proposed. In addition to concerns regarding security protocols, there has been increasing concern that all employed voting systems need to be verified. Recently, research has focused on the formal analysis of trustworthy voting systems [NAN05, DKR06, BHM08, DKR09, DKR10, SRKK10, Smy11]. However, there is still a lack of mature methodology that can be used to provide a foundation for automated verification of voting systems due to difficulties entailed in modelling and analysis. These obstacles are inherited from the security protocols, whereby [RSG+00]: the security properties can be too highly subtle to define precisely; describing the model in a hostile environment can be too complex; defining any intruder's ability is extremely challenging and the concurrency involved in security protocols further complicates analytical endeavours. This increased complexity has meant that hand proof verification of voting systems is losing its effectiveness and hence, the need for automated protocol verification.

The substantive aim in this thesis is to investigate in current definitions of privacy-related properties, in general, and to redefine them so as to facilitate formal analysis of any voting system. Formally defining such properties is not straightforward as there can be various requirements for different scenarios. Regarding this, [SS96] presents a formal definition in relation to the anonymity for security protocols, which this researcher contends does not apply in the case of such voting systems. In order to ensure a rigorous security protocol analysis, the first task is to develop a framework and this is to be achieved by drawing upon the definitions of privacy-related properties and the passive and active intruder models. This framework will then be tested to find out how effective it is for identifying the vulnerabilities of certain voting systems. That is, the goal is to establish the best way to model and analyse validation of voting systems in terms of their trustworthiness. Moreover, because voting systems differ in terms of their designs aimed at providing privacy, the belief here is that the framework adapted is capable of evaluating the effectiveness of these diverse systems in protecting such as privacy and secrecy.

1.1 Formal Methods

Voting systems can be construed similarly to security protocols, which involve cryptographic primitives designs and protocols. In the literature, provable security and formal methods are the two main techniques used for security protocol analysis. The former takes the form of mathematical proof regarding a security protocol, where the aim is to reduce the security of the modelled system to a mathematically hard problem in the presence of an adversarial activity. However, systems involving cryptographic protocols have become increasingly complex and consequently such attacks do not only depend on flaws and weaknesses in the underlying cryptographic algorithms. That is, provable security is not sufficient to cover all possible nonintuitive intruder attacks [Mea03]. As a result, formal methods, defined by Meadows [Mea03] as

> "A combination of a mathematical or logical model of a system and its requirements, together with an effective procedure for determining whether a proof that a system satisfies these requirements have been introduced by way of improvement on simple mathematical approaches."

Meadows also highlights these treatments' usefulness in cryptographic protocol analysis as they provide both a thorough analysis of the different paths that an intruder can take as well as specifying precisely the environmental assumptions that have been made. In this thesis, formal methods are deemed suitable for rigorous specification descriptions and modelling and analysis of the various voting systems, in particular, because they have been used effectively for much security protocol analysis since being proposed by Dolev and Yao [DY83].

These authors were the first to formalise a hostile environment in which a protocol can run concurrently, and the intruder can, for instance: overhear, modify, block, replay or fake the messages on the communication channels. Since then research has focused on model checking tools, such as Failures-Divergence Refinement (FDR) [GGH$^+$]. Regarding this, Lowe implemented a formal analysis of the Needham-Schroeder Public-Key (NSPK) protocol for the first time using FDR and process algebra in the form of Communicating Sequential Processes (CSP) [Low95]. The goal of this protocol was to set up mutual authentication for two agents by a trusted server using public-key cryptography. However, Lowe demonstrated that it is not secure when an honest agent commences a session with an intruder. That is, he found that the subsequent attack is not related to the cryptographic algorithm that the NSPK is based on, but to the protocol as a whole and that consequently, if it were to be used with any other public-key protocol would still be insecure. In other words, even when this cryptographic

algorithm is secure, the protocol relying on these primitives may not be so. Similarly, voting systems based on either cryptographic or non-cryptographic primitives may be vulnerable to any sorts of security attacks. As a result, the main challenge is to maintain the security properties of voting systems intact in the presence of malicious agents.

1.2 Defining Desired Properties and Protocol Analysis

Trustworthy voting systems can be required to provide a number of crucial properties such as: secrecy, anonymity, and verifiability. Hence, instead of saying that a voting system protocol is secure, it is checked whether each of any desired properties are satisfied or not. However, before this can be achieved it is to assign precise formal definitions to these properties, which is this researcher's opinion need to be focused. Security properties are generally defined in terms of reachability or indistinguishability. Regarding the former, the secrecy property can be defined as being whether malicious agents or an intruder can reach certain data in a protocol run. This kind of analysis requires checking all possible states of unauthorised access to see whether any lead to intruders or malicious agents knowing data that should remain secret. Indistinguishability is the term often used to investigate the anonymity property. Under this lens, the goal is to check whether an intruder can ever distinguish events performed by the protocol agents and hence, such analysis is made through a comparison between two protocol states.

A review of the existing literature reveals that research has focused on giving precise formal definitions of the security properties of trustworthy voting systems [KR05, COPD06, DKR06, BHM08, DKR09, DKR10, Smy11] in several paradigms, such as the applied pi calculus [FA02], a modular approach [HS04], epistemic logic [GHPv05, BRS07, LJP10], and probabilistic and non-deterministic approaches [CPP06, DPP07, BP05]. As the main challenge here is not only to give precise definition requirements, but also to automate the analysis, CSP is adopted owing to its maturity and suitability for automation with FDR. Additionally, although some of the earlier works on security analysis of voting systems have involved automating the analysis, formal language has not supported a mature full-automation in privacy analysis. Instead, a hand proof was given [KR05, BHM08, DRS08] or a compiler was written and their soundness needed to be proven [CS11, Smy11]—this author's understanding of the soundness is that when there is no error, there must be none (no false negatives)—or an inefficient tool was adopted [CCK12].

In process algebras, such as CSP, a protocol is modelled by a process, which may also be a composition of other processes. Ideally, the overall process reflects the same behaviour as protocol. Moreover, the agents in the protocol are also present in the individual processes, for instance, the intruder process. When the model is composed and the requirements are translated into CSP, then the protocol can be analysed under the assumptions made about the intruder. If the model checking against the specification fails, the model-checking tool produces counter-examples, which means that the system does not satisfy the specification. In the case that model checking is performed successfully, one can question about the correctness of the system model. Such possible errors in designing the desired behaviour of systems can be detected by further reasoning via various sanity checks written as specifications. If the system model behaves correctly for all possible cases, then one can be sure that the model is correct. However, it is not an easy task to try all the possible cases, but a number of necessary sanity checks can help in gaining confidence in the model.

1.3 Contributions that address limitations of existing literature

A key objective in this thesis is to focus on automated formal analysis of cryptographic and non-cryptographic voting systems. In particular, the contributions made in the forthcoming chapters match the objectives of: providing concise definitions of requirements for voting systems so as to eliminate the vagueness in the anonymity definitions in the existing literature, and hence to facilitate future automated analysis of such complex systems (Chapter 4); modelling and analysing a non cryptographic voting system (Chapter 5) to show the validity of the adopted approach, with particular emphasis on the importance of the assumptions when designing voting systems; modelling and analysing a paper-based cryptographic voting system (Chapter 6) to illustrate that the mechanised analysis of such systems can be conducted by the tool introduced in this work; and adapting a formal framework of a more powerful intruder model into voting systems protocol analysis (Chapter 7) so as to avoid attacks on new systems before they are deployed. In the remainder of the section the contributions of each of the empirical chapters of this thesis (Chapter 4 to 7) are presented.

Chapter 4. Anonymity of voters lies at the heart of the democratic process, for if the link between them and their vote is uncovered, then not only is the secrecy breached but also the integrity of the election is threatened, because votes could be bought, or voters coerced into supporting particular candidates. Moreover, as rigorous protocol analysis requires a concise formal definition of the

properties, the anonymity property also needs to be clearly defined. However, as pointed out above, formally defining such properties is not straightforward as these can vary for different scenarios. Regarding this, there have been number of research efforts towards formally defining the anonymity property in the context of voting systems. However, little work has been undertaken aimed at providing a foundation for the automated verification of such properties.

In the extant literature, Juels *et al.* [JCJ05] describe anonymity for elections in terms of provable security. Other definitions of the desired properties of voting systems have made use of formal methods, for instance, Delaune *et al.* [DKR06] defined privacy properties in terms of adaptive simulation in the pi calculus. These authors then discovered in [DKR09] that their previous work had undesirable properties, and proposed redefinition based on observational equivalence, but did not provide any means of automatic verification. Backes *et al.* [BHM08] proposed a new formalization of coercion resistance for remote voting protocols in terms of observational equivalence, which implies vote privacy; their verification is automated, but some human effort is still required when transforming equivalences in their definition into a pair of processes, which have the same structure but differ only by terms. Chothia *et al.* [COPD06] presented a framework for automatically checking anonymity using bisimilarity in the process algebraic language μCRL [BFG$^+$01], by analysing the voting scheme first given in [FOO92]. Their model, like that put forward for this research, uses a passive intruder; however, their use of bisimilarity can produce false positives (that is, false attacks), because it effectively allows the intruder to see not just what actions are taken but where internal choices are resolved. Bisimilarity is more efficient for checking [HHK95], but seems too strong a notion of process equivalence for this application. More recently regarding the pi calculus, Delaune *et al.* [DRS08] and Smyth [Smy11] attempted to verify the FOO voting system with the tool ProVerif [Bla01]; however, the soundness of the transformation they used has yet to be proven. However, Chadha *et al.* [CCK12] managed to verify the anonymity of the FOO voting system using a prototype, Active Knowledge in Security Protocols (AKISS), which was written in the OCaml programming language and implemented to check equivalences. Nevertheless, it turns out that this tool needs to be improved in terms of efficiency, as accepted in these authors' paper. Perhaps even more importantly, they were unable to determine precisely the termination of the saturation procedure, which is required for deciding trace equivalences, only being able to conjecture what it was.

In Chapter 4, formal definitions of anonymity properties for voting protocols are elicited using the process algebra CSP. To this end, first, a number of anonymity definitions in the literature are investigated and those claiming to represent *strong* and *weak* anonymity are tested in terms of their suitability for various voting

systems. It emerges that the weak anonymity definition can be used to verify the anonymity property of voting systems mechanically, but the strong form is not appropriate. It is generally accepted that the conventional ballot voting system* in the vast majority of cases provides anonymity, if this system when tested supports this perspective then it will provide validation for formal definitions of this aspect derived for this research.

Chapter 5. Rivest's ThreeBallot voting system is important because it aims to provide security (voter anonymity and voter verifiability) without requiring cryptography. Nevertheless, a valid vote consists of three individual ballots, making the system very difficult to verify by hand proof methods and hence, an efficient way of automatic analysis is highly desirable. Moreover, carefully and concisely defined underlying assumptions are also necessary for a rigorous formal analysis. Analysis of the ThreeBallot system is provided to demonstrate that concise meaning of these assumptions is crucial when developing a voting system.

In the literature, the ThreeBallot voting system has been subject of analysis of one sort or another many times since its publication [Str06b, Str06a, CEA07, App07, dMPQ07, TPR07, CKW08, HSS09, KTV11]. Perhaps the earliest of this was conducted by Strauss [Str06b, Str06a], who established the success probabilities of attacks for various numbers of candidates and voters in multiple races. That is, the various attacks against the system, in particular, reconstruction and pattern request attacks were considered. The experiments were coded in Python, using modelled elections with a number of races on a single multi-ballot form. Clark *et al.* [CEA07] also investigated ThreeBallot, and elicited that its multi-ballot form can reveal information that can compromise voter privacy. Moreover, a simulation-based analysis of this system was made by de Marneffe *et al.* [dMPQ07] using the universally composable security framework [Can01]. They found that ThreeBallot lacks an election fairness guarantee (by looking at some receipts a real world adversary is able to obtain an estimation of the election outcome). Additionally, a modified system protocol in which a voter chooses his/her receipt before expressing her preference was proposed in [dMPQ07]. This protocol was shown to guarantee election fairness, at the cost of some noise in the final tally, with the short ballot assumption (SBA)†, and an additional assumption that most of the receipts are not known to the adversary. One drawback, however, is that the voter cannot express his/her preference on the mini-ballot that she has chosen as her receipt, which makes voting more complicated. Statistical

*The conventional (also called classical or traditional) voting system is the system where the voter indicates her preferred candidate by marking a ballot form in a private environment (booth) located in polling stations, and casts her vote by dropping the ballot paper into a ballot box. After the election is closed, the cast votes are collected at a central place and tallied.

†The SBA states that the information content of a ballot should be low and this assumption is further investigated in Chapter 5.

results about the relation between the number of candidates in an election and the privacy level of this system were provided by Cichoń *et al.* [CKW08] as well as a critique on the effectiveness of Strauss' attacks. Cichoń *et al.* claimed that it is impossible to reconstruct voters' preferences in a single election run with two candidates with a 'reasonable number of voters'. However, the definition of weak anonymity used in [CKW08] is much different from this researcher's, given in [MHS12]. That is, considering that an individual mini-ballot can be used to construct two different multi-ballots cast for the same candidate, their definition seems necessary, but not sufficient. That is, the observer would notice that one of the voters is not able to vote for that candidate. A more theoretical work was carried out by Henry *et al.* [HSS09], who focused on a two-candidate race, and determined the secure ballot sizes under reconstruction and pattern requesting attack conditions. Finally, Küsters *et al.* [KTV11] computationally analysed the level of privacy offered by the ThreeBallot voting system and the proposed system by de Marneffe *et al.* [dMPQ07], and concluded that the latter provides better privacy than the original.

In Chapter 5, a CSP model of ThreeBallot is constructed and used to produce the first automated formal analysis of its anonymity property using FDR. During this process, it emerges that one of the crucial assumptions under which ThreeBallot operates, the Short Ballot Assumption, is highly ambiguous in the literature. Consequently, various plausible precise interpretations are put forward, but it is discovered that in each case, the interpretation is either unrealistically strong or else fails to ensure anonymity. Therefore, a version of the Short Ballot Assumption in relation to ThreeBallot that is realistic but still provides a guarantee of anonymity is adopted.

Chapter 6. The definition of anonymity for voting systems and the tool used for model checking should be compatible with the automated analysis of cryptography-based voting systems, which use such features as: symmetric, asymmetric and homomorphic encryptions as well as digital signatures and mixnets.

In the literature, although automated formal analysis of Prêt à Voter does not exist, this researcher is aware of a few full system threat analyses of it. For example, Karlof *et al.* [KSW05] presented an analysis of Chaum's scheme [Cha04], on which the Prêt à Voter system is based. Their threat analysis mainly covered those from DREs, talliers, and outside coercive parties as well as collusions between these. They identified a number of vulnerabilities, for instance, subliminal channel attacks, denial of service and message reordering attacks and social engineering attacks caused by unreliable, but honest participants, such as voters. Another interesting work more specific to Prêt à Voter is that of Ryan and Peacock [RP05, RP10]. They performed a system perspective analysis of Prêt à Voter

by first investigating Karlof *et al.*'s observations and describing further vulnerabilities and threats, for instance, the doll matching, side-channel and kleptographic attacks as well as the undermining of public confidence. Finally, Jonker *et al.* [JMP09] proposed a formal framework for quantifying voter-controlled privacy and an analysis of Prêt à Voter was conducted in [Jon09] using this framework. Although, this analysis did not cover a complete modelling of Prêt à Voter and was not automated, they illustrated that their framework could be used to determine the privacy level of Prêt à Voter. Following that Jonker and Pang [JP11] extended the framework in [JMP09] with the model of a bulletin board and by capturing the coercion-resistance property, and they applied the extended framework to the Prêt à Voter voting system to measure loss of privacy.

In Chapter 6, the ability of the chosen methodology of automated analysis of cryptographic voting systems with a paper-based trustworthy voting system Prêt à Voter [RS06] that aims to provide anonymity based on mixnets and cryptography is investigated. Subsequently, the first-fully automated verification of this voting system from an observer point of view is presented. Moreover, a number of conspiracy theories, such as the election authority's colluding with the intruder, are examined to see if anonymity is preserved under such conditions.

Chapter 7. The more powerful the intruder, the better the security analysis required, as they can act in an increasing number of different ways. Hence, a natural next step is to model a more powerful intruder along the lines of a Dolev-Yao one [DY83], who is in control of the network, and can mount active attacks. This type of intruder not only observes a protocol run, but also interacts with the protocol participants, overhears communication channels, intercepts and spoofs any messages that he/she has learned or generated from any prior knowledge.

In the literature, there have been a number of attempts for automated anonymity verification of voting systems, which have deployed the Dolev-Yao intruder model. For example, Backes *et al.* [BHM08] analysed voting systems mechanically in terms of verifiability properties. However, no automated analysis of anonymity property was provided as the ProVerif employed was ineffectual for coping with equivalences, and hence, a hand-proof was required. Similarly, Delaune *et al.* [DRS08, DKR10] and also Smyth [Smy11] verified vote privacy of the FOO voting system with an additional compiler (ProSwapper), but these lacked proof of its soundness. Moreover, Chadha *et al.* [CCK12] managed to verify the anonymity of the FOO voting system using a prototype, AKISS, which was written in the OCaml programming language and implemented to check equivalences. However, the tool adopted was inefficient, and an important part of the analysis, the termination of the saturation procedure as required for deciding trace equivalences, was conjectured.

In Chapter 7, intruder capabilities are expanded with the adaptation of a full Dolev-Yao [DY83] intruder model, which is then modified with additional assumptions. For this purpose, the lazy spy (perfect spy) intruder model [RG97] in relation to voting systems is drawn upon, as it is very efficient in terms of cutting down unnecessary states as well as being flexible for usage with any other privacy related properties. In order to demonstrate, the suitability of this intruder model for evaluating voting systems, the vVote voting system, a promising real-world e-voting system is, subsequently analysed. In this part of the investigation, the secrecy and anonymity properties are covered as well as it being shown that a generic voting system can be analysed effectively using lazy spy in CSP with the FDR model checker.

1.4 Publications

Our contributions listed above were published in the following journal and conference proceedings, upon which the thesis is partly based.

[MHS12]: Murat Moran, James Heather, and Steve Schneider. Verifying anonymity in voting systems using CSP. *Formal Aspects of Computing*, pages 1–36, 2012

[MHS13]: Murat Moran, James Heather, and Steve A Schneider. Automated anonymity verification of the ThreeBallot voting system. In *IFM*, pages 94–108, June 2013

[MH13]: Murat Moran and James Heather. Automated analysis of voting systems with Dolev-Yao intruder model. In *Automated Verification of Critical Systems AVOCS*, September 2013

1.5 Outline

The rest of this thesis is organised as below.

Chapter 1 has introduced the thesis by identifying the gap in the literature, thereby eliciting research problems on voting systems that need to be addressed.

Chapter 2 describes the necessary concepts for helping the reader to understand basic concepts covered throughout the thesis. Firstly in this regard, cryptographic primitives are introduced, on which some of the case study voting systems are based, (e.g., public-key encryption, digital signatures and mixnets). Secondly, election methods and the privacy properties of voting systems are investigated. Next, four paper-based, supposedly, trustworthy non-cryptographic (the conventional and the ThreeBallot voting systems), and cryptographic voting systems

(Prêt à Voter and vVote) are introduced. Finally, the syntax and semantics of the process algebra, CSP, are explained.

In Chapter 3, an analysis for a problem called Dining Cryptographers is provided in order to explain modelling in CSP and the anonymity concept in general. In addition, the NSPK security protocol is drawn upon to provide a stronger intruder model and analysed in terms of the secrecy property. The intruder model will be used for analysis of the vVote voting system using CSP and FDR in Chapter 7.

A formal framework for the anonymity definition using CSP is described in Chapter 4. Subsequently, using this definition, the conventional voting system is modelled and analysed, followed by presentation of the initial results of the automated privacy investigation into voting systems.

The suitability of the adopted framework is probed by modelling and automatically analysing a paper-based non-cryptographic trustworthy voting system, ThreeBallot, in a more efficient way than previously in Chapter 4. Additionally, strong evidence is provided regarding the aforementioned ambiguity of the short-ballot assumption and consequently, alternative definitions for this are provided that are more efficient and realistic given its underpinning meaning.

In Chapter 6 the research framework is extended by formalising some of the cryptographic primitives as well as presenting the first automated anonymity analysis of Prêt à Voter. This analysis highlights the ability of this framework in modelling and analysing cryptography-based voting systems. Moreover, there is analysis of this particular system under alternative assumptions, including misbehaving election authorities.

In Chapter 7 the Dolev-Yao [DY83] intruder model is adapted to voting systems that can be automated using FDR and use this framework to model and analyse the vVote voting system. Further, it is demonstrated that secrecy analysis of voting systems is effective when using this adapted framework.

Finally, in Chapter 8 the contributions of this thesis are reviewed, the limitations considered and suggestions for future research directions are put forward.

Note on experimental configuration. The experimental results regarding the verification times for the automated analysis of voting systems given in Chapter 5 6 and 7 were produced on a machine with Intel(R) Core(TM) i5 CPU 2.40GHz, and 1GB RAM. Additionally, FDR 2.94 academic version was used in all mechanical analysis.

Note on CSP models of the voting systems analysed in this thesis. The CSP models of the voting systems, from which the experimental results given in this thesis were produced can be downloaded from the author's personal webpage `http://muratmoran.wordpress.com/publications/` under the CSP codes ti-

tle. Additionally, they will be available on the departmental webpages `http://www2.surrey.ac.uk/computing/people/murat_moran/index.htm` and `http://epubs.surrey.ac.uk/id/eprint/804928`.

Chapter 2

Background

The aim of this chapter is to give enough detail for the reader to understand the basic concepts used in the remainder of the thesis rather than presenting an exhaustive survey on voting systems. Firstly, a brief background on cryptography used in the construction of voting systems provided and subsequently the deductions rules capturing the properties of cryptographic primitives based on [Sch96] in Section 2.1 are spelt out. These rules will help in the understanding of the basic ideas behind the modelling and analysis of security protocols using CSP and FDR model checker as given in Chapter 3. Moreover, an overview of voting systems, and their desired properties is presented in Section 2.2 as well as the electoral methods. Finally, a CSP notation along with the abstraction methods and semantics used in this thesis is provided in Section 2.3.

2.1 Cryptography

Cryptography is generally used as a medium to provide a secure communication between principals over a hostile network. If the intruder is not bounded, he can perform infinitely many attacks in order to break the cryptographic protocols' goal (e.g., key distribution or authentication). As in [Low95], the attacks are not necessarily based on the flaws in the cryptographic algorithm used, for they could be down to the protocol itself.

Generally, in formal methods, a strong assumption is made regarding cryptography that abstracts away the underlying cryptographic algorithm and its properties, i.e., its strengths and weaknesses. That is, the encryption and decryption are treated as symbolic operators. For instance, an encrypted message with the key (pk) can only be decrypted by the corresponding secret key (sk) holder. Moreover, although the cryptographic algorithms are not unbreakable under the assumption of an intruder with enough computing time and power, it is typically

assumed that the intruder is bounded by the computational power, and that the probability of him breaking the cryptographic algorithm is negligible. Therefore, the main focus here will be on the analysis of security protocols under secure cryptographic algorithms, in the sense that no one except the corresponding key holder is able to decrypt an encrypted message.

In Chapter 7, a structure is required that enables the intruder to deduce information from what he sees and what he already knows. Such derivations regarding cryptographic primitives are captured in terms of deduction rules by the entailment relation $\vdash$ as described in [Sch96]. The relation is defined as the relation between the set of messages that are known by, say the intruder, and those that can be generated, $\vdash$: $\mathbb{P}(MESSAGE) \times MESSAGE$. In other words, $\mathcal{M} \vdash m$ means that the message m can be deduced from the set $\mathcal{M}$ and the relation is closed under the following axioms (Table 2.1) for an information system.

$A1.$	if $m \in \mathcal{M}$ then $\mathcal{M} \vdash m$
$A2.$	if $\mathcal{M} \vdash m$ and $\mathcal{M} \subseteq \mathcal{M}'$ then $\mathcal{M}' \vdash m$
$A3.$	if $\mathcal{M} \vdash m_i$ for each $m_i \in \mathcal{M}'$ and $\mathcal{M}' \vdash m$ then $\mathcal{M} \vdash m$

Table 2.1: Axioms

The cryptographic messages can also be structured so that they can be generated. In Table 2.2 and throughout this thesis: symmetric encryption of a message m with the symmetric key k is denoted as $\mathsf{E}_k(m)$; public key encryption with the public key pk is shown as $\mathsf{E}_{pk}(m)$; similarly, signing the message with the secret key s_k is $\mathsf{S}_{sk}(m)$.

M1.	$\mathcal{M} \vdash m \wedge \mathcal{M} \vdash k$	$\Rightarrow$	$\mathsf{E}_k(m)$
M2.	$\mathcal{M} \vdash m \wedge \mathcal{M} \vdash pk$	$\Rightarrow$	$\mathsf{E}_{pk}(m)$
M3.	$\mathcal{M} \vdash m \wedge \mathcal{M} \vdash sk$	$\Rightarrow$	$\mathsf{S}_{sk}(m)$

Table 2.2: Deduction rules capturing cryptographic message construction

The properties of the encryption methods, symmetric-key and public-key cryptography, and the signatures can be captured as in Table 2.3, where $\mathsf{D}_k(\mathsf{E}_k(m))$ models symmetric decryption, $\mathsf{D}_{sk}(\mathsf{E}_{pk}(m))$ public key decryption, and $\mathsf{V}_{pk}(\mathsf{S}_{sk}(m))$ message extraction from a signature with the signing public key.

Table 2.4 illustrates some of the deduction rules regarding composable messages by any agents (in the models in this thesis, it will be an intruder). In other words, an agent can generate a fact or a message, if he knows a specific set of messages.

K1.	$\mathsf{D}_k(\mathsf{E}_k(m))$	$\vdash$	m
K2.	$\mathsf{D}_{sk}(\mathsf{E}_{pk}(m))$	$\vdash$	m
K3.	$\mathsf{V}_{pk}(\mathsf{S}_{sk}(m))$	$\vdash$	m

Table 2.3: Deduction rules covering cryptographic primitives (m is not necessarily a message, it could be any fact. For instance, in K3., m can be a hash of a message)

For instance, one possessing the encryption key k and message m can deduce, $\mathsf{E}_k(m)$.

SYM-ENC.	$\{k, m\}$	$\vdash$	$\mathsf{E}_k(m)$	$A1.M1.$
SYM-DEC.	$\{k, \mathsf{E}_k(m)\}$	$\vdash$	m	$A1.M1.K1.$
ASYM-ENC.	$\{pk, m\}$	$\vdash$	$\mathsf{E}_{pk}(m)$	$A1.M2.$
ASYM-DEC.	$\{sk, \mathsf{E}_{pk}(m)\}$	$\vdash$	m	$A1.M2.K2.$
SIGN-SIG.	$\{sk, m\}$	$\vdash$	$\mathsf{S}_{sk}(m)$	$A1.M3$
SIGN-EXT.	$\{pk, \mathsf{S}_{sk}(m)\}$	$\vdash$	m	$A1.M3.K3.$

Table 2.4: Derived deduction rules involving cryptographic primitives

2.1.1 Symmetric key Encryption

Symmetric key encryption is the class of cryptographic algorithms that use the same key for message encryption and ciphertext decryption (i.e., encryption and decryption key is the same key). Hence, in order for two parties to communicate securely, they need to share the same key. Such encryption is denoted here as $\mathsf{E}_k(m)$, where k is the shared secret and m is a message. Thus, it is expected that anyone possessing the shared key k can encrypt the message m, and decrypt the ciphertext $\mathsf{E}_k(m)$, as defined by the derivations SYS-ENC and SYS-DEC in Table 2.4.

In order to establish a secure communication with shared keys, it is assumed that the shared key is only shared between legitimate agents, and no one else. Hence, this requires their having a prior arrangement to agree on a shared key and this issue is partially resolved by public-key cryptography.

2.1.2 Public-key Encryption

Public-key cryptography (also known as asymmetric key cryptography) was first put forward by James H. Ellis, Clifford Cocks, and Malcolm Williamson at the Government Communications Headquarters (GCHQ) in the UK in 1973, which was kept secret. Later on, an asymmetric-key cryptosystem was designed by Diffie and Hellman [DH76] in 1976. This cryptographic system, unlike symmetric key cryptography, requires two different keys for encryption and decryption, a public key pk and secret (private) key sk, respectively. Hence, each agent has a pair of keys, (pk, sk), which are mathematically linked such that $\mathsf{D}_{sk}(\mathsf{E}_{pk}(m)) = m$ for a message m. The previous issue in symmetric key cryptography, where the agents need to share a key, is resolved in this system, as the public key of an agent, say pk_a for agent a, is public and hence, anyone can encrypt a message m using a's public key, subsequently sending it to a, also in public. Moreover, although, any potential intruder can see the encrypted message, he cannot decrypt it in a reasonable amount of time, unless he possesses the secret key sk_a and therefore, only the secret key holder can decrypt the message.

RSA

RSA [RSA78] is a ground breaking algorithm designed for public-key cryptography by Rivest, Shamir and Adleman. It is based on the difficulty of factoring large integers, which is computationally hard. In more detail, two distinct primes, p and q, and an integer e are chosen such that $1 < e < \phi(n)$ and $gcd(e, \phi(n)) = 1$, where ϕ is Euler's totient function and $\phi(n) = (p-1)(q-1)$. The values n and e will be publicly accessible. For each value that forms a corresponding secret key, an integer d is chosen, such that $e \cdot d = 1 \bmod \phi(n)$. Then, a message m can be encrypted with the equality $c = m^e (\bmod\ n)$ by anyone knowing the public values n and e and consequently, the decryption of the ciphertext c can be performed as $c^d = (m^e)^d = m^{e \cdot d} = m (\bmod\, n)$.

ElGamal

ElGamal [ElG84] is a non-deterministic public key algorithm and unlike the RSA, the encryptions of the same plaintext always result in different ciphertexts due to its probabilistic feature. The public and secret generation for each agent is performed by firstly choosing large random prime numbers p and q, such that $p = 2q + 1$ holds. Secondly, a generator g is chosen from the $\mathbb{G}_q$ subgroup of $\mathbb{Z}_p^*$—ElGamal can be generalised to work in any finite cyclic group, such as, the group of points on an elliptic curve over a finite field [MVO96]. Finally, a random secret value $x \in \mathbb{Z}_q^*$ is chosen. Thus, the triple (p, g, g^x) will form the public-key, and (p, g, x) the secret key of an agent. The subsequent encryption of a message, m, can be calculated as (g^r, mh^r), where $h = g^x$ and $r \in \mathbb{Z}_q^*$ is a random value.

Moreover, in order to decrypt the ciphertext and to obtain m, one should compute $(g^r)^{-x}mh^r$ using the secret value x as $(g^r)^{-x}m(g^x)^r = m$.

Table 2.4, contains the two rules ASYM-ENC and ASYM-DEC, thereby capturing the desired properties of public-key encryption systems in a symbolic way. In addition, public-key cryptography also allows the message m encrypted under the secret key to be extracted if it is re-encrypted under the corresponding secret key, i.e., $\mathsf{E}_{pk}(\mathsf{D}_{sk}(m)) = m$, which is useful for constructing digital signatures, as discussed next.

2.1.3 Digital Signatures

The digital signature of a message provides a guarantee for its authenticity and origin. A public key implementation of the digital signatures can also provide non-repudiation, should a dispute arises about the signer of the message (the signer of a message is not be able to successfully challenge the validity of the signature). Although, a signature can be created differently depending on the algorithm used, here it is considered that a signed message is created by encrypting the hash of a message with the signer's secret key sk and reversely, the signed fact is extracted with the signer's public-key pk. Hence, as $\mathsf{E}_{pk}(\mathsf{D}_{sk}(m)) = m$, the output of the verification will demonstrate whether the message was indeed signed with the corresponding secret key sk, and whether it has been tampered with or not. Signing a message and extraction of the message from a signed one are captured in Table 2.4 with the derived deduction rules, SIGN-SIGN and SIGN-EXT. According to these, an agent a can sign a message m with his secret key sk_a, and anyone knowing a's public key can verify m's authenticity and origin by checking whether $\mathsf{V}_{pk_a}(\mathsf{S}_{sk_a}(\mathsf{H}(m))) = \mathsf{H}(m)$ where $\mathsf{H}()$ is an agreed publicly known hash function. In the modelling approach to digital signatures in this thesis, the origin and the authenticity of a message, say $\mathsf{S}_{sk_a}(m)$, is ensured via the authenticated channels. That is, an agent is ready to accept any message in the expected form, e.g., if the agent b is waiting for a signed message from the agent a, as long as: the message is signed with sk_a, b believes that $\mathsf{S}_{sk_a}(m)$ is generated by a, and m is not tampered with.

2.1.4 Homomorphic Encryption

Homomorphic encryption is a property of cryptographic systems, which allows for computations to be made on ciphertexts without revealing the secret. That is, the encryption algorithm $\mathsf{E}()$ is homomorphic if given $\mathsf{E}(m_1)$ and $\mathsf{E}(m_2)$ one can obtain $\mathsf{E}(m_1 \odot m_2)$ for some operation $\odot$ and messages, m_1 and m_2. For instance, ElGamal shows a multiplicative homomorphic property as: for given two ciphertexts $\mathsf{E}_{(p,g,x)}(m_1) = (g^{r_1}, m_1h^{r_1})$ and $\mathsf{E}_{(p,g,x)}(m_2) = (g^{r_2}, m_2h^{r_2})$, where $h =$

g^x and $r_1, r_2 \in \mathbb{Z}_q^*$ and p and q are two large random prime numbers, one can show that $\mathsf{E}_{(p,g,x)}(m_1) \cdot \mathsf{E}_{(p,g,x)}(m_2) = (g^{r_1+r_2}, (m_1 \cdot m_2)h^{r_1+r_2}) \mathsf{mod}\, p = \mathsf{E}_{(p,g,x)}(m_1 \cdot m_2)$.

Here, the interest is in the homomorphic property because of its applicability to voting systems in terms of providing vote privacy. That is, as every vote is encrypted under the election public-key, authorities add all encrypted votes together, and decrypt and tally the final results without knowing individual ones. Prêt à Voter [RS06] and Civitas [CCM08] are among such voting systems.

2.1.5 Threshold Cryptosystem

A threshold technique allows for an initial secret key to be shared securely among a specified number of agents. More specifically, a cryptosystem is (t, n) threshold, if t or more agents recover the initial secret key sk by combining their individual secret shares, whereas fewer than t are unable to do so. In electronic voting, however, each threshold party produces a partial decryption of a secret using their key share, and the combination of these partial decryptions form a full decryption of the secret. For further reading on threshold cryptosystems, the reader is referred to [Ped92].

2.1.6 Re-encryption

Re-encryption allows for a ciphertext of a message to be encrypted again without needing the secret key x. That is, although the two ciphertexts encode the same plaintext, they differ in terms of randomness, which results in separate ciphertexts. In the case of the ElGamal public-key algorithm, if it is assumed that (g^r, mh^r) is the ciphertext that encodes the message m with the random value r, in order to re-encrypt the ciphertext, another value $r' \in \mathbb{Z}_q^*$ is chosen at random. Hence, the re-encryption of the message m can be produced with $(g^r g^{r'}, mh^r h^{r'})$. To decrypt this value, the secret value x is applied as before to produce $((g^{r+r'})^{-x}, mh^{r+r'})$, which will be equal to m. In voting, re-encryption is used to break the link between the voter and the ciphertext sent by her as they look different after re-encryption. An advanced version of this technique is used in the mixnets in order to anonymise the cast votes in an election.

2.1.7 Mixnets

Chaum [Cha81] designed mixnets for anonymous communications. The goal of a mixnet is to shuffle its input list in such a way that no one can trace the output list back, thus breaking the link between its inputs and outputs. This is done either by decrypting each element of the input list, which requires the mix possessing the corresponding secret key, or by re-encrypting them so that the input and output lists all look different, but still encode the same messages.

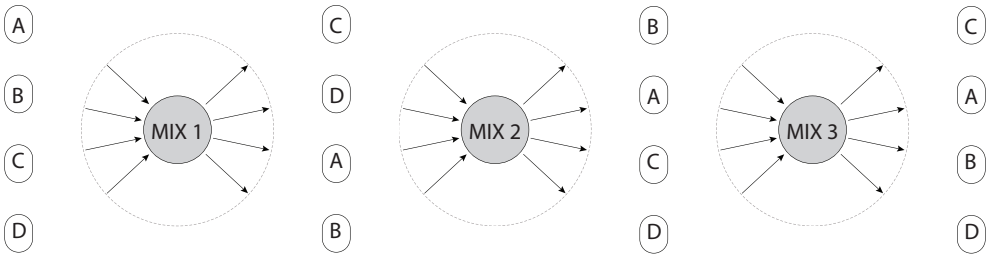

Figure 2.1: An example mix network (mixnet) with three mixes and four inputs

The only entity knowing the link is the mix itself. However, because a dishonest mix can reveal the anonymous links, a better approach involving multiple mixes was suggested by Chaum. According to this, only a collusion of all mixes could break the order of the elements in the input list. In Chaums's decryption mixnet, the inputs are the ciphertexts constructed as a layer for each mix server in the network that needs to be decrypted with the mix server's private key. Having decrypted the layer corresponding to its own public key, the mix permutes the list and forwards it to the next mix in the network. In the re-encryption type of mixnets [PIK94], the inputs are the messages encrypted under an algorithm, such as ElGamal [ElG84], which allows mix servers to re-encrypt them and after a re-encryption, the mix server permutes the inputs, subsequently outputting them to the next mix.

Figure 2.1 depicts a generic mixnet consisting of three mixes. The first takes a list of messages in the order $\langle A, B, C, D\rangle$ as its input—the list in its application to voting systems can be a list of encrypted candidate names, or onion values in order to provide an anonymous channel on which the voters cast their votes, thus hiding the link between the voters and their cast ballot forms. Subsequently, the output of the first mix becomes the input for the second, and following that the output of the second mix will be the input for the third. At the end of shuffling, the last mix server outputs the mixed order of the list, $\langle C, A, B, D\rangle$ in Figure 2.1.

In order to ensure whether a mix in the network is honest, a method called randomised partial checking (RPC) [JJR02] as illustrated in Figure 2.2 can be performed, which requires that each mix reveals half of its links between the elements in the input and output lists. Note that if a link to an element of the input list has already been revealed for a mix, the link for that element in the next mix should not be revealed. For instance, in Figure 2.2, the links for the inputs B and D are revealed for RPC in the first mix server, however, in the second mix, only the input and output links for A and C are revealed. Hence, an audit on these published links can be performed to verify that the mix is, indeed, honest, if the element has not been tampered with, without revealing the

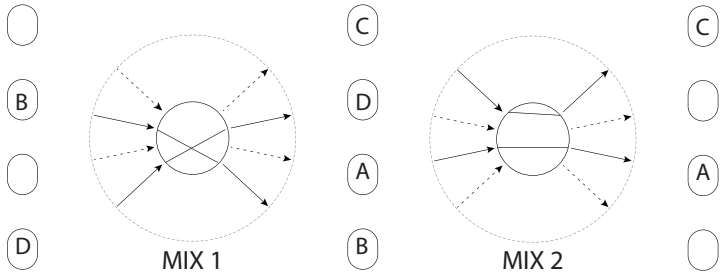

Figure 2.2: Randomised partial checking of a mixnet with two mixes and four inputs

input-output links of the mixnet. Moreover, if a mix acts dishonestly, the chance of its being caught is 2^{-1}.

2.1.8 Security Implications

Although re-encryption enables a ciphertext to be modified without the need of a secret key and plaintext would appear to be a useful mechanism for anonymising mixnets, this can also lead to undesired severe consequences as it can also be modified by the intruder in a meaningful way. This means that such cryptographic systems that use re-encryption (indeed inherent in any homomorphic cryptosystem), such as mixnets, are not secure under ciphertext indistinguishability under adaptive chosen ciphertext attack (IND-CCA2). However, they can achieve a weaker notion of security, such as, indistinguishability under chosen plaintext attack (IND-CPA).

2.2 Voting Systems

Under the traditional voting system, a trade-off is introduced between voter privacy and verifiability of the election. That is, as the ballot box keeps the link between the voter and her* ballot secret, it provides voter anonymity. However, the voter has no way of tracing his ballot or verifying whether it has been cast as intended and tallied correctly at the end of the election. Hence, the integrity of the election depends on the assumption of honest election officials (also called the chain-of-custody). Moreover, the honesty and correctness of the election officials are generally audited by the independent observers or the representatives from each political party in the election race. The deployment of electronic devices,

*Throughout the thesis, a voter will be referred to as female and an intruder as male. For instance: the voter takes "her" receipt, and the intruder will try to figure out how a voter has voted using "his" abilities.

such as lever and DRE machines, aimed to assist the voters' casting their votes, introduced a new trust in hardware and/or software used in the machines as well as in the election officials, as these are generally provided by private companies. In order to provide verification of such hardware and software components in voting systems, the voter-verified paper audit trail (VVPAT) was proposed by Rebecca Mercuri in 1992 [Mer92], which is a way of verifying a ballot being cast as intended by the voter that involves having a printer on the voting machine that prints out a receipt of the vote. However, although VVPAT provides a certain level of verification, it is insufficient for verifying all of the chain-of-custody points.

End-to-end (E2E) voting systems ensure that a voter is not only able to verify that her ballot is recorded as intended, but also counted as cast (included in the final tally). Moreover, anyone can verify that only eligible voters are allowed to cast ballots and the final tally is correct. This is generally done by producing a receipt for the voters, who can then check against the published receipts in a public domain like the web bulletin board (WBB). However, a receipt should not leak any information about how the voter has voted, as this could cause vote buying and coercion attacks as described in Section 2.2.2. The following are examples of paper-based E2E voting systems: non-cryptographic the ThreeBallot [Riv06], and cryptographic Prêt à Voter [Rya05] and vVote [BCH+12a, BCH+12b] voting systems. In this thesis, the focus is particularly on paper-based E2E in-person voting systems.

In the rest of this section, first, the terminology of voting system participants is introduced and then the desired properties of these systems provide the focus. Finally, the different types of electoral methods are outlined.

2.2.1 Terminology

In this subsection, some of the concepts of the voting systems and elections covered throughout this thesis are clearly explained so as to avoid ambiguity in their interpretation.

Voter: an eligible and registered individual attending to an election in order to express her preference.

Candidate: an individual running in an election process.

Vote: a representation of a voter's preference.

Ballot: a medium for voters to cast their votes.

Election Authority: the individuals responsible for the conduct of elections, e.g., preparing ballot forms and voter registration.

Electoral Officials: the individuals working during an election, who are responsible for voter authentication, vote collection and tallying, such as polling station workers.

Trusted Third Party (TTP): an independent and trusted organisation, which can help with the verification of some parts of an election as and when needed, for instance, auditing the correctness of the final tally and voting systems' components.

Intruder: a hostile individual deliberately trying to violate protocol objectives, and in this thesis two types are considered according to their destructive capabilities: passive and active intruders. More specifically, a passive intruder cannot only observe all public information but also the private channels that can be shared by dishonest agents. By contrast, an active one acts as described in Dolev-Yao [DY83] and the details on the capabilities of such intruders are given in Chapters 4 and 7.

Voting System Attacks

The followings are the attacks related to voting systems.

Vote-selling Attack: the voter being able to sell her vote, as she can prove to the intruder how she has voted.

Italian Attack: the intruder being able to ask the voter to fill a ballot in an uncommon way, so he can later check whether the unusual ballot appears on the bulletin board.

Reconstruction Attack: the intruder being able to reconstruct a cast ballot form with the information available. He may then find out how the voter has voted.

Forced-abstention Attack: the voter being forced to abstain from voting.

Randomisation Attack: the voter being forced to cast a ballot form in a random way.

Impersonating or Simulation Attack: the voter being forced to give her credentials to a coercer, who can then vote instead of her.

2.2.2 Desired Properties of Voting Systems

The design of trustworthy voting systems is not a trivial challenge since they depend heavily on a range of trust assumptions regarding such aspects as: the electoral officials, the voters and the hardware and software used. That is, obviously, these elements can be exploited with the intention of modifying the elec-

tion results or undermining their credibility by decreasing the voter's trust in the system. Consequently, in order to reduce the probability of elections being compromised, increased transparency and better monitoring are desirable during the electoral process. Thus, a trustworthy voting system can be considered as a challenge to get the level of transparency needed to gain confidence in the system, the degree of privacy that allows for the necessary freedom of choice to the voter and election integrity, all at the same time. Next, some of the desirable properties of voting systems are discussed in turn, i.e., privacy-related (anonymity, receipt-freeness, and coercion-resistance), verifiability aspects (eligibility, individual and universal) and integrity and fairness.

Anonymity

Anonymity of a voter means that the voting system should not reveal or give any information about how a particular vote was cast, often referred to as *vote-privacy*. For this to be the case, the identity of the voter associated with a ballot paper must be hidden. In order to clarify the distinction between secrecy and anonymity in voting systems, we can think of a situation where the intruder happens to link the voter to her ballot form, violating the anonymity property, but cannot interpret what her real vote is, preserving the secrecy of the vote. Therefore, the intruder can reveal the secrecy of a vote (knowing for whom the vote is), such as, by decrypting the encrypted vote, but still cannot link the vote to the voter casting it, preserving the anonymity property. There exist a numerous number of anonymity definitions, referring to it as either secrecy of a vote or the link between the voter and her vote. This is a subject covered in detail in Chapter 4 where the various definitions of anonymity in the literature are drawn upon in order to derive a concise formal definition regarding this requirement.

Receipt-Freeness

Receipt-freeness [BT94] ensures that a voter does not possess any information (e.g., receipt) that can be used by any third party to show how she has voted. A violation of this requirement may end up the voter being encouraged to sell her vote to a coercer (vote-selling attack). For instance, although the Prêt à Voter voting system provides a receipt for the voters for verifiability purposes, it does not illustrate how she has voted.

Coercion-Resistance

Essentially, coercion can be described as a voter being threatened or forced to act according to the instructions she has been given by a coercer. The coercion can be that the voter is forced to vote for a particular candidate, but it can take various forms, such as, *forced-abstention*, *randomisation*, and *impersonating*

or simulation. The corollary of this, coercion-resistance [JCJ05], refers to the coercer failing to have any evidence as to how a voter has voted, even if she has interacted with him.

Individual, Universal and Eligibility Verifiability

Individual verifiability allows a voter to verify whether her vote has been *cast-as-intended* and *counted-as-cast* at the end of the election.

By contrast, universal verifiability implies that any independent organisations, party or charity are able to verify that the election result is correct at the final tally, which is indeed the sum of all valid votes that are *cast-as-intended*.

Eligibility verifiability ensures that only eligible/legitimate voters can vote, and that they should be allowed to vote only once. The concept is introduced in [DKR10] as "anyone can check that each vote in the election outcome was cast by a legitimate voter and there is at most one vote per voter".

Integrity

Integrity (also called correctness or accuracy) ensures that the valid cast ballots are counted correctly in the final tally.

Fairness

The term *fairness*, refers to the situation that no partial results are revealed before the tallying phase, which otherwise would affect the voters' preferences during the election.

The attacks on voting systems listed in the previous subsection aim to break the privacy related properties; anonymity, receipt-freeness and coercion-resistance, which are mainly focused on this thesis. An attack may have impact on several privacy requirements depending on the reason to the attack. For instance, vote-selling may occur due to the voting system itself (violation of anonymity) or voter receipts (violation of receipt-freeness) or a interaction with a powerful coercer during vote casting (violation of coercion-resistance). Moreover, reconstruction and Italian attacks violate receipt-freeness property of voting systems because of the information leakage in voter receipts. Finally, the forced-abstention, randomisation, and impersonating or simulation attacks require the coercer communicate with the intruder at any time during the election period violating coercion-resistance property. However, because of the relation between privacy properties—coercion-resistance implies receipt-freeness and receipt-freeness implies anonymity (vote privacy) [DKR09]—if a voting system is protected against the attacks targeting coercion-resistance, it should also be immune to the receipt-freeness and anonymity attacks.

In this thesis, the focus will be mainly on privacy properties, anonymity, receipt-freeness and coercion-resistance, and on the attacks against these voting system requirements.

2.2.3 Electoral Methods

There are numerous electoral methods that are used in different countries and on various occasions. They mostly vary according to the number of winners, e.g., single and multiple-winner electoral methods. First-past-the-post (FPTP) or plurality voting is probably the most common single-winner electoral method and under this method, a candidate should receive more votes than any other in order to win. Additionally, each voter is allowed to vote for only one candidate. Generally, an FPTP ballot paper consists of a list of candidates and a marking box for each, with the voter just marking one of the boxes that corresponds to her preferred candidate. Although, this type of electoral method provides the main focus in the rest of the thesis owing to its simplicity and also the fact that it is overwhelmingly the most commonly used, the preferential voting procedure is also given some attention.

Preferential or ranked voting allows voters to rank the candidates in terms of their preferences. This method also varies according to tallying methods and the number of runs in an election. For instance, the single transferable vote (STV) is a preferential method, which permits "wasted" votes to contribute to the overall result. That is, when a candidate reaches the quota declared by the election authorities, he wins a seat, then the other votes for the winner are transferred to other candidates depending on the preferences on those ballots as it is a multiple winner electoral method. If no candidate exceeds the quota, the candidate with the least votes is eliminated, and the ballots he has received are transferred to other candidates. This process is repeated until a candidate wins a seat or the number of seats left is equal to the number of the remaining candidates. In the case of a single winner election, the STV becomes the alternative vote (AV) or instant-runoff voting (IRV). The reader is referred to [RRE+05] for further details about electoral methods.

2.3 Communicating Sequential Processes

CSP is a formal language designed to describe concurrent systems in terms of components that interact by means of message passing. It belongs to the process calculus family and was introduced by Hoare in 1978 [Hoa78]. Since then it has been improved in terms of its ability to model concurrent systems as well as analysing security protocols, in particular, regarding its effectiveness when the model checking tool FDR [Low96, Ros97, RSG+00, Ros10] is employed. More-

over, CSP allows for systems to be modelled in terms of *processes*, which can synchronise and interact with the environment. In addition, it provides several semantic models to analyse the behaviour of processes and systems.

2.3.1 Syntax

Processes are defined in terms of a collection of *events* that they may perform and for CSP the occurrence of an event should be regarded as an atomic action without time. More specifically, a synchronised event can happen when all processes agree on executing it, i.e., it happens when it is inevitable. The set of events that are visible is called Σ, and the internal events are written as τ. Processes are associated with an interface or *alphabet*, denoted αP. If no alphabet is explicitly defined then it will be the set of events that the process can perform and the simplest process is $STOP$, which fundamentally means doing nothing. $SKIP$ is another named process, which terminates immediately. However, it is not a deadlock as in $STOP$, but a successful termination. In addition, $RUN(A)$ is the process that can always perform any member from the given set of events $A \subseteq \Sigma$ and the process, RUN is defined as $RUN(A) \mathrel{\widehat{=}} \Box_{x \in A}\, x \rightarrow RUN(A)$.

The CSP grammar is used for the processes, P, and Q, the set of events, A, variable, x, channel, c, events, a and b, and data, v with data-type, T. The elements of language used in this work are set out in Table 2.5 (See [Ros10, Sch99] for a fuller account of the language).

Given a process P and an event a in Σ, the prefix process $a \rightarrow P$ is initially willing to perform an event a. That is, it waits until the event, a, is performed then behaves like the process P. For instance, the process, $P_1 \mathrel{\widehat{=}} a \rightarrow b \rightarrow STOP$ will perform the events a and b, then it will terminate.

Events can also be structured into any number of parts. For example, an event of the form $c.v$ can represent a channel c passing value v. The set of values T that can pass along c is the *type* of c, so the set of events associated with channel c of type T is $\{c.v \mid v \in T\}$. This can also be written as $\{|c|\}$. If C is a set of channels, then $\{|C|\} = \bigcup_{c \in C} \{|c|\}$.

The input process $c?x \rightarrow P(x)$ is initially prepared to accept a value that will be bound to the locally introduced variable x along channel c, and then behave as P having received input x. The output process $c!v \rightarrow P$ outputs value v along channel c. Throughout the thesis, structured events are used to describe events in voting systems. For example $vote.v.c$ can represent voter v casting a vote for candidate c.

Recursive processes in CSP, are described by means of recursive definitions of the form $N \mathrel{\widehat{=}} P$, where N is a process name that can appear in process P. N can also

$P, Q ::=$	processes
$STOP$	stop (deadlock)
$SKIP$	successful termination
$a \rightarrow P$	prefixing
$c?v \rightarrow P(v)$	data input
$c!v \rightarrow P$	data output
$P \Box Q$	external choice
$\Box_{x \in A} P(x)$	indexed external choice
$P \sqcap Q$	nondeterministic choice
$\sqcap_{x \in A} P(x)$	indexed nondeterministic choice
if b then P else Q	conditional choice
$b \& P$	if b then P else $STOP$
$P \ {}_{\alpha P}\|_{\alpha Q}\ Q$	alphabetised parallel composition
$\Big\|_{x \in A}(P(x), \alpha P(x))$	indexed alphabetised parallel composition
$P \underset{A}{\parallel} Q$	generalised or interface parallel
$P \ \|\|\| \ Q$	interleaving
$\Big\|\|\Big\|_{x \in A} P(x)$	indexed interleaving
$P \setminus A$	hiding
$P[\![R]\!]$	relational renaming

Table 2.5: CSP notation

take parameters, giving definitions of the form $N(p) \mathrel{\widehat{=}} P(p)$. Thus, the process, $P_2 \mathrel{\widehat{=}} a \rightarrow b \rightarrow P_2$, are recursively defined, alternating between the events, a and b. Moreover, instead of defining a recursive process with one equation, *mutual* recursion can be used for this purpose. For instance, the process definitions $P_3 \mathrel{\widehat{=}} c?x \rightarrow P_4(x)$ and $P_4(x) \mathrel{\widehat{=}} d!x \rightarrow P_3$ describe a process that repeatedly inputs and then outputs a value.

2.3.2 Choice Operators

CSP offers choice operations for processes, which are called *external* and *nondeterministic* choice operators denoted as $\Box$ and $\sqcap$, respectively. The process $P \Box Q$ can act like P or Q depending on the choice of the initial event chosen by the environment. For instance, for the process $(a \rightarrow P) \Box (b \rightarrow Q)$, if the first event chosen is a then the process will behave as the process P, after performing the event a. Similarly, if the first event chosen is the event b, subsequently

the process will act as the process Q. While the external choice operator leaves the choice to its environment, in a nondeterministic process, the choice is made internally. Thus, the process $(a \rightarrow P) \sqcap (b \rightarrow Q)$ can act as either $a \rightarrow P$ or $b \rightarrow Q$ and the environment has no control over which this is. Indexed versions of external and nondeterministic choices allow for choices to be made among a number of processes. In addition to these, there is also the traditional conditional choice if-then-else.

2.3.3 Parallel Operators

Systems can be made up of a collection of processes that run in parallel and that are synchronised in relation to the events that they agree to perform. For instance, the alphabetised parallel $P \ {}_{\alpha P}\|_{\alpha Q} \ Q$ executes P and Q in parallel, where they have to synchronise regarding those events that are in both of their alphabets, but they can perform other events independently. Thus, they must only agree on the events in the intersection $\alpha P \cap \alpha Q$. $P \parallel Q$ will be used as shorthand for $P \ {}_{\alpha P}\|_{\alpha Q} \ Q$ in the remainder of this thesis. Additionally, the events that need to be synchronised can be identified with the generalised or interface parallel operator $P \underset{A}{\parallel} Q$, where all events in A must be synchronised, but not the events outside A, which can be performed independently. These parallel operators are associative and commutative, so any number of processes in parallel can be combined, in any order, without ambiguity. Thus, $P \parallel Q \parallel R$ can be written as representing the parallel combination of three processes P, Q and R.

Alternatively, the desire may be to run any two processes independently of each other, i.e., they do not synchronise with any events, not even those that they share and this behaviour can be implemented using the interleaving operator written as "|||", which also has an indexed form to describe the interleaving of a family of processes. Finally, all parallel operators, including interleaving ones are symmetric, associative and distributive over external and nondeterministic choice.

2.3.4 Abstraction Methods

The abstraction methods that are frequently used in the analysis are: the *hiding* abstraction method used as $P \setminus A$ to make occurrences of events in A internal, and hence invisible to an observer and the *renaming* method shown as $P[\![R]\!]$ for a relation R, so that the occurrences of an event a are replaced by events b such that aRb. An example for the hiding operator is that for a given set of events

$A \in \Sigma$ with the following step law:

$$(a \to P) \setminus A = \begin{cases} P \setminus A & \text{if} \quad a \in A, \\ a \to (P \setminus A) & \text{if} \quad a \notin A, \end{cases}$$

The latter method is used under the circumstance that an observer can see that an event is happening, but is unable to detect which event it is. More specifically, in renaming if R is a relation on the alphabet of process P, then $P[\![R]\!]$ behaves like P except that it performs different events. Moreover, whenever P can perform the event a, $P[\![R]\!]$ can perform each event from its relational image, $R[\{a\}]$. For instance, given the process $P \mathrel{\widehat{=}} a \to a \to STOP$, and the relations aRb and aRc, $P[\![R]\!]$ should be considered as:

$$P[\![R]\!] \mathrel{\widehat{=}} (b \to (b \to STOP \,\Box\, c \to STOP)) \,\Box\, (c \to (b \to STOP \,\Box\, c \to STOP))$$

Some earlier accounts of CSP [Sch99] used a function or its inverse in place of the relation R, to provide *alphabet renaming* and *inverse renaming*, but in this thesis the more general approach using the relation as described in [Ros10] is employed.

A substitution-like notation is often used for describing relations. Regarding this, $P[\![^a/_b]\!]$ means that the event or channel b is replaced by a in P, e.g., $(b \to STOP)[\![^a/_b]\!] \mathrel{\widehat{=}} a \to STOP$, and $(b?x \to STOP)[\![^a/_b]\!] \mathrel{\widehat{=}} a?x \to STOP$. More generally, multiple substitutions, including $P[\![^{a,\,b}/_{b,\,a}]\!]$ (a maps to b and b maps to a), many-to-one renaming, $P[\![^{a,\,a}/_{b,\,c}]\!]$ (b and c both map to a) and one-to-many renaming $P[\![^{b,\,c}/_{a,\,a}]\!]$ (a maps to both b and c) are allowed. In addition, overload notation is used, with $[\![^X/_Y]\!]$ referring to the relation corresponding to the renaming.

A useful result when composing renamings is that renaming via relation R followed by renaming through R' is equivalent to renaming through the relational composition $R\,;R'$.

Lemma 1. $P[\![R]\!][\![R']\!] \mathrel{\widehat{=}} P[\![R\,;R']\!]$

2.3.5 Traces and Other Semantic Models

CSP provides a wide range of semantic models, which help in the description of a process behaviour. With respect to these, *the traces model*, $\mathcal{T}$ is employed in this thesis, which refers to the finite sequences of events that a process can perform as denoted by $\langle a_1, a_2, \ldots, a_n \rangle$. The empty trace is denoted $\langle\rangle$, and the concatenation of two traces as $tr_1 \hat{} tr_2$. $tr \upharpoonright A$ is the projection of tr onto the set A (i.e., the sequence of events in tr that are in A), and $tr \setminus A$ is the projection of tr onto $\Sigma \backslash A$, i.e., the trace tr with events from A removed. Two traces tr_1 and tr_2 are related by R, if they are also pointwise related, i.e., they are the same length and the events at each position are related by R.

The set of all traces of the process P is written $traces$(P), which is a non-empty set as every process has the empty trace, $\langle\rangle$, in its trace set. For instance, the set of traces of the process, $a \rightarrow b \rightarrow STOP$, is $\{\langle\rangle, \langle a\rangle, \langle a, b\rangle\}$. Some of the definitions in terms of the traces model are as follows:

$$
\begin{array}{lcl}
traces(STOP) & = & \{\langle\rangle\} \\
traces(SKIP) & = & \{\langle\rangle, \langle\checkmark\rangle\} \\
traces(a \rightarrow P) & = & \{\langle\rangle\} \cup \{\langle a\rangle \hat{} s \mid s \in traces(P)\} \\
traces(P \,\Box\, Q) & = & traces(P) \cup traces(Q) \\
traces(P \sqcap Q) & = & traces(P) \cup traces(Q) \\
traces(P \setminus X) & = & \{s \setminus X \mid s \in traces(P)\} \\
traces(P \parallel Q) & = & \{s \in (\alpha P \cup \alpha Q)^* \mid s \upharpoonright \alpha P \in traces(P) \wedge \\
 & & \qquad\qquad s \upharpoonright \alpha Q \in traces(Q)\} \\
traces(P[\![R]\!]) & = & R[traces(P)]
\end{array}
$$

Additionally, CSP offers *the failures model*, $\mathcal{F}$, which provides more information about what a process may refuse to perform. A *failure* is a pair (tr, X), where $tr \in traces(P)$ and X is the refusal set of the process P after the trace tr. A *refusal* is a set of events that a process refuses to perform in a particular state. Thus, the trace/refusal pair helps to distinguish two processes that have the same traces, but differ in terms of what events they refuse to perform. For instance, although the traces model cannot distinguish $P \,\Box\, Q$ from $P \sqcap Q$, it is possible with the failures model. To see this, suppose that $P \mathrel{\widehat{=}} a \rightarrow STOP \,\Box\, b \rightarrow STOP$ and $Q \mathrel{\widehat{=}} a \rightarrow STOP \sqcap b \rightarrow STOP$, under these process definitions, both process traces refine each other as their set of traces are the same and equal to $\{\langle\rangle, \langle a\rangle, \langle b\rangle\}$. However, in terms of the failures model, the refusal sets of these processes differ as the refusal set for P is $\{(\langle\rangle, \emptyset), (\langle a\rangle, \{a, b\}), (\langle b\rangle, \{a, b\})\}$ and for Q is $\{(\langle\rangle, \{a\}), (\langle\rangle, \{b\}), (\langle a\rangle, \{a, b\}), (\langle b\rangle, \{a, b\})\}$. In more detail, the difference between the refusal sets of these processes is the initial choice between performing the event a or b. That is, with the external choice, P does not refuse to perform any of a and b as the decision is made by the environment, while Q can refuse to perform a or b. Further, the *Failures/divergence model*, $\mathcal{M}$, gives more information than the traces model regarding whether a process ever reaches a state where it can *diverge*, in other words, the process continues performing τ's forever and refuses all visible events. Generally, the failures model is used to check liveness properties, in terms of whether the process or system performs any good behaviour, whereas in the traces model the checking is for the bad behaviour being performed by the process. In our analysis, a number of compression functions have been used. More details about these operators can be found in Section 5.4 and in Section 7.7.

2.3.6 Traces Refinement and Model Checking

Traces refinement is offered in CSP to compare behaviour of processes. Regarding this, if every trace of Q is also a trace of P, then Q trace-refines P or P is refined by Q, denoted $P \sqsubseteq_{\mathrm{T}} Q$ and this is used in this thesis. If P and Q refine each other then they are trace equivalent as denoted $P \equiv_{\mathrm{T}} Q$. It is deemed that the traces model is sufficient for the purposes of this research aimed at verifying whether the model of a system satisfies a certain specification, such as anonymity.

Failures-Divergence Refinement (FDR) [GGH$^+$] is the model checking tool engaged with for the analysis, which was designed by Formal Systems (Europe) Ltd to check formal models created with the CSP formal language. This allows for the automated checking of assertions of specifications *SPEC* and implementation (*MODEL*). That is, *MODEL* meets the specification *SPEC* if *MODEL* is a refinement of *SPEC*. FDR checks the assertions automatically, and if the refinement does not hold, then it produces the first 100 counter-examples of the refinement, which are sequences of events that demonstrate the violation of the specification. Although FDR is easily used to check refinements, it suffers from a generic problem that all model checking tools suffer from: state space explosion. With regards to this problem and for further details in CSP and FDR, the reader is referred to [Ros97, Sch99, GGH$^+$, Ros10].

Apart from FDR, there exist other model checking tools for CSP, such as; ProB [LB03] and Process Analysis Toolset (PAT) [SLD09]. ProB is a tool set that can be used as a model checker for CSP specifications besides other formal languages. It supports analysis of CSP processes using both refinement checking and linear temporal logic (LTL) model checking. The other model checking tool for CSP is PAT. It can perform refinement checking, LTL model checking and simulation of CSP processes. Because FDR is a mature model checking tool for CSP specifications with a number of semantics available, and our departmental experience with CSP and FDR model checker, they have been employed in all our analysis throughout this thesis.

In summary, in this thesis CSP is used as the formal language to define the anonymity requirement of voting system, and to model a number of voting systems as parallel compositions of individual processes. Additionally, FDR is employed to assist in verifying whether the models of the voting systems meet their requirements (specifications).

2.4 Summary

In this chapter, the essential background to the cryptographic primitives, which are drawn upon to carry out the investigation into voting systems in Chapter 6

and 7 have been described. Subsequently, an overview of the key voting systems, their requirements and electoral methods was given. Additionally, the formal language CSP was introduced with its notation and semantics, as well as the model checking procedure and the tool FDR. In the next chapter abstractions (deduction rules) of the cryptographic primitives are employed when modelling and analysing security protocols using CSP and FDR. In particular, the use of the CSP approach, as the key methodology employed in this research, is explained and justified by drawing upon case studies provided in the existing literature.

Chapter 3

Modelling and Analysis of Security Protocols using CSP

From the previous chapters, the necessity for automated formal analysis of privacy-related requirements in voting systems has emerged with justification. This chapter forms the foundation of this researcher's approach to the modelling and analysis of voting systems using CSP and FDR by presenting two protocol methods found in the literature. Firstly in Section 3.2, by presenting a formal analysis of the dining cryptographers protocol [Cha88], the aim is to demonstrate how security protocols and their specifications may be modelled using CSP. The modelling and analysis of this protocol given in [RSG+00] is considered in a hostile environment from an observer's point of view and this approach will be used in the analysis of voting systems in Chapters 4 – 6. Additionally, the anonymity definition (strong anonymity) as the security protocol specification given in this chapter will be used in Chapter 4 in order to compare two different anonymity definitions for voting systems. Following this, Lowe's notorious attack [Low96] on the Needham-Schroeder Public-Key (NSPK) protocol [NS78] shall be presented in Section 3.3. This critical analysis of the NSPK will lead to the provision of an active Dolev-Yao-like intruder model, namely lazy spy, which will then be tested for its efficiency with regards to voting systems using CSP. Moreover, the analysis of the secrecy property of the NSPK will help in defining that for voting systems in Chapter 7.

3.1 Overview

A security protocol is designed to provide a certain goal comprising various security-related properties, such as: secrecy, integrity, anonymity and so forth [RSG+00]. In order to claim that a protocol is secure, it should be able

to meet all of its security-related objectives in a hostile environment where an intruder is trying to break them. Ryan *et al.* [RSG+00] point out some of the difficulties encountered when designing and analysing security protocols because: the security properties can be too subtle to define precisely; describing the model in a hostile environment can be too complex; defining any intruder's ability is extremely challenging and the concurrency involved further complicates any analytical endeavours. Although the protocols themselves are easy to express, for instance, the NSPK can be defined in three lines, they are too complex to evaluate using hand-proofs. Nevertheless, such protocols can be easily analysed by model checking tools, such as FDR, as long as the security objective is defined precisely, and the communication network between agents and the hostile environment are modelled efficiently. However, because model checking tools are bounded by the number of participants due to the state space explosion problem, as explained in Chapter 2, they cannot achieve the security objectives of the protocols for infinitely many agents, but they can effectively find the attacks breaking the security objectives involving a small number of agents.

The following two protocol analyses in the rest of this chapter based on [RSG+00] and [Ros97] illustrate how to overcome the aforementioned difficulties. First of all, the focus is on a well-known security protocol, the dining cryptographers, and involves explanation on how a number of anonymity notions can be captured by modelling and analysis of this toy example under a passive intruder model. Such an intruder model is easy to define as the intruder can only observe the public channels. Following this, analysis of the NSPK protocol under an active intruder model and the precise formalisations of its security objective, secrecy, are given. This is of relevance because in this researcher's opinion the characteristics of these security protocols and using the CSP approach in their analysis are appropriate for the analysis of voting systems. Finally, throughout the thesis the analysis is conducted with the FDR model checker.

3.2 Dining Cryptographers

The dining cryptographers protocol was proposed by David Chaum [Cha88] and illustrated that it was possible to send and receive anonymous messages unconditionally or cryptographically. Fundamentally, it involves a scenario where three cryptographers are having dinner together in a restaurant, and each of them is informed by their organisation whether or not he or she is paying the bill. However, the organisation cannot only choose one of cryptographers to pay the bill, but can also opt to pay it itself and the aim is to allow them to find out whether one of them is paying or not. Moreover, if one of the cryptographers is paying, for some reason, they would like to keep his/her identity anonymous. How can the participants ensure this is achieved?

The protocol works as follows. Each cryptographer tosses a coin, which is also made visible to their right-hand neighbour and thus, each is able to see two coin tosses. Once they examine the results of these tosses, they can either say *agree* if they are the same or *disagree* if they are different. If a cryptographer is not paying then he/she will always tell the truth about the comparison of tossing results. However, the paying cryptographer will say the opposite. If the number of disagrees are even then the organisation is paying, whereas if odd, then one of the cryptographers is, but is able to remain anonymous.

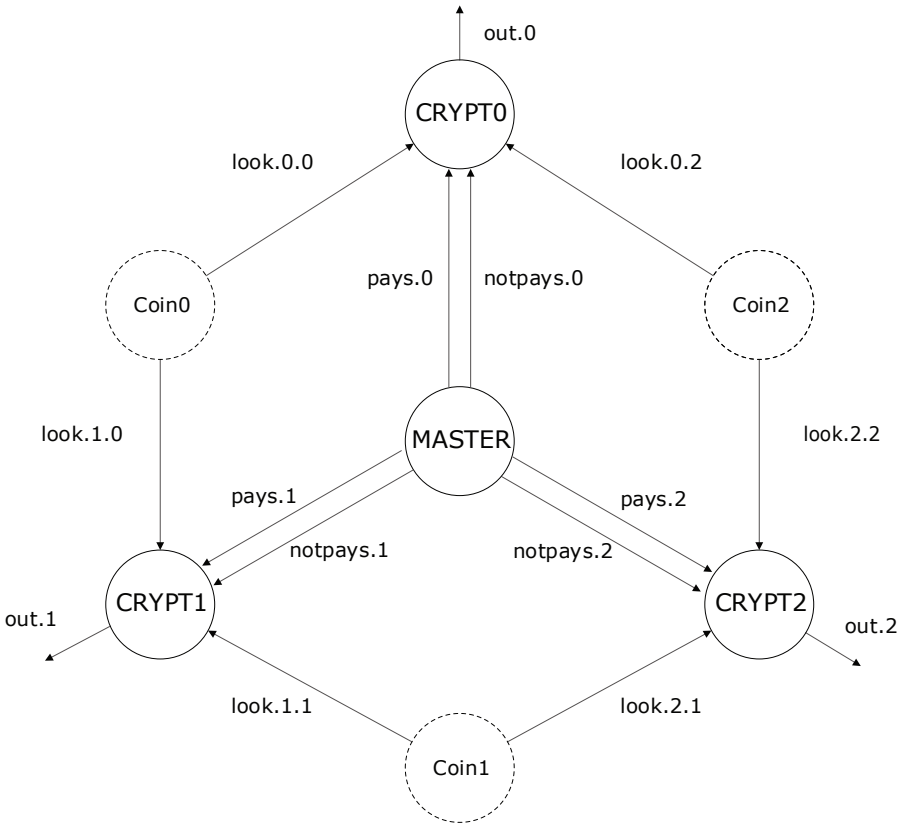

Figure 3.1: Dining cryptographers protocol (adopted from [RSG+00])

As illustrated in Figure 3.1, the protocol can be modelled as a parallel composition of the cryptographers, coins, and master processes, with the lattermost determining who is paying the bill (note that in the following CSP descriptions of these processes are based on [SS96]). The events *pays.i* and *notpays.i* between the cryptographers and the master are the instructions regarding the payment, and the events *look.i.j.x* model the cryptographer i reading the value x from the coin j. Lastly, the events of the form *out.i* model the declarations made by the cryptographers.

In more detail, the process *MASTER* chooses either to pay himself, or one of the cryptographers to pay non-deterministically and hence, the process definition for the master is defined as follows, where $\mathcal{C}$ is the set of cryptographer identities:

$$\begin{aligned} MASTER \mathrel{\widehat{=}} & notpays.0 \rightarrow notpays.1 \rightarrow notpays.2 \rightarrow STOP \\ & \sqcap \\ & \sqcap_{i \in \mathcal{C}} \left(\begin{array}{l} pays.i \rightarrow \\ notpays.((i+1) \bmod 3) \rightarrow \\ notpays.((i+2) \bmod 3) \rightarrow STOP \end{array} \right) \end{aligned}$$

Consequently, the process $CRYPT()$ models the cryptographers. The cryptographer i is first given instructions regarding payment with the events $pays.i$ and $notpays.i$ by the *MASTER*, then they compare two coins that they can see by the channels $look.i$. If the coins are the same then the cryptographer performs an $out.i.agree$ event, or otherwise an $out.i.disagree$ event. If the cryptographer pays and the coins are the same then the cryptographer performs an $out.i.disagree$ event, else an $out.i.agree$ event and hence, the process that models this behaviour can be defined as:

$$\begin{aligned} CRYPT(i) \mathrel{\widehat{=}} & \left(\begin{array}{l} notpays.i \rightarrow look.i.i?x_1 \rightarrow \\ look.i.((i+1) \bmod 3)?x_2 \rightarrow STOP \\ \left(\begin{array}{l} \text{if } (x_1 = x_2) \text{ then } out.i.agree \rightarrow STOP \\ \quad \text{else } out.i.disagree \rightarrow STOP \end{array} \right) \end{array} \right) \\ & \Box \\ & \left(\begin{array}{l} pays.i \rightarrow look.i.i?x_1 \rightarrow \\ look.i.((i+1) \bmod 3)?x_2 \rightarrow STOP \\ \left(\begin{array}{l} \text{if } (x_1 = x_2) \text{ then } out.i.disagree \rightarrow STOP \\ \quad \text{else } out.i.agree \rightarrow STOP \end{array} \right) \end{array} \right) \end{aligned}$$

Finally, the process that models each coin is defined in terms of a choice between reading heads and reading tails.

$$\begin{aligned} COIN(i) &\mathrel{\widehat{=}} Heads(i) \sqcap Tails(i) \\ Heads(i) &\mathrel{\widehat{=}} look.i.i.heads \rightarrow Heads(i) \\ &\quad \Box\, look.((i-1) \bmod 3).i.heads \rightarrow Heads(i) \\ Tails(i) &\mathrel{\widehat{=}} look.i.i.tails \rightarrow Tails(i) \\ &\quad \Box\, look.((i-1) \bmod 3).i.tails \rightarrow Tails(i) \end{aligned}$$

In order to construct the model, the cryptographers are interleaved as they act independently each other $CRYPTS \mathrel{\widehat{=}} |||_c CRYPT(c)$. For the same reason, the coins are interleaved among themselves as $COINS \mathrel{\widehat{=}} |||_i COIN(i)$. However, the cryptographers and coins should synchronise on common events *look*, and this

composition should be run in parallel with the master process synchronising on *pays* and *notpays* events when modelling the overall protocol, say *MEAL*.

$$MEAL \mathrel{\widehat{=}} ((CRYPTS \underset{\{|\, look \,|\}}{\parallel} COINS) \underset{\{|\, pays \,|\} \cup \{|\, notpays \,|\}}{\parallel} MASTER)$$

The dining cryptographers model in CSP has now been modelled, however; the *anonymity* property, which is to be checked against this protocol, needs to be specified before the analysis can proceed and therefore in the next subsection this issue is addressed.

3.2.1 Formal Specification of Anonymity

Schneider and Sidiropoulos [SS96] state that anonymity is a property of agents rather than the messages carried on the channels, for the latter is concerned with confidentiality. Moreover, these authors give an anonymity definition for security protocols, which is termed here *strong* anonymity (this notion of anonymity will be further discussed in Chapter 4). The definition is expressed by these authors informally as "a message that could have been originated from one agent could equally have been originated from any other" [SS96, p.5]. That is, if the message x originated by the user i is considered in the form $i.x$, then it could equally have been in the form $j.x$, where j is a user from the set of all users.

In the CSP approach to anonymity, agents' actions in relation to events are of the form $channel.i.x$, where the *channel* represents the type of event, i is the identity of the agent and x is the content of the event. As anonymity is concerned with the origin of such events, it refers to cases where an event $channel.i.x$ cannot, in some sense, be distinguished from $channel.j.x$, where i and j are two agents within the group of *USERS*, and x is in the set *Data*. Hence, the set of all the messages communicated on the channels covering all identities of the agents that need to be hidden can be written as:

$$A = \{|\, channel.i.x \mid i \in USERS, x \in Data \,|\}$$

The intuition is that if an observer has access to only the content x of the message, and the identity of the agent i is hidden from them, then the content could equally have been generated by any of the agents.

Definition 1 (Strong Anonymity [SS96])**.** *A process P is* strongly anonymous *on the alphabet $A \subset \Sigma$ if:*

$$P[\![^{x}/_{y} \mid x, y \in A]\!] \equiv_{\mathrm{T}} P$$

The original definition in [SS96] expressed this definition using functional and inverse functional renaming, as follows (now cast in relational notation):

$P[\![^{\beta}/_{A}]\!][\![^{A}/_{\beta}]\!] \equiv_{\mathrm{T}} P$ where $\beta \notin \alpha P$

$[\![^{\beta}/_{A}]\!]$ is used as shorthand for $[\![^{\beta}/_{x} \mid x \in A]\!]$, and $[\![^{A}/_{\beta}]\!]$ as shorthand for $[\![^{x}/_{\beta} \mid x \in A]\!]$. Moreover, $[\![^{A}/_{A}]\!]$ is used as shorthand for $[\![^{x}/_{y} \mid x, y \in A]\!]$. The definition given here is equivalent, since the composition of the relations $[\![^{\beta}/_{A}]\!]$ and $[\![^{A}/_{\beta}]\!]$ is indeed $[\![^{A}/_{A}]\!]$. $P[\![^{A}/_{A}]\!] \equiv_{\mathrm{T}} P$ means that the two processes, P, and the renamed process, $P[\![^{A}/_{A}]\!]$, are trace equivalent and so are indistinguishable from the point of view of an observer who can see the traces of each of these processes. The two corollaries are:

1. If the abstracted system P is anonymous on the sets A and A', then P is anonymous on $A \cup A'$, if $A \cap A' \neq \emptyset$.

2. If P is anonymous on the set A and $A' \subseteq A$, then P is anonymous on the set A'.

3.2.2 Analysis

Anonymity is often considered in presence of an observer. In the dining cryptographers example, the observer could be an outsider sitting at another table or other cryptographers sitting at the same one and in the anonymity analysis for this protocol only these two cases are considered (for more case analysis of this protocol, the reader is referred to [SS96]).

Anonymity from an Outsider's Point of View

The anonymity of the system *MEAL* from an observer's point of view is provided where he/she can only see the *out* events. Thus, the internal events like *look* will be abstracted away, but not *pays* events, because they are needed for checking anonymity between the cryptographers on the set $A_1 = \{\mid pays \mid\}$. Regarding this, according to the *strong* anonymity definition, it is necessary to check whether the trace equivalence

$MEAL[\![^{A_1}/_{A_1}]\!] \equiv_{\mathrm{T}} MEAL$

holds. However, prior to this the *look* events need to be hidden as the observer is not allowed to see a cryptographer looking at a coin yet and the best way to model this is by using the *hiding* operator. Therefore, the assertion required for the anonymity check is:

$$MEAL[\![^{A_1}/_{A_1}]\!] \setminus \{|\, look \,|\} \equiv_{\mathrm{T}} MEAL \setminus \{|\, look \,|\}$$

However, if the observer is allowed to see when a cryptographer is looking at coins, but not the content or outcome of the tosses then renaming is the most suitable abstraction method and under these circumstances the trace equivalence required to check for anonymity of the system is:

$$MEAL[\![^{look.i.j}/_{look.i.j.x}]\!][\![^{A_1}/_{A_1}]\!] \equiv_{\mathrm{T}} MEAL[\![^{look.i.j}/_{look.i.j.x}]\!]$$

FDR confirms that anonymity is provided by the dining cryptographers protocol model in both cases.

Anonymity Against Other Cryptographers

Anonymity against the other cryptographers is also provided by the protocol, because each is not able to distinguish the payer from other two. In more detail, the observer, say cryptographer i, now possesses more information than the outsider in the previous subsection, because he/she can see the $look.i$ events. As a result, the anonymity of the protocol needs to be checked for the set A_2 below.

$$A_2 = \{|\, pays.((i+1) \bmod 3), pays.((i+2) \bmod 3) \,|\}.$$

Following that, the set of sensitive information (the $look$ events of the other cryptographers):

$$S = \{|\, look.((i+1) \bmod 3), look.((i+2) \bmod 3) \,|\}$$

needs to be hidden from the cryptographer i and therefore, the assertion required for anonymity in this case is the following.

$$MEAL[\![^{A_2}/_{A_2}]\!] \setminus S \equiv_{\mathrm{T}} MEAL \setminus S$$

However, in the case where cryptographer i is able to see other cryptographers looking at coins but not the contents, the renaming operator is more appropriate, as used previously. As a consequence, the following equivalence needs to be checked for the anonymity of the protocol.

$$MEAL_1[\![^{A_2}/_{A_2}]\!] \equiv_{\mathrm{T}} MEAL_1$$

where $MEAL_1 \mathrel{\widehat{=}} MEAL[\![^{look.a.b}/_{look.a.b.x \mid a \in \mathcal{C} \setminus i}]\!]$.

FDR verifies that the dining cryptographers protocol satisfies the anonymity requirements against the other cryptographers.

3.2.3 Summary

In this subsection it has been demonstrated how FDR and CSP can be used to model a simple security protocol and to verify mechanically whether the protocol satisfies the anonymity requirement. Moreover, the formal specification of *strong anonymity* was presented as such a requirement and this on first consideration might relevant to the analysis of voting systems (the differences are explored in Chapter 4). The analysis has illustrated that the operators under CSP, such as *hiding* and *renaming*, are useful for abstracting away sensitive information in the presence of an observer or passive intruder. This kind of intruder model is used in Chapters 4 - 6, when analysing the ThreeBallot and Prêt à Voter voting systems with a different specification for anonymity.

In the next section, an active and stronger intruder model is described, which is then used for analysing the effectiveness of the NSPK protocol in terms of its secrecy property. Subsequently, this intruder model is used in Chapter 7 during the evaluation of the vVote voting system.

3.3 Needham Schroeder Public-Key Protocol Analysis

The Needham Schroeder Public-Key (NSPK) protocol aims to provide mutual authentication after a sequence of message exchanges between a number of agents. More specifically, mutual authentication between the agents Alice and Bob ensures that if Alice thinks she has been communicating with Bob, then she should indeed be communicating with him and the same applies for Bob as well, i.e., he should be sure that who he thinks he is talking to should indeed be Alice. Additionally, the protocol aims to provide a secrecy property, which means the contents of the communication between Alice and Bob should be kept solely to them. That is, under the NSPK protocol, the secret is the nonce (an abbreviation for "number used once"), which takes the form of a shared symmetric key, such that no party other than the key holders are able to decipher messages encrypted by using it.

In order to achieve its goals, the protocol uses public-key cryptography, whereby each agent a and b, and the server s possess their own public and secret keys, namely, (pk_a, sk_a), (pk_b, sk_b) and (pk_s, sk_s), respectively. Additionally, the protocol uses nonces, i.e., used only once in a protocol run, denoted as n_a and n_b generated by the agents a and b, respectively. Moreover, the protocol agents can be either an initiator to start a session or a responder for a request to establish one. The following describes how the NSPK protocol works in steps.

$m.1 \quad a \rightarrow s : a, b$
$m.2 \quad s \rightarrow a : \mathsf{S}_{sk_s}(pk_b, b)$
$m.3 \quad a \rightarrow b : \mathsf{E}_{pk_b}(n_a, a)$
$m.4 \quad b \rightarrow s : b, a$
$m.5 \quad s \rightarrow b : \mathsf{S}_{sk_s}(pk_a, a)$
$m.6 \quad b \rightarrow a : \mathsf{E}_{pk_a}(n_a, n_b)$
$m.7 \quad a \rightarrow b : \mathsf{E}_{pk_b}(n_b)$

In the protocol, it is assumed that agents do not necessarily know each other's public keys and hence, the steps where the agents communicate with the server ($m.1$, $m.2$, $m.4$, $m.5$) in order to obtain each others' public key can be omitted. This is because the messages in $m.1$ and $m.4$ are in the clear, meaning the intruder can see what is being sent and thus it is not important that the intruder knows when the agent a is willing to communicate with b. Consequently, it can be assumed that the agents already know each other's public-keys, thus shrinking the protocol steps down to three ($m.3$, $m.6$, $m.7$). In this three-message version of the protocol, agent a sends his randomly chosen nonce n_a and his identity by encrypting with agent b's public key pk_b. Therefore, as the only agent knowing the secret key sk_b is b, if a receives n_a back from b, then a should be sure that she is talking to b, because only b can decrypt and evaluate the ciphertext encrypted under pk_b. Subsequently, in step $m.6$, b sends not only the nonce n_a derived from the ciphertext, but also a freshly generated random nonce, n_b to a and so he can authenticate the agent a by challenging her with the encryption $\mathsf{E}_{pk_a}(n_a, n_b)$. In the final message, a encrypts n_b with b's public key and sends it to b. Hence, as only agent a can decrypt the ciphertext, b is now sure that he is indeed talking to a. Now that a and b share a secret, which should be only known by them, they can use this key for further communication.

In order to analyse such a protocol as the NSPK in CSP, the behaviour of honest agents, *Alice Bob* and the server Tom as well as the misbehaving agent, the intruder, have to be modelled. Moreover, the intruder will have the ability to act with other good agents with a fake identity, in the name of say $Cameron$, thus functioning as a Dolev-Yao intruder [DY83]. The well-behaved agents will follow the rules of the protocol properly, whereas the intruder will not. The modelling and analysis of the NSPK in the following subsections will be based on [Ros97] and [RSG+00].

3.3.1 Defining Reliable Agents

An agent can either initiate the protocol to establish a secure communication with another agent or respond to a session request. The following process $Agents(a, n_a)$ describes the behaviour of the agent a with the nonce n_a. That

is, if the agent possesses a nonce then he chooses to be either an initiator or a responder, but if no fresh nonce exists, then the process terminates.

$$Agent(a, \langle n_a \rangle) \mathrel{\widehat{=}} \\ \quad \text{if } \langle n_a \rangle = \langle\rangle \text{ then } STOP \text{ else } Initiator(a, n_a) \,\Box\, Responder(a, n_a)$$

An initiator aims to be in a secure authenticated session with a target agent. Thus, initiator, a, chooses a target from the set of agents, $\mathcal{A}$, except herself and the server, T, with which to establish a session. Afterwards, she sends her identity a and the target's identity b to the server and following this, the initiator should accept the public-key certificate of b sent from the trusted server. Having received the public-key of the target b, the initiator challenges him with an encryption of her freshly generated nonce n_a. The initiator now should be ready to accept the correct message sent by the responder b along with a new nonce challenge n_b chosen from the set of nonces, $\mathcal{N}$. Finally, having answered the challenge with the message including n_b, the initiator enters a session with the responder. Note that the channel *comm*, on which honest agents communicate, has the form *agents.agents.messages*, which specifies the origin of the message as well as its destination.

$$Initiator(a, n_a) \mathrel{\widehat{=}} \\ \prod_{b \in \mathcal{A} \setminus \{a,T\}} \left(\begin{array}{l} comm.a.T.\langle a, b \rangle \rightarrow \\ comm.T.a.\mathsf{S}_{sk_T}(pk_b, b) \rightarrow \\ comm.a.b.\mathsf{E}_{pk_b}(n_a, a) \rightarrow \\ \quad \Box_{n_b \in \mathcal{N}} \left(\begin{array}{l} comm.b.a.\mathsf{E}_{pk_a}(n_a, n_b) \rightarrow \\ comm.a.b.\mathsf{E}_{pk_b}(n_b) \rightarrow \\ Session(a, b, n_b) \end{array} \right) \end{array} \right)$$

A responder has to accept any message initiating the protocol by other agents except the server, but it must be in an agreed form. For instance, if he is waiting a nonce, then the message should be a nonce, or if he is willing to accept a signed message by the server, then the message should indeed need to be signed by the server (any other messages that are not in an acceptable form are rejected). The responder's behaviour when he communicates with the initiator is the same, except obviously they have opposite perspectives. Moreover, the responder communicates with the server to obtain the other agent's public-key just like the initiator and finally enters a state in which he is in a session with the initiator.

$$Responder(b, n_b) \mathrel{\widehat{=}} \Box_{\substack{a \in \mathcal{A} \setminus \{b,T\} \\ n_a \in \mathcal{N}}} \left(\begin{array}{l} comm.a.b.\mathsf{E}_{pk_b}(n_a, a) \rightarrow \\ comm.b.T.\langle b, a \rangle \rightarrow \\ comm.T.b.\mathsf{S}_{sk_T}(pk_a, a) \rightarrow \\ comm.b.a.\mathsf{E}_{pk_a}(n_a, n_b) \rightarrow \\ comm.a.b.\mathsf{E}_{pk_b}(n_b) \rightarrow \\ Session(b, a, n_b) \end{array} \right)$$

Once the NSPK protocol is completed, the initiator and responder are in an authenticated session, and they can send secret messages to each other using the second nonce as the symmetric encryption key, with these messages being symbolically defined as ($AtoB$ and $BtoA$).

$$\begin{array}{l} Session(a, b, n) \mathrel{\widehat{=}} \\ \quad comm.a.b.\mathsf{E}_{pk_n}(AtoB) \rightarrow Session(a, b, n) \\ \quad \Box\ comm.b.a.\mathsf{E}_{pk_n}(BtoA) \rightarrow Session(a, b, n) \\ \quad \Box\ close \rightarrow Agent(a, \langle n \rangle) \end{array}$$

Similarly, the behaviour of the server, Tom, can be modelled in the protocol (note that the server is not needed in the three-message version of the NSPK), under the assumption that he is an honest party who both the initiator and the responder trust. He is responsible for issuing public-key certificates, and when asked for a particular agent's public key, he obligingly sends the signed public-key to the requesting agent.

$$Server(a, b, n) \mathrel{\widehat{=}} \Box_{\substack{a \in \mathcal{A} \setminus \{T\} \\ b \in \mathcal{A} \setminus \{a,T\}}} \left(\begin{array}{l} comm.a.T.\langle a, b \rangle \rightarrow \\ comm.T.a.\mathsf{S}_{sk_T}(pk_b, b) \rightarrow \end{array} \right)$$

Finally, in order to model the hostile environment, the intruder process is modelled in the next subsection.

3.3.2 Building the Intruder

The intruder can overhear the messages between the honest agents, being able to intercept, block and/or send fake messages as well as interacting with the latter. Moreover, he can also act as a legitimate agent and interact with the other honest agents using his trustworthy credentials, such as identity, public-key pair, a fresh nonce and a secret which he can send to the honest agents. However, he cannot decrypt a ciphertext unless he possesses the appropriate secret key. Additionally, he can extend his initial knowledge with the information learned by observing

communication channels either via overhearing or intercepting and subsequently generate any message by using the facts in his knowledge base. He is also given the power to deduce further knowledge from what he knows already under the deduction rules.

The knowledge that such an intruder can build is huge, as he can always *learn* new messages, and *say* those that he knows. However, this kind of modelling requires him be ready to act for any single fact he possesses, but it is not an efficient way to model such complex protocols, as the required state space is too large to be handled by the model checker tools. In order to address this, the lazy spy intruder model is a standard technique for security protocol analysis, in which the intruder is called *lazy* as it avoids the eagerness of pre-computation of unnecessary inferences [Ros10]. It is modelled in a way that the intruder should know only the facts that he does not initially know, or that cannot be deduced from his initial knowledge. Therefore, to this end, the intruder is modelled as parallel composition of those facts that can be learned (*learnable facts*).

To allow the intruder to build messages a number of deduction rules need to be defined. A deduction is a pair (X, f), where X is a finite set of facts and f is the fact that can be generated providing that the intruder possesses all the facts in X. Four of the deductions regarding the rules in Table 2.4 are as follows: $deductions_1$ captures the symmetric key encryption, $deductions_2$ refers to public-key encryption, $deductions_3$ covers digital signatures, and $deductions_4$ encapsulates sequences.

$$
\begin{aligned}
deductions_1(X) &= \{(\{f, k\}, \mathsf{E}_k(f)), (\{\mathsf{E}_k(f), k\}, f) \mid \mathsf{E}_k(f) \in X\} \\
deductions_2(X) &= \{(\{f, pk\}, \mathsf{E}_{pk}(f)), (\{\mathsf{E}_{pk}(f), dual(pk)\}, f) \mid \mathsf{E}_{pk}(f) \in X\} \\
deductions_3(X) &= \{(\{f, sk\}, \mathsf{S}_{sk}(f)), (\{\mathsf{S}_{sk}(f), dual(sk)\}, f) \mid \mathsf{S}_{sk}(f) \in X\} \\
deductions_4(X) &= \{(\{\langle m \rangle\}, nth(j, \langle m \rangle)), \\
&\qquad (\{nth(i, \langle m \rangle) \mid i \in \{0 \ldots \#\langle m \rangle - 1\}\}, \langle m \rangle) \\
&\qquad\qquad \mid \langle m \rangle \in X, j \in \{0 \ldots \#\langle m \rangle - 1\}\}
\end{aligned}
$$

Moreover, the finite set $deductions(X) = \bigcup_i deductions_i(X)$ is the set of all possible deductions covering all the deduction rules.

As mentioned above, the intruder only needs to fake messages that are valid, i.e., the messages that really travel on the protocol channels between agents, which makes it more efficient. Hence, in order for the intruder to understand the nature of a message, its type also needs to be defined. For instance, the message that is an encryption of a nonce can be described as being in the set $message_1$, and the justification of the message being heard can be defined with the set $comm_1$ as shown below.

$$
\begin{aligned}
message_1 &= \{\mathsf{E}_{pk_a}(n_a) \mid a \in \mathcal{A}, n_a \in \mathcal{N}\} \\
comm_1 &= \{a.b.m \mid m \in message_1, a \in \mathcal{A}, b \in \mathcal{A}, a \neq b\}
\end{aligned}
$$

$$
\begin{aligned}
message_2 &= \{\mathsf{E}_{pk_a}(n_a, n_b) \mid a \in \mathcal{A}, n_a \in \mathcal{N}, n_b \in \mathcal{N}\} \\
comm_2 &= \{a.b.m \mid m \in message_2, a \in \mathcal{A}, b \in \mathcal{A}, a \neq b\}
\end{aligned}
$$

$$
\begin{aligned}
message_3 &= \{\mathsf{E}_{pk_a}(n_b, b) \mid a \in \mathcal{A}, b \in \mathcal{A} \setminus \{a\}, n_b \in \mathcal{N}\} \\
comm_3 &= \{a.b.m \mid m \in message_3, a \in \mathcal{A}, b \in \mathcal{A}, a \neq b\}
\end{aligned}
$$

$$
\begin{aligned}
message_4 &= \{\mathsf{E}_k(f) \mid f \in Secret\} \\
comm_4 &= \{a.b.m \mid m \in message_4, a \in \mathcal{A}, b \in \mathcal{A}, a \neq b\}
\end{aligned}
$$

$$
\begin{aligned}
message_5 &= \{\mathsf{S}_{sk_T}((pk_a, a)) \mid a \in \mathcal{A}\} \\
comm_5 &= \{a.b.m \mid m \in message_5, a \in \mathcal{A}, b \in \mathcal{A}, a \neq b\}
\end{aligned}
$$

$$
\begin{aligned}
message_6 &= \{\langle a, b\rangle \mid a \in \mathcal{A}, b \in \mathcal{A} \setminus \{a\}\} \\
comm_6 &= \{a.b.m \mid m \in message_6, a \in \mathcal{A}, b \in \mathcal{A}, a \neq b\}
\end{aligned}
$$

The union of all such messages form the set of messages, $\mathcal{M} = \bigcup_i message_i$, and the set $comms = \bigcup_i comm_i$ is used in the intruder process in order for him to learn only the messages sent between the protocol participants. That is, these messages are the ones that he can learn or wants to construct for faking, but he can also construct messages in parts. In order to do so, the intruder needs a way to identify the facts that are relevant to the messages, $\mathcal{M}$ and this is made possible using the function called *explode*. This function can be applied to all the previously defined message types as follows:

$$
\begin{aligned}
explode(\mathsf{E}_{pk}(m)) &= \{\mathsf{E}_{pk}(m), pk, dual(pk)\} \cup explode(m) \\
explode(\mathsf{E}_k(m)) &= \{\mathsf{E}_{pk}(m), k\} \cup explode(m) \\
explode(\mathsf{S}_{sk}(m)) &= \{\mathsf{S}_{sk}(m), sk, dual(sk)\} \cup explode(m) \\
explode(\langle xs\rangle) &= \{\langle xs\rangle\} \cup \bigcup(\{explode(x) \mid x \in set(\langle xs\rangle)\}) \\
explode(x) &= x
\end{aligned}
$$

where $dual(pk) = sk$ and $dual(sk) = pk$. Thus, the function *explode* extracts the set of all facts that are of relevance to the given message. Similarly, all facts can be learned as:

$$AllFacts = \bigcup(\{explode(m) \mid m \in \mathcal{M}\})$$

and then by using the *deductions*() function all possible deductions being built.

$$AllDeductions = deductions(Allfacts)$$

In addition, the set of deductions and facts can be further cut down by omitting facts that cannot be reachable given the initial knowledge by the following procedure. *Known*, being the full initial knowledge closed with a function called $Close(\mathcal{IK})$, extracts all possible deductions by applying the deduction rules for the set, $\mathcal{IK}$, where the initial knowledge $\mathcal{IK}$ given to the intruder for the analysis of the NSPK is the set of all public-keys, the secret key of Cameron, a fresh nonce and a secret.

$$\begin{aligned} PossibleBasicKnowledge &= Known \cup messages \\ KnowableFacts &= Close(PossibleBasicKnowledge) \\ Learnablefacts &= KnowableFacts \setminus Known \end{aligned}$$

After discarding all the deductions that are already known to the intruder and the ones that cannot be reached by him, the deductions, $\mathcal{D}$, can be defined as follows:

$$\begin{aligned} \mathcal{D} = \{(X, f) \mid\ & (X, f) \in AllDeductions, \\ & f \in Learnablefacts, \\ & f \notin X, \\ & X \setminus Knowlablefacts = \emptyset\} \end{aligned}$$

The following process, *Ignorantof*, ensures that all facts in *Learnablefacts* have state: the process can always *learn* a fact, but it can only *say* once the fact is known. Moreover, the channel *infer* models the occurrence of inferences and an inference on the deduction (X, f) can happen when all its assumptions X are known, but f not.

$$\begin{aligned} Ignorantof(f) \mathrel{\widehat{=}} & f \in \mathcal{M} \,\&\, learn.f \rightarrow Knows(f) \\ & \Box\ infer?t \in \{(X, f') \mid (X, f') \in \mathcal{D}, f' = f\} \rightarrow Knows(f) \end{aligned}$$

$$\begin{aligned} Knows(f) \mathrel{\widehat{=}} & f \in \mathcal{M} \,\&\, say.f \rightarrow Knows(f) \\ & \Box\ f \in \mathcal{M} \,\&\, learn.f \rightarrow Knows(f) \\ & \Box\ infer?t \in \{(X, f') \mid (X, f') \in \mathcal{D}, f \in X\} \rightarrow Knows(f) \\ & \Box\ f \in Secret \,\&\, intruderknows.f \rightarrow Knows(f) \end{aligned}$$

The set *Secret* in the last line consists of the secret messages that the intruder should not be able to know and in this case, they are defined as the set $\{AtoB, BtoA\}$. Hence, when he gets to know an element of this set, the channel *intruderknows* flags this up signalling that the secret has been compromised. The following $AlphaL(f)$ defines the alphabet of each fact.

$$\begin{aligned} AlphaL(f) = \ & \{say.f, learn.f \mid f \in \mathcal{M}\} \\ & \cup \{intruderknows.f \mid f \in Secret\} \\ & \cup \{infer.(X, f') \mid (X, f') \in \mathcal{D}, f' = f\} \\ & \cup \{infer.(X, f') \mid (X, f') \in \mathcal{D}, f \in X\} \end{aligned}$$

Consequently, the intruder process can be defined as a parallel composition of facts, where the internal events *infer* are hidden.

$$\begin{aligned} & Intruder \mathrel{\widehat{=}} \\ & \quad \mathsf{chase}((\underset{f \in Learnablefacts}{\|} (Ignorant(f), AlphaL(f)))\backslash\{|infer|\}) \\ & \qquad ||| \ SayKnown \end{aligned}$$

The process *SayKnown* defined below ensures that the proper messages that are already known are learned and said, since the facts that are of relevance to the initial knowledge $\mathcal{IK}$ are not included in the *Learnablefacts* and *Deductions*.

$$\begin{aligned} SayKnown \mathrel{\widehat{=}} & \ say.f \in Known \cap \mathcal{M} \rightarrow SayKnown \\ & \Box \ learn.f \in Known \cap \mathcal{M} \rightarrow SayKnown \end{aligned}$$

chase is a special function that makes the analysis more efficient by avoiding checking all possible states [Ros10], because it always follows τs (unstable states) until no more τ is possible. Hence, some other possible ways are left to be explored and the operator has the right to choose any path to follow. In addition, although $\mathsf{chase}(P)$ changes the value of the process P sometimes, the behaviour of $\mathsf{chase}(P)$ is equivalent to P whenever P is deterministic and in this analysis of the intruder process this is always the case [Ros97]. In the following subsection, a description is provided on how to connect together the reliable agents and the intruder.

3.3.3 Composition of the System

Having modelled the reliable agents, including the server, as well as the intruder, the model that connects all these agents needs to be composed. That is, the reliable agents and the intruder need to be modified in a way that the messages are ready to be tampered with by the latter. To this end, the *comm* events, and *learn* and *say* ones need to be connected to each other (see Figure 3.2). Hence, for an agent a, the outgoing messages along the channel $comm.a.b.m$ and $learn.m$ events of intruder are renamed to either a $comm.a.b.m$ or $take.a.b.m$ events, and similarly inbound messages on the $comm.b.a.m$ and $say.m$ events are renamed to either a $comm.b.a.m$ or $fake.b.a.m$ events. The reliable agents will not notice the difference whether they communicate over the *comm* channels or the *take* and $fake$ channels and consequently, they have no way of knowing whether the

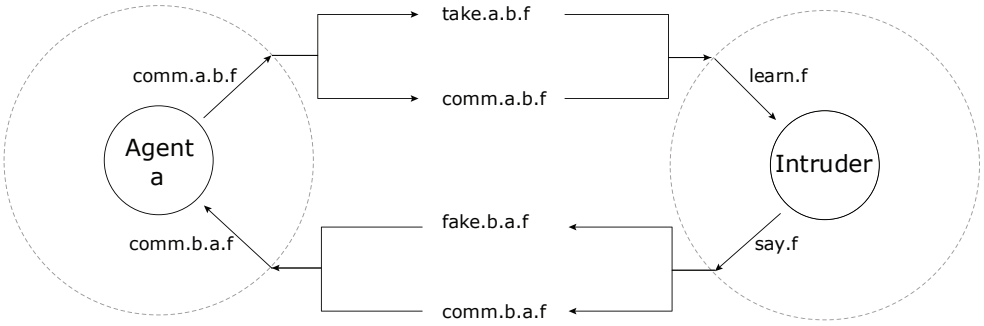

Figure 3.2: Connecting agents with renaming

intruder is communicating with them. The number of agents in such analysis needs to be kept to a minimum, because of the earlier discussed state explosion problem. Hence, in this model there exist two reliable honest agents *Alice* and *Bob* trying to establish an authenticated communication via the server, Tom, and an intruder who can act as $Cameron$ trying to break the protocol objectives. The roles for each agent can be predefined, such as, Alice being the initiator with a nonce, n_A, and Bob being the responder with the nonce n_B. The following processes model the renamed honest agents.

$$
\begin{aligned}
&Alice \mathrel{\widehat{=}} \\
&\quad Initiator(A, n_A) \\
&\qquad [\![take.A.p, fake.p.A, comm / comm.A.p, comm.p.A, comm \mid p \in \mathcal{A} \setminus \{A\}]\!]
\end{aligned}
$$

$$
\begin{aligned}
&Bob \mathrel{\widehat{=}} \\
&\quad Responder(B, n_B) \\
&\qquad [\![take.B.p, fake.p.B, comm / comm.B.p, comm.p.B, comm \mid p \in \mathcal{A} \setminus \{B\}]\!]
\end{aligned}
$$

$$
\begin{aligned}
&Tom \mathrel{\widehat{=}} \\
&\quad Server \\
&\qquad [\![take.T.p, fake.p.T, comm / comm.T.p, comm.p.T, comm \mid p \in \mathcal{A} \setminus \{T\}]\!]
\end{aligned}
$$

As a result, the $Network$ process below models the connected reliable agents in such a way as to allow the intruder to perform manipulations on the *comm* channels.

$$
Network \mathrel{\widehat{=}} (Alice \underset{\{|\, comm.A.B, \\ comm.B.A \,|\}}{\parallel} Bob) \underset{\{|\, comm.T.p, \\ comm.p.T \,|\, p \in \{A,B\} \,|\}}{\parallel} Tom
$$

Similarly, the intruder is renamed so that what he says can be said or learned again. Moreover, he learns only the valid messages due to the second renaming whereby all of them take the form $p, p'.f$, which indicates that they are from real communications between the agents. Finally, what the intruder *says* becomes $fake$.

$$
\begin{array}{l}
rIntruder \mathrel{\widehat{=}} \\
\quad Intruder \\
\quad\quad [\![say, learn / say, say]\!] \\
\quad\quad [\![comm.p.p'.f, take.p.p'.f / learn.f, learn.f \mid \ p.p'.f \in comms \ ]\!] \\
\quad\quad [\![fake.p.p'.f / say.f \mid p.p'.f \in comms,\, p \neq p']\!]
\end{array}
$$

The NSPK protocol, the process $System$, is then defined in terms of the parallel composition of $Network$ and $rIntruder$ that synchronise on $comm$, $take$ and the $fake$ events.

$$System \mathrel{\widehat{=}} Network \underset{\{\mid comm, take, fake \mid\}}{\parallel} rIntruder$$

Having modelled the protocol, the secrecy specification for the analysis is defined in the next subsection.

3.3.4 Formal Specification of Secrecy

Secrecy can be defined in a number of ways. For example, non-interference can be used in situations where the intruder is interested in agents' activity without any observation on the communications among legitimate agents. In other words, the view at the low level (not highly classified inputs and outputs) is unable to tell whether or not high level activity with highly sensitive data has occurred [RSG+00]. This sort of formalism of secrecy is too strict for the purpose of our analysis in this thesis where the intruder needs to derive content of any message transmitted over the network. Moreover, non-interference style secrecy definition is known to have issues with the secrecy of an encrypted channel as the high-level plaintext influences the ciphertext visible to the low-level user [RSG+00]. On the other hand, secrecy can be defined in a simpler and more appropriate way as a safety property. Such characterization (also called as reachability) considers the situation where the intruder reaches a certain state of knowing a secret as a secrecy breach. The intruder with such formalization can perform traffic analysis and deduce the facts about the protocol messages. In this thesis, the formal definition of secrecy as a reachability property will be focused.

As stated previously, the secrecy property can be defined as safety property, i.e., nothing bad should happen and hence, the protocol should not leak any secret

information during a protocol run. In the NSPK protocol, the set $Secret$ is defined as $\{AtoB, BtoA\}$. Therefore, in order for the protocol to provide secrecy for any messages, the intruder should not be appraised of these secrets by the end of the protocol run (the secret is kept secret) and to achieve this there is an extra channel called $intruderknows$. Although, this channel is not a part of the actual protocol, it is useful for security analysis, since its occurrence means that the intruder process is in a state where it knows the secret data, thus breaking the secrecy of the sensitive information. For instance, if the secret $AtoB$ has been possessed by the intruder then Alice must be thinking that she is in an authenticated communication with Bob, when she is actually interacting with the intruder.

As explained in Section 2.3, safety properties are generally modelled using the traces model in CSP and so is the secrecy property. The following trace refinement checks whether the event $intruderknows$ ever happens in a protocol run:

$$STOP \sqsubseteq_t System \setminus \Sigma \setminus \{| intruderknows |\}$$

where Σ is the alphabet of the process $System$:

$$\Sigma = \{| comm, take, fake, intruderknows |\}$$

Model checking of this protocol with FDR provides a trace in which the intruder performs an $intruderknows$ event, meaning the protocol fails to keep the secrecy of the messages $AtoB$ or $BtoA$. The following counter-example in Figure 3.3 is a confirmation of Lowe's attack.

The attack is mounted with two parallel running protocol sessions, α and β. In the protocol run α, the honest agent Alice wants to communicate with the corrupt agent Cameron, whereas during session β, the intruder initiates the protocol for a secure session with the other honest agent Bob, and makes him believe that he is interacting with Alice by learning all the secrets from the session α and by impersonating her.

In more detail, in step $\beta.6$, Bob responds the intruder's nonce challenge by sending the encrypted n_A and n_B under pk_A. However, the intruder, at this stage, cannot decrypt the message and retrieve the nonce n_B and instead he sends it back to Alice using her session with Cameron in step $\alpha.6$. At the end of the α session in step $\alpha.7$, Alice responds to Cameron's challenge by decrypting the message $\mathsf{E}_{pk_A}(n_A, n_B)$ and sending n_B to Cameron by encrypting it under his public key, $\mathsf{E}_{pk_C}(n_B)$. Because the intruder possesses Cameron's secret key, he can then obtain the nonce n_B and answer Bob's challenge by encrypting n_B under Bob's public-key. Having completed the session β, Bob now thinks that he is in an

$\alpha.1$	A	$\rightarrow$	T	$: \langle A, C \rangle$	`comm.A.T`
$\alpha.2$	T	$\rightarrow$	A	$: \mathsf{S}_{sk_T}(pk_C, C)$	`comm.T.A`
$\boldsymbol{\alpha.3}$	A	$\rightarrow$	C	$: \mathsf{E}_{pk_C}(n_A, A)$	`take.A.C`
$\beta.1$	$I(A)$	$\rightarrow$	T	$: \langle A, B \rangle$	`fake.A.T`
$\beta.2$	T	$\rightarrow$	$I(A)$	$: \mathsf{S}_{sk_T}(pk_B, B)$	`take.T.A`
$\boldsymbol{\beta.3}$	$I(A)$	$\rightarrow$	B	$: \mathsf{E}_{pk_B}(n_A, A)$	`fake.A.B`
$\beta.4$	B	$\rightarrow$	T	$: \langle B, A \rangle$	`comm.B.T`
$\beta.5$	T	$\rightarrow$	B	$: \mathsf{S}_{sk_T}(pk_A, A)$	`comm.T.B`
$\boldsymbol{\beta.6}$	B	$\rightarrow$	$I(A)$	$: \mathsf{E}_{pk_A}(n_A, n_B)$	`take.B.A`
$\boldsymbol{\alpha.6}$	C	$\rightarrow$	A	$: \mathsf{E}_{pk_A}(n_A, n_B)$	`fake.C.A`
$\boldsymbol{\alpha.7}$	A	$\rightarrow$	C	$: \mathsf{E}_{pk_C}(n_B)$	`take.A.C`
$\boldsymbol{\beta.7}$	$I(A)$	$\rightarrow$	B	$: \mathsf{E}_{pk_B}(n_B)$	`fake.A.B`

Figure 3.3: Lowe's attack on seven-message version of NSPK: The right hand side is the FDR output equivalent of the messages on the left hand side. Moreover, the steps in bold, like $\boldsymbol{\alpha.3}$, represent the attack for a three-message version of the NSPK protocol where there is no server, T, involved. Finally, $I(A)$ models impersonating Alice, hence, the intruder can intercept the messages coming to Alice and send fake messages as if Alice is sending them.

authenticated session with Alice, when he is actually talking to the intruder. Therefore, when Bob wants to send a secret $BtoA$ to Alice, he will send it by encrypting it with the nonce n_B, $\mathsf{E}_{k_{n_B}}(BtoA)$, which is the shared secret key. As the intruder possesses n_B and therefore can extract $BtoA$, there is a violation of the secrecy property of the NSPK protocol.

Lowe [Low96] suggested a fix for this situation by adding the sender's identity in step 6 of the seven-message version of the NSPK. In the previous attack, it was explained how the intruder is able to receive $\mathsf{E}_{pk_A}(n_A, n_B)$ from Bob in step $\beta.6$ and replay it to Alice via Cameron in $\alpha.6$. However, with Lowe's fix he is not able to perform the same actions, as there is the sender's identity in the message that he receives in step $\beta.6$, $\mathsf{E}_{pk_A}(n_A, n_B, B)$. Therefore, when he replays this message to Alice, which includes Bob's identity, she would notice that the original sender of this message was not Cameron, but Bob, thereby being able to avoid the attack.

3.3.5 Summary

This above section has introduced the Dolev-Yao intruder model for CSP, which is active and stronger than the observer defined in the Dining Cryptographers Problem covered in the previous section. Additionally, an efficient method for implementing the intruder model (lazy spy) in CSP, which can also be checked

automatically using FDR, has been described. In terms of modelling cryptographic protocols, the NSPK protocol was modelled and analysed with respect to the secrecy property as is a common desired in relation to voting systems. In Chapter 7 this intruder model is adapted for the analysis of cryptographic voting systems, specifically vVote and although it is modified a lot for this purpose, the main idea underpinning the analysis remains the same.

In the next chapter the anonymity requirement for voting systems is formally defined, and subsequently applied to conventional voting system analysis under the passive intruder model as categorised in Section 3.2, using the CSP language and the FDR model checker.

Chapter 4

Formal Anonymity Definition and Automated Verification

This chapter* presents a novel approach to defining a formal anonymity specification as well as modelling and analysis of non-cryptographic voting systems using CSP and FDR. To this end, a number of anonymity definitions in the literature are investigated in Section 4.2 and subsequently the *weak anonymity* is formally defined in Section 4.3 in CSP for automated verification. Moreover, the *strong* and *weak* anonymity specifications are tested in terms of their suitability for voting systems with a referendum example using the abstraction methods introduced in Section 3.2. Additionally, the conventional voting system (CVS) is modelled and analysed with respect to these specifications in Section 4.4 and in Section 4.5 there is further analysis which considers the possibility of corrupt agents as described in the previous chapter regarding dining cryptographers problem, where sensitive data may be leaked to an intruder. Finally, in Section 4.6 a discussion on the formal definitions of anonymity for voting systems is presented.

4.1 Overview

Anonymity of one's vote lies at the heart of the democratic process, for if the link between a voter and her vote is uncovered, then not only the secrecy but also the integrity of the election is threatened, because votes may be bought, or the voters may be coerced into supporting particular candidates. As rigorous protocol analysis requires a concise formal definition of the properties, the anonymity property also needs to be clearly defined. However, as mentioned previously, formally defining such properties is not straightforward as these can vary for dif-

*This chapter is mainly based on the published work in the Formal Aspects of Computing Journal [MHS12].

ferent scenarios. Regarding this, there have been a number of research efforts towards formally defining the anonymity property in the context of voting systems. However, little work has been undertaken aimed at providing a foundation for the automated verification of this property.

4.2 Related Work

First, drawing on the existing literature, several approaches to anonymity are considered.

Pfitzmann *et al.* [PK00] define anonymity in a message sender-receiver setting, where it is specified as the state of someone not being identifiable within a given *anonymity set* of subjects (a set of all possible subjects who might cause an action). In a voting context, this would mean that no specific vote is linkable to any particular voter ID. In addition, an element from the anonymity set possesses *indistinguishability* if it is indistinguishable from all other elements in the set. In terms of voting, this would naturally mean the inability to distinguish a particular vote from within a set of votes. *Unobservability* describes when an intruder cannot observe that a particular event has occurred, for example, that a particular voter has voted. Finally, the term *pseudonymity* describes the use of pseudonyms as identifiers of subjects. For instance, ballot serial numbers can be considered as pseudonyms that link voters to ballot papers and the latter to votes.

Fournet and Abadi [FA02] give a general privacy definition in the pi calculus with respect to private authentication protocols. They define anonymity as the case where "two process behaviours have the same interpretation on the model as long as they are indistinguishable by observation in all contexts." That is, two user processes U_1 and U_2 are identical in any context from the environment's point of view; in what follows, this is defined as *weak anonymity*. Moreover, in their description, an observational equivalence notion[†] is employed to formalise properties. Mauw *et al.* [MVd04] define anonymity based on the work in [PK00] described above in that a coercer should not be able to distinguish a user u from another user u' in the anonymity group of u. That is, for every behaviour of the system that can be attributed to user u, there is another indistinguishable system behaviour that can be attributed to u'. Shmatikov and Hughes [HS04] give a specification framework for anonymity and privacy based upon a view in which system behaviour is described as a set of functions and the desired properties are defined with observational equivalence using a modular approach. In their paper, several forms of anonymity in terms of a sender-receiver relation

[†]The observational equivalence notion in this context is the analogue of the trace equivalence notion in CSP that are used in the definitions of anonymity in the next sections.

are described and some of those definitions are adopted in this work as they are deemed applicable to the voting scenario, as described next.

- *Absolute voter anonymity*: an attacker cannot tell anything about the voter's identity, as every voter is plausible for every observed vote. In addition, in this model an attacker should not be able to link a pseudonym (for example, a ballot serial number) with a sender ID (voter). It should be noted that this strong form of anonymity corresponds to that discussed in Chapter 3.

- *Type-anonymity*: an attacker may learn the type of voter. That is, in the case of postal voting, if there are relatively few voters who have registered and cast their votes by post, an attacker may in some cases be able to reduce the number of possible voters for a particular vote to a proper subset of the set of voters (either the set of postal voters or its complement).

- *Session-level*: an attacker may know the entire set of voters and their votes, but is unable to link the latter to the former's identities during an election (a *session* in their definition). For instance, if an attacker is observing a polling station where only one vote has been cast, and each polling station declares their results separately, he may be able to deduce the voter's identity.

- *Unobservability*: an attacker should not be able to identify that a particular voter has cast a vote; that is, a voting act should be unobservable.

- *Untraceability*: an attacker or an observer should not be able to determine whether two votes cast in different locations have been cast by the same voter.

Taking a different approach, Juels *et al.* [JCJ05] describes anonymity as the property of privacy, where the coercer or adversary cannot guess how a voter voted better than an adversarial algorithm whose only access is the final tally. Although Kremer and Ryan [KR05] and Delaune *et al.* [DKR06, DKR09] also consider the issue of privacy, they adopt Fournet and Abadi's general privacy definition [FA02] in pi calculus for voting system protocols. Moreover, Delaune *et al.* use the term "vote privacy" as a synonym for anonymity, and look for cases where nobody has enough information to identify whether two voters swapped their votes. That is, if an observer cannot tell whether two arbitrary honest voters swapped their votes, then he cannot deduce information about how they cast them and some of the applications of this notion of anonymity include [BHM08, DRS08, Smy11, CCK12].

In the next section, the appropriateness of two formal anonymity definitions: Schneider and Sidiropoulos's [SS96] and Fournet and Abadi's [FA02], are investigated, being denoted as *strong* and *weak* anonymity, respectively.

4.3 Formal Definition of Anonymity

This section discusses the applicability of the strong anonymity definition with a referendum example using the abstraction methods introduced in Section 3.2. Subsequently, both strong and weak anonymity are assessed in the context of voting systems using CSP and FDR.

The strong anonymity definition has already been provided in Section 3.2.1 whereby a process P is said to be *strongly anonymous* on the alphabet $A \subset \Sigma$, if $P[\![^x/_y \mid x, y \in A]\!] \equiv_{\mathrm{T}} P$, and it was shown that it is equal to $P[\![^\beta/_A]\!][\![^A/_\beta]\!] \equiv_{\mathrm{T}} P$, where $\beta \notin \alpha P$. The first relation maps all the events on the alphabet A to a single event β, and the second maps β to any event in A. Thus, if every event of P from the alphabet, A, is renamed to the event, β (many-to-one renaming), then whenever an event, $a \in A$, is possible for P to perform, $P[\![^\beta/_A]\!]$ can perform its image β. Conversely, $[\![^A/_\beta]\!]$ is one-to-many renaming, which maps the event β back to the alphabet A. Thus, when β is possible for the renamed process $P[\![^\beta/_A]\!]$, any event in A is also possible for $P[\![^\beta/_A]\!][\![^A/_\beta]\!]$.

With the following referendum example, the aim is to examine the strong anonymity definition for voting systems as well as to demonstrate the appropriateness of abstraction methods in CSP and effectiveness of FDR in analysing such protocols.

4.3.1 Referendum Example

The following example of a referendum[‡] involves two possible voters v_1 and v_2, with only one of them voting, either for or against it. However, the voter v_1 always says *yes* (if he votes at all), and similarly the other voter v_2 always says *no*. As a result, the process modelling this behaviour is defined as the process *Ref* by:

$$Ref \mathrel{\widehat{=}} vote.v_1 \rightarrow yes \rightarrow STOP \;\Box\; vote.v_2 \rightarrow no \rightarrow STOP$$

If the aim is to verify whether the process satisfies strong anonymity, it is necessary to check the trace equivalence $StrongSpec_A(Ref) \equiv_{\mathrm{T}} Ref$ for the set $A = \{|\, vote.v_1, vote.v_2 \,|\}$, where

[‡]A referendum is a form of democracy, whereby the voters are asked to accept or reject a particular proposal and hence, generally, the ballots consist of *yes* and *no* options.

$$\begin{aligned} StrongSpec_A(Ref) &\mathrel{\widehat{=}} Ref[\![^A/_A]\!] \\ &\mathrel{\widehat{=}} vote.v_1 \rightarrow (yes \rightarrow STOP \,\Box\, no \rightarrow STOP) \\ &\quad \Box\, vote.v_2 \rightarrow (no \rightarrow STOP \,\Box\, yes \rightarrow STOP) \end{aligned}$$

However, the trace equivalence above does not hold, because $\langle vote.v_1, no \rangle$ is a trace of $StrongSpec_A(Ref)$, but not of *Ref*. This has happened because the 'no' vote is sufficient to identify the voter. However, if the events in $H = \{|\, yes, no \,|\}$ are hidden from the observer, then the abstracted process Ref^{abs_1} is defined as:

$$\begin{aligned} Ref^{\mathsf{abs}_1} &\mathrel{\widehat{=}} Ref \setminus H \\ &\mathrel{\widehat{=}} vote.v_1 \rightarrow STOP \,\Box\, vote.v_2 \rightarrow STOP \end{aligned}$$

Now, when the strong anonymity definition is applied to Ref^{abs_1}, the resulting process $StrongSpec_A(Ref^{\mathsf{abs}_1})$ has the same trace as Ref^{abs_1}, and the specification is met.

Additionally, the observer can be limited so that he can see the occurrence of events, but is unable to identify which the process is performing. For example, imagine that the votes are cast in envelopes, by using the renaming operator on the set H above, the sensitive data can be abstracted away. The new abstracted process Ref^{abs_2} can be written as:

$$\begin{aligned} Ref^{\mathsf{abs}_2} &\mathrel{\widehat{=}} Ref[\![^{envelope, envelope}/_{yes, no}]\!] \\ &\mathrel{\widehat{=}} vote.v_1 \rightarrow envelope \rightarrow STOP \,\Box\, vote.v_2 \rightarrow envelope \rightarrow STOP \end{aligned}$$

In order to verify whether Ref^{abs_2} provides anonymity, the trace equivalence

$$StrongSpec_A(Ref^{\mathsf{abs}_2}) \equiv_{\mathrm{T}} Ref^{\mathsf{abs}_2}$$

needs to be checked. As the equality holds for the set $A = \{|\, vote.v_1, vote.v_2 \,|\}$, this demonstrates that the abstracted process Ref^{abs_2} provides strong anonymity.

Another abstraction method that may be used is the *masking*, through which the sensitive information carried by the events can be masked during the protocol using all the same events as noise. For instance, the process *Ref* can be written as the parallel composition of *Ref* with $RUN(H)$, where H is the set of events to be abstracted, namely *yes* and *no* events. Hence, the abstracted process Ref^{abs_3} as follows:

$$Ref^{\mathsf{abs}_3} \mathrel{\widehat{=}} Ref \,|||\, RUN(H)$$

can perform any event from the set H, and the observer cannot tell whether the occurrence of such events is from Ref^{abs_3} or from $RUN(H)$. Hence, the equality

$$StrongSpec_A(Ref^{\mathsf{abs}_3}) \equiv_{\mathrm{T}} Ref^{\mathsf{abs}_3}$$

holds, meaning that the process Ref^{abs_3} with the vulnerable events masked provides strong anonymity.

However, if it is assumed that both voters are participating in the referendum, and each can vote for any candidate they want, but only once, the new referendum process can be described as follows:

$$\begin{aligned} Ref_{new} \mathrel{\widehat{=}} & vote.v_1 \rightarrow yes \rightarrow vote.v_2 \rightarrow no \rightarrow STOP \\ & \Box\ vote.v_1 \rightarrow no \rightarrow vote.v_2 \rightarrow yes \rightarrow STOP \end{aligned}$$

Subsequently, the new process with the sensitive information abstracted is:

$$Ref_{new}^{\mathsf{abs}} \mathrel{\widehat{=}} Ref_{new} \setminus H$$

Hence, the trace equivalence that needs to be checked is:

$$StrongSpec_A(Ref_{new}^{\mathsf{abs}}) \equiv_{\mathrm{T}} Ref_{new}^{\mathsf{abs}}$$

However, the trace equivalence above does not hold when checked using FDR, because the counter-example produced by it, $\langle vote.v_1, vote.v_1 \rangle$, shows that voter v_1 can vote twice, whereas Ref_{new} does not let this trace happen. To sum up, the strong anonymity definition, as discussed in Chapter 3, requires that any voter is plausible for any vote and that she can vote multiple times. That is, once the attacker has discovered that v_1 cast a particular vote, he then knows that v_1 did not cast any of the others. This is something that is true of voting systems generally, but not true under strong anonymity. Therefore, this definition of anonymity is not appropriate for those voting systems where a voter is allowed to vote only once.

The anonymity definition given in [FA02], however, requires two voters swapping their votes. That is, the occurrences of $vote.v_1$ and $vote.v_2$ events in the processes need to be swapped, which solves the problem faced in the strong anonymity definition. However, the sensitive information *yes* and *no* still needs to be abstracted away from the observer's point of view by using one of the techniques described previously. To this end, the anonymity definition due to Fournet and Abadi (weak anonymity) can be described as the following.

Definition 2 (Weak Anonymity)**.** *The process P is weakly anonymous on a set of channels C of type T if:*

$$P[\![^{c.x,\, d.x}/_{d.x,\, c.x} \mid x \in T]\!] \equiv_{\mathrm{T}} P$$

for any $c, d \in C$

This states that the process has the same behaviours if any two channels from C are swapped. That is, if $c.x$ and $d.x$ within P are consistently swapped for all values of x, then the result is indistinguishable from the original protocol behaviour from an observer's point of view. The ability to swap them without making any difference provides anonymity with respect to the channel that has been used. Note that in the following $[\![^{c,\,d}/_{d,\,c}]\!]$ is written as shorthand for $[\![^{c.x,\,d.x}/_{d.x,\,c.x} \mid x \in T]\!]$.

In the context of voting, suppose two honest voters v_a and v_b cast their votes for candidates c_x and c_y, modelled by the occurrence of events $vote.v_a.c_x$ and $vote.v_b.c_y$. The weak anonymity definition for a voting system model, *System*, is applied on the set of channels $C = \{|\, vote.v_i \mid v_i \in Voters \,|\}$, where any v_a, v_b values in the specification can be defined as:

$$WeakSpec_C(System) \mathrel{\widehat{=}} System[\![^{vote.v_a,\,vote.v_b}/_{vote.v_b,\,vote.v_a}]\!]$$

If the following refinement check holds, then the voting system provides anonymity under this definition.

$$WeakSpec_C(System) \equiv_{\mathrm{T}} System$$

It follows that strong anonymity on A implies weak anonymity on channels contained within A, as stated in the following lemma:

Lemma 2. *If P is strongly anonymous on A and $\{|C|\} \subseteq A$, then P is weakly anonymous on channels C.*

Proof. Assume P is strongly anonymous. Then consider some arbitrary $c, d \in C$:

$$\begin{array}{lll} P[\![^{c,\,d}/_{d,\,c}]\!] & \equiv_{\mathrm{T}} \; P[\![^{A}/_{A}]\!][\![^{c,\,d}/_{d,\,c}]\!] & \text{by strong anonymity on } A \\ & \equiv_{\mathrm{T}} \; P[\![^{A}/_{A}]\!] & \text{since } [\![^{A}/_{A}]\!] \,;\, [\![^{c,\,d}/_{d,\,c}]\!] = [\![^{A}/_{A}]\!] \\ & \equiv_{\mathrm{T}} \; P & \text{by strong anonymity on } A \end{array}$$

□

4.4 Modelling and Analysis of a Conventional Voting System

This section presents a formal modelling and analysis of the conventional voting system (CVS) with respect to the anonymity requirement, a system that is still under use in many countries, such as the UK. Under such a system, although the voter is not able to audit her vote once it is cast, the CVS, in the vast majority of

cases, provides anonymity since the voters drop their anonymised ballots into a ballot box, which shuffles the anonymous ballots further. Therefore, by checking this voting system with the weak anonymity specification, further confidence can be gained regarding the suitability of the formal definition of anonymity for voting systems. Moreover, it will demonstrate the ability of CSP in modelling a simple voting system and of FDR for automatically checking whether the CVS model satisfies the anonymity requirements.

The model is described by means of the processes shown at the top of Figure 4.1. That is, it is a parallel composition of these individual processes. Each process has its own process definition and they are run in parallel, synchronising on the common events in order to model the message flow between the agents of the protocol. In the rest of this section, the process definitions will be given for each agent, and the composition of the model presented. Following this, the ability of the intruder will be described and, finally the CVS model will be analysed in terms of the strong and weak anonymity definitions.

4.4.1 Modelling Assumptions

Although the modelling assumptions are emphasised where necessary when presenting CSP process definitions of the voting system model, the CVS model is based on the following assumptions.

- The model consists of a limited number of agents, which are assumed to work honestly: a voter, election official, booth, ballot box and a counter process. The message sequence chart of the communication between these individual processes are shown in Figure 4.1. As there are multiple voters in the model, each voter will follow this message sequence and each election phase.

- The voters are also assumed to choose a candidate to vote for before the registration phase because of the adaptation of the anonymity specification. In more detail, once the voter has started taking part in the election by registering to the election official, the voter cannot be replaced with another voter—this is required by the anonymity specification, which is focused in Section 4.3. Hence, if the voter is allowed to choose the candidate she would like to vote for during the election process, this would cause false positives. Therefore, choosing the candidate before the registration is an assumption that should be made in order to eliminate false positives and also it is not so unrealistic.

- Yet another assumption is made on the number of booths in a polling station. In the model there exist only one booth which synchronises with

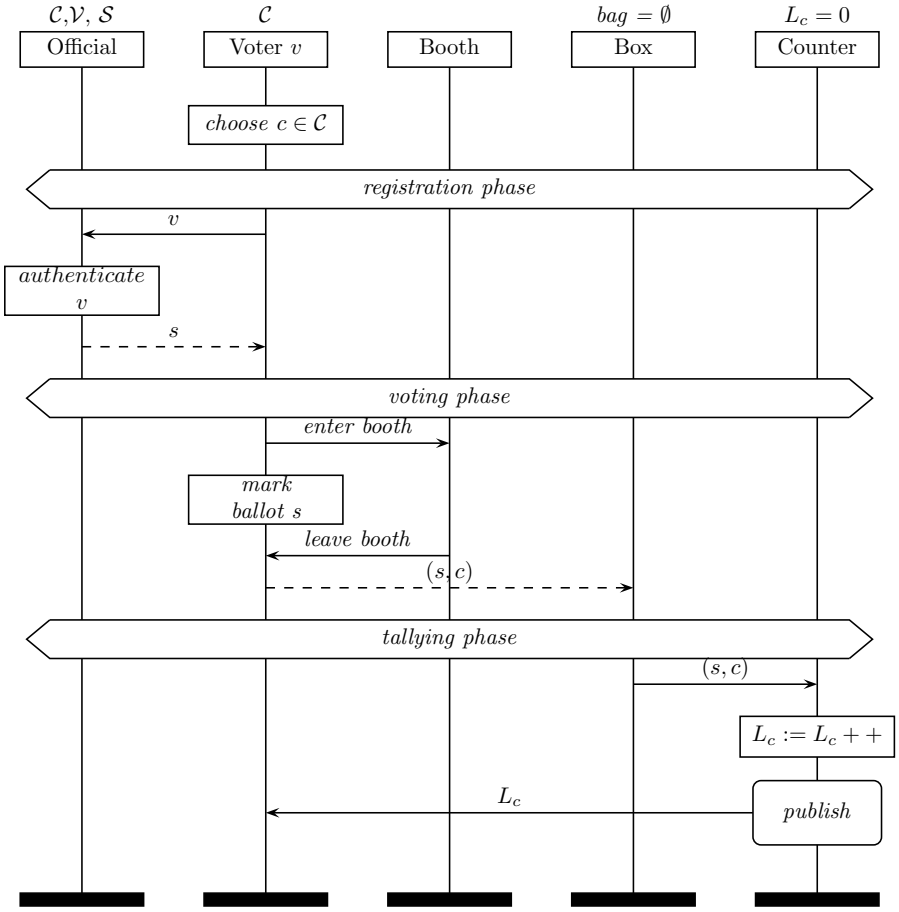

Figure 4.1: Message sequence chart of CVS

the voters and does allow multiple voters to be in and casting a vote at the same time. There could have been more than two booths. However, having checked the system behaviours against the anonymity specification, this assumption has no harm in the formal analysis of voting systems. This is because the booth only allows the voters to mark their ballot forms in a private environment. Thus, in a polling station with multiple booths voters simply vote in separate booths, and queue to drop their envelopes into the ballot box as usual.

4.4.2 Honest Participants

In the CVS, the correctness of the election results, the secrecy and the verifiability of the votes depend on a chain-of-custody, whereby each step of the protocol is verified by a trusted third party or election officials. Hence, it is first assumed that

each participant is honest and follows the protocol steps to ensure its objectives are met, in particular, the voter anonymity. The honest agent processes of the CVS model are modelled as follows.

Voter Process

A *Voter* from the set of voters chooses a candidate to vote for from the given candidate list before going to the polling station and identifying herself to the electoral official. To this end, the choosing action for the voter v is modelled with the events $choose.v.c$, where $c \in \mathcal{C}$. As the choice of candidate is made by the voter, non-deterministic choice is the appropriate CSP operator, since the choice is not under the control of the system. Afterwards, she receives a ballot form with a serial number s on it on the channel *collectform*. In more detail, as the ballot form is given by the authority to the voter, an external choice operator is used to show that the voter accepts any ballot form given by the authority. Subsequently, she goes into a booth, votes according to her preference on the channel $mark.v.s.c$, modelling v votes for c on the ballot paper with the serial number s. Finally, she leaves the booth, casts her vote by dropping the ballot form in the ballot box, modelled as $cast.v.s.c$, and leaves the polling station. Hence, the following process models the honest behaviour of a voter.

$$
\begin{aligned}
&Voter(v) \mathrel{\widehat{=}} \\
&\quad \prod_{c \in \mathcal{C}} \left(\begin{array}{l} choose.v.c \rightarrow openElection \rightarrow auth.v \rightarrow \\ \quad \Box_{s \in \mathcal{S}} \left(\begin{array}{l} collectform.v.s \rightarrow \\ enterBooth.v \rightarrow \\ mark.v.s.c \rightarrow \\ leaveBooth.v \rightarrow \\ cast.v.s.c \rightarrow \\ closeElection \rightarrow STOP \end{array} \right) \end{array} \right)
\end{aligned}
$$

In the model, all the voters can be described as the interleaving of voter processes for each as below.

$$Voters \mathrel{\widehat{=}} \|_{v \in \mathcal{V}} Voter(v)$$

The alphabets of the processes are not explicitly stated here, but are taken to be the set of all the events they can perform. For instance, the alphabet of the process *Voters* is shown below. Likewise, the alphabets of the other processes modelled include all the events that can be performed by these processes.

$$
\begin{aligned}
\alpha Voters = \{\!|\ & openElection,\ auth,\ collectform,\ enterBooth,\ leaveBooth, \\
& choose,\ mark,\ cast,\ closeElection |\!\}
\end{aligned}
$$

Election Official Process

An *Election Official* working in a polling station authenticates eligible voters by their identification documents, and issues the ballot papers on which there are arbitrary and unique serial numbers. In the model, a set of pre-existing serial numbers are assigned to the voters by the election official, which is performed non-deterministically as this official chooses them independently. Moreover, the same serial number is never given twice, so two different voters cannot receive the same one (the same ballot form) to vote with. The election official process also opens and closes the election for a polling station and other processes synchronise with this official on the *openElection* and *closeElection* channels, thereby maintaining the different phases of the election.

$$
\begin{array}{l}
Elecofficial \mathrel{\widehat{=}} openElection \rightarrow Official(\mathcal{V}, \mathcal{S}) \\
Official(ids, serials) \mathrel{\widehat{=}} \\
\quad closeElection \rightarrow STOP \\
\quad \Box \\
\quad \Box_{v \in ids} \left(\begin{array}{l} auth.v \rightarrow \\ \quad \sqcap_{s \in serials} \left(\begin{array}{l} collectform.v.s \rightarrow \\ Official(ids \setminus \{v\}, serials \setminus \{s\}) \end{array} \right) \end{array} \right)
\end{array}
$$

Booth Process

A *Booth* is a private environment for the voters to cast their ballots without being observed. Thus, in the model, the booth process allows one voter to go in to vote and to leave before the next is allowed to enter.

$$Booth \mathrel{\widehat{=}} enterBooth?id \rightarrow leaveBooth.id \rightarrow Booth$$

Ballot Box Process

A *Ballot Box* is a box where all cast votes are collected under the control of the election official. It is assumed that there is a private untappable channel between a voter and a ballot box (or, in other words, the voter fills in the ballot paper and casts the ballot unobserved). In this model, the ballot box accepts the ballots from the voters and gathers them for collection. Hence, whenever a $cast.v.s.c$ occurs the process will store the tuple (s, c) in a set. Once the election is closed, the box can be opened, and all the ballots can be withdrawn for the tallying, with this event from the set *bag* being performed non-deterministically, this complies with the anonymous behaviour of a ballot box. Finally, when there are no more ballots left in the box, the process terminates.

$$Box \mathrel{\widehat{=}} openElection \rightarrow Box_1(\emptyset)$$
$$Box_1(bag) \mathrel{\widehat{=}} closeElection \rightarrow Box_2(bag) \;\Box\; \Box_{\substack{v\in\mathcal{V}\\ s\in\mathcal{S}\\ c\in\mathcal{C}}} \Big(cast.v.s.c \rightarrow Box_1(bag \cup \{(s,c)\}) \Big)$$
$$Box_2(\emptyset) \mathrel{\widehat{=}} empty \rightarrow STOP$$
$$Box_2(bag) \mathrel{\widehat{=}} \prod_{(s,c)\in bag} withdraw.s.c \rightarrow Box_2(bag \setminus \{(s,c)\})$$

Counter Process

A *Counter* is an election official who removes all cast ballots from the ballot box and tallies them. The following process models the counter's behaviour for each candidate keeping a record of *withdraw* events for each candidate. When a *withdraw* event happens, the counter checks for which candidate the particular vote is, and subsequently increments the number of votes that he/she has received so far by one. Once the ballot box is empty, meaning there is no ballot to be counted, he announces the total number of votes that each candidate has received.

$$Counter(c,r) \mathrel{\widehat{=}} \Box_{s\in\mathcal{S}} \Big(withdraw.s.c \rightarrow Counter(c, r+1) \Big) \;\Box\; empty \rightarrow total.c.r \rightarrow done \rightarrow STOP$$

$$Counters \mathrel{\widehat{=}} \|_{c\in\mathcal{C}} Counter(c,0)$$

The system for the conventional voting is defined as a parallel composition of all of the above described five processes, as seen below, and the correctness of the model is verified via a number of sanity checks, such as: "no voter is allowed to vote after the election is closed" and "the correctness of final tally". These sanity checks increase the confidence in the behaviour of the model (see Appendix A.1).

$$System_{CVS} \mathrel{\widehat{=}} Voters \parallel Elecofficial \parallel Booth \parallel Box \parallel Counter$$

Having modelled the CVS, in the next subsection, the passive attacker or observer's behaviour is defined.

4.4.3 The Passive Attacker

In this analysis, it is assumed there exists a passive attacker or an observer who aims to break voter anonymity, and is capable of seeing all the public information

over the election protocol. That is, the intruder's ability is restricted to the public data, which needs to be specified carefully.

Firstly, as in the real traditional elections, under the model anyone knows when the election is opening and closing, and what the final result is for each candidate in the end. Additionally, a list of all voters who are eligible to vote is hung on a public board in the polling station before the election and remains so during it. Hence, it is assumed that the intruder can observe who takes part in an election run as well as the voters going in and out of the voting booth. It is further accepted that the intruder can also see the ballot forms withdrawn from the ballot boxes and the tellers counting them.

However, the intruder cannot access any information that can link a voter to her ballot form and/or to her chosen candidate. For instance, if the intruder sees the ballot form on which a voter has cast her vote, then he can violate her anonymity as he can also see the tallying of the votes at the end of the election. Hence, the channel between the voter and the election official, where the ballot form is given to the voter needs to be hidden, i.e., private channel. Obviously, the voter's marking her ballot form also needs to be concealed from the observer.

An initial description of the system that the observer can see is put forward as:

$$System'_{CVS} \mathrel{\widehat{=}} System_{CVS} \setminus \{| mark, collectform |\}$$

This process models the conventional voting system that the intruder observes, in which the *mark* and *collectform* events are hidden. However, a number of other scenarios could be tried, like hiding the *cast* events completely, thus not allowing the observer to see the voter casting her vote. By contrast, he could be allowed to observe that the voter is casting a vote, but not for whom or what serial numbers are used. In order to do this, *cast* events have to be renamed to another event called *envelope*, which models the voter casting her vote in an envelope rather than in public, thereby hiding the private data. Consequently, the process $System^{\mathsf{abs}}_{CVS}$ below models this behaviour of the CVS model.

$$System^{\mathsf{abs}}_{CVS} \mathrel{\widehat{=}} System'_{CVS}[\![^{envelope}/_{cast.id.s.c}]\!]$$

4.4.4 Strong Anonymity Analysis

As noted by Schneider and Sidiropoulos in [SS96], different definitions of anonymity are required for different situations. For instance, in a voting system where the anonymity of the voters' identity is required, the strong anonymity definition that was given previously is too powerful, because multiple votes are allowed under this arrangement, which is not the case for all voting systems as

they only permit the casting of one ballot, even though there might be several candidates voted for. To see this, the strong anonymity specification for the CVS model can be defined as the following:

$$StrongSpec_A(System^{\mathsf{abs}}_{CVS}) \mathrel{\widehat{=}} System^{\mathsf{abs}}_{CVS}[\![^{dummy}/_{choose.v.c_1}]\!][\![^{choose.v.c_1}/_{dummy}]\!]$$

The anonymity is checked for the set $A = \{|\, choose \,|\}$, hence if the strong anonymity definition is applied to the CVS model, the *choose* events are renamed to the event $dummy \notin A$, and renamed back to the set A. Hence, in order for the CVS model to provide strong anonymity, the following trace equivalence should be satisfied.

$$StrongSpec_A(System^{\mathsf{abs}}_{CVS}) \equiv_{\mathrm{T}} System^{\mathsf{abs}}_{CVS}$$

FDR produces the counter-example trace of $\langle choose.v_1.c_1, choose.v_1.c_1 \rangle$, meaning that the system does not satisfy this specification as the abstracted model $System^{\mathsf{abs}}_{CVS}$ does not allow the voter v_1 to vote twice in an election run. As a result, the CVS model does not provide the strong anonymity given in [SS96] from the observer's point of view.

The next subsection investigates the suitability of the weak anonymity definition for voting systems.

4.4.5 Weak Anonymity Analysis

The weak anonymity definition formalised in Section 4.3 will be used in the second analysis. This analysis of the CVS model is conducted by comparing two situations: the first, in which the voters v_1 and v_2 vote any way they like; and the second, in which the voters swap their votes. From the intruder's point of view, the processes $System^{\mathsf{abs}}_{CVS}$ (the first situation) and $WeakSpec_A(System^{\mathsf{abs}}_{CVS})$ (the second) should be indistinguishable. Hence, according to the weak anonymity definition, the specification is defined by swapping two votes, (i.e., $choose.v_2.c$ and $choose.v_1.c$, where $c \in \mathcal{C}$) using the renaming abstraction method as in the following:

$$WeakSpec_A(System^{\mathsf{abs}}_{CVS}) \mathrel{\widehat{=}} System^{\mathsf{abs}}_{CVS}[\![^{choose.v_1.c,\, choose.v_2.c}/_{choose.v_2.c,\, choose.v_1.c}]\!]$$

In order for the CVS model to provide anonymity under the weak anonymity definition, the following trace equivalence should hold:

$$WeakSpec_A(System^{\mathsf{abs}}_{CVS}) \equiv_{\mathrm{T}} System^{\mathsf{abs}}_{CVS}$$

FDR verifies that the equality holds meaning that the two systems are indistinguishable from the intruder's point of view and therefore, the conventional voting system model provides anonymity under this definition.

In the next section, the case of corrupt agents in the protocol who can share their knowledge or observations with the intruder will be considered. Note that the weak anonymity definition is to be used in the remainder of this thesis as the valid anonymity definition for voting systems, i.e., when anonymity is mentioned, the weak definition is being referred to.

4.5 Analysis under Alternative Assumptions

For the dining cryptographers problem, the anonymity of the protocol was previously checked in relation to two cases: from an outsider's point of view and with respect to other cryptographers. Similarly, the analysis of voting systems in general and in the CVS model, specifically can be conducted under such assumptions. For instance, a corrupt election official may share their knowledge with the intruder, thereby enlarging his power over sensitive data, which can subsequently be used to break the anonymity of voters. Under these circumstances, the anonymity can be checked by focusing on election official and this can be modelled in CSP using the abstraction methods: renaming and hiding.

A trusted election official possesses crucial information regarding which ballot form was given to a particular voter and hence he has the power to break the anonymous link between the voter, her ballot form and the chosen candidate. Misbehaviour in this regard can be modelled by revealing the private channel between the voter and the election official and in order to do so, the event *collectform* should not be hidden when defining the abstracted system model $System^{\mathsf{abs}}_{CVS}$. Therefore, only the *mark* events will be abstracted away and this whole process is as follows:

$$System^{\mathsf{abs}}_{CVS} \mathrel{\widehat{=}} (System_{CVS}[\![^{envelope}/_{cast.id.s.c}]\!]) \setminus \{|\, mark \,|\}$$

Following this, when the equivalence $WeakSpec_A(System^{\mathsf{abs}}_{CVS}) \equiv_{\mathrm{T}} System^{\mathsf{abs}}_{CVS}$ is checked mechanically using FDR, it produces the following counter-example.

$\langle choose.v_3.c_3,$
$choose.v_2.c_1,$
$choose.v_1.c_3,$
$openElection,$
$auth.v_1,$
$collectform.v_1.s_2,$
$auth.v_3,$
$collectform.v_3.s_3,$
$auth.v_2,$
$collectform.v_2.s_1,$
$envelop,$
$envelop,$
$envelop,$
$closeElection,$
$withdraw.s_1.c_1\rangle$

The trace above identifies an attack against the anonymity of the voter v_2. In more detail, although the observer (electoral official in this case) cannot see the votes, because they are cast in an envelope, he knows which serial number the voter v_2 was given by observing the event $collectform.v_2.s_1$. When the votes begin to be counted, the corrupt official can also observe the occurrence of the $withdraw.s_1.c_1$ event, which links the serial number s_1 to the candidate c_1. As the serial number s_1 was cast by the voter v_2, the observer is now certain about how v_2 has cast her vote, thus violating v_2's anonymity.

4.6 Results and Discussion

The analysis has shown that the strong anonymity definition is too strong for analysing this concept in voting systems, if voters are not allowed to cast multiple ballots. That is, this specification requires an actor to perform different tasks that cannot be linked together in any way, which is not the case in general voting systems as only one such action, namely a ballot, can be cast and counted per voter—although, some schemes allow voters to cast multiple ballots, such as, JCJ [JCJ05] and Civitas [CCM08], only one of the votes is counted at the tallying phase. Hence, the definition is not appropriate for these voting systems either. As a consequence, the weak anonymity was investigated for its suitability to provide a formal specification for voting systems, and it was verified that the CVS model provides such anonymity. Moreover, the appropriateness of the formal language CSP to model voting systems, and the capability of the FDR model checking tool for automated analysis were demonstrated.

Additionally, some of the corner cases have been investigated. For instance, in case of unanimity in an election run, whereby all the voters vote for the same candidate, although it is clear how each voter voted, an observer still cannot identify whether two voters have swapped their votes (because this is a null operation), and so the anonymity definition is still satisfied. Similarly, the definition is still met in elections with an electorate consisting of a single voter, because swapping votes is still applicable even though there is only one vote. Additionally, in the case in which the electoral official can assign the same serial number to two different voters, the weak anonymity is still satisfied by the CVS model. Indeed, assigning one serial number to two different voters may introduce a better anonymity. Similarly, it was also demonstrated that the CVS model satisfies the weak anonymity definition even if there are no serial numbers on the ballot forms.

Moreover, the weak anonymity definition covers voting systems in which the final tally is published. However, although the strong anonymity definition is too strict for most voting systems, it can still be used in systems that allow a voter to vote multiple times in an election, or where only the winner is announced and not the full tally. For instance, strong anonymity may be an appropriate definition for television polls where votes are cast by sending an SMS to a particular number.

4.7 Summary

This chapter has investigated related work on formal anonymity definitions, in particular, *strong* and *weak* anonymity for voting systems. Moreover, a model of the conventional voting system has been used as a case study to validate these definitions, and using the process algebra CSP and the FDR model checker, it was demonstrated how to provide automated analysis of ballot-based non-cryptographic voting systems with respect to the different anonymity definitions. In addition, the importance of underlying assumptions in the formal analysis of a protocol has been highlighted as well as the precise definitions of requirements presented in terms of comparing what an intruder is capable of doing under different definitions and assumptions. In the next chapter there is further investigation into the assumptions under which certain protocols are claimed to provide voter anonymity, such as Rivest's ThreeBallot voting system.

Chapter 5

Automated Analysis of the ThreeBallot Voting System

This chapter* demonstrates the applicability of the framework introduced in the previous chapter, for automated anonymity analysis of non-cryptographic voting systems and emphasises the importance of security protocol assumptions. To this end, Rivest's non-cryptographic voting system ThreeBallot [Riv06] is investigated. It is particularly interesting because it uses no cryptography, however, still aims to provide voter anonymity, integrity of the election, verifiability and incoercibility. Moreover, although the ThreeBallot voting system has been the subject of analysis of one sort or another many times since its publication [Str06b, Str06a, CEA07, App07, dMPQ07, TPR07, CKW08, HSS09, KTV11] (see Section 1.3 for details), it has not yet been subjected to automated formal verification.

In this chapter, a CSP model of ThreeBallot has been constructed and used to produce the first automated formal analysis of its anonymity property using FDR. Throughout the analysis, the anonymity definition given in Section 4.3 Definition 2, and the passive intruder model given in Chapter 4 will be considered. Additionally, various modified versions of ThreeBallot in the literature [KZ10, KTV11] are investigated and it is shown that they suffer from the same attacks despite the improvements, such as *Reconstruction Attacks*. That is, with the information available to the intruder on the bulletin board, he may find out whether the voter has voted for a particular candidate just by matching the mini-ballots on the bulletin board and the voter's receipt to reconstruct valid multi-ballots. Consequently, Rivest and Smith [RS07] proposed the short ballot assumption (SBA), under which ThreeBallot is claimed to be secure. How-

*This chapter is mainly based on the published work for the 10th International Conference on integrated Formal Methods (iFM) [MHS13].

ever, during the analysis of this voting system, it emerges that the SBA is highly ambiguous in the literature. Roughly speaking, this assumption states that the information content of a ballot should be low. However, the phrasing of this assumption in the description of ThreeBallot is vague and open to a number of radically different interpretations. Consequently, various plausible precise interpretations are discussed here, and it is discovered that in each case, the interpretation was either unrealistically strong, or else failed to ensure anonymity. Therefore, a version of the SBA in relation to ThreeBallot that is realistic but still provides a guarantee of anonymity is adopted. Finally, because the approach to the analysis of voting systems considered in this thesis is possibilistic rather than probabilistic, two cases where the ThreeBallot voting system provides guaranteed anonymity without the SBA are verified automatically.

This chapter is structured as follows. In the next section, an outline of ThreeBallot is provided as well as there being discussion on the SBA. In Section 5.2, ThreeBallot is modelled as a parallel composition of agents: voters, an authority, the bulletin board and a counter process. In Section 5.3 the first automated analysis of the ThreeBallot voting system and its versions are presented. Furthermore in that section, the analyses of the SBA interpretations are conducted manually and subsequently a better formulation for this assumption is given. Next, the ThreeBallot versions providing guaranteed anonymity without the SBA are verified using the model checker FDR. Finally, in Section 5.5 the chapter is concluded with a summary of findings.

5.1 The ThreeBallot Voting System

In this section, the original ThreeBallot voting system [Riv06] is briefly introduced as well as the short ballot assumption given by Rivest and Smith [RS07].

Voting in ThreeBallot proceeds as follows. Initially, the authenticated voter receives a multi-ballot from a poll worker, which consists of three mini-ballots (see Figure 5.1). The mini-ballots are all identical except for the IDs or serial numbers, located at the bottom, which are all unique and unrelated. In particular, there is no way of determining what mini-ballot serial numbers go together to make up a multi-ballot. The voter fills two bubbles in total for the chosen candidate, and only one bubble for each other candidate. The completed multi-ballot is inserted into a checker, which confirms that it has been correctly completed. Finally, the voter chooses one of the mini-ballots, and receives a duplicate of it as her receipt. She then separates the three mini-ballots, and casts them all individually into a ballot box. At the end of the election day, the cast ballots are scanned to the BB on which all mini-ballots are published along with a list of everyone who voted. The voter may then verify that the mini-ballot for which she has a receipt

Alice	●	Alice	●	Alice	○
Bob	○	Bob	●	Bob	○
Chris	○	Chris	○	Chris	●
David	●	David	○	David	○
	56248		04578		31489

Figure 5.1: A ThreeBallot multi-ballot, filled in as a vote for Alice

appears unaltered on the BB; if it does not, she can appeal using her receipt as evidence.

The number of votes for each candidate is counted as in the traditional voting system. As each voter fills in exactly two bubbles for the chosen candidate and one for the others, the number of voters can then be subtracted from each candidate's final tally to find the correct number of votes for each candidate, i.e., this identifies those multi-ballots with two bubbles filled out. Subsequently, all the mini-ballots are posted on the bulletin board so the final tally can be verified by anyone and each voter can check whether their vote has been tampered with in any way, using their duplicate as evidence.

ThreeBallot is claimed in [RS07] to be secure under the short ballot assumption (SBA), which Rivest and Smith in [RS07, p.4] defined as:

> "the ballot is short—there are many more voters in an election than ways to fill out an individual ballot [...] It is reasonable to assume under the SBA that each possible ballot is likely to be cast by several voters."

However, ambiguities arise from the terms "Individual ballot" (mini-ballots or multi-ballots?) and "several voters" (how many?) employed by these authors. Moreover, according to [CKW08] the SBA assumes that "the list of candidates on a ballot is short enough in order to guarantee security" and in [dMPQ07] it is stated that "the length of the ballots must be kept small (possibly by splitting them into several parts)". In summary, in the literature no precise meaning of the SBA is to be found.

Because ThreeBallot is claimed to guarantee voter anonymity under the SBA, its analysis is not possible without a clear and unambiguous reading of the assumption. The three possible interpretations of this assumption, which will be analysed later in this chapter, are provided next. Note that in each case the intention is that the assumption will be guaranteed probabilistically; that is, the number of voters, candidates, etc., will be sufficient to ensure that the assumption is broken with only negligible probability. In what follows, serial numbers will be

ignored; that is, two mini-ballots will be considered the same if they contain the same marks but different serial numbers.

Assumption 1 (SBA-multi)**.** *Every possible multi-ballot will be cast at least once.*

The formulation of the SBA given in Assumption 1 requires that every possible way of completing a multi-ballot should be adopted by at least one voter. For small numbers of candidates, this is plausible, but even moderate numbers, though, the assumption quickly becomes less likely to hold.

Note that once a candidate has been chosen, there are then exactly three ways of completing each row: for the chosen candidate's row, one must choose a bubble to leave empty, and for each other row, one must choose a bubble to fill. There are thus $c \cdot 3^c$ distinct multi-ballots, where c is the number of candidates standing in the election. It is not feasible to calculate the number of voters required to make this reasonable, because it depends on the probability distribution of multi-ballots: voters do not cast multi-ballots randomly (one hopes). A full calculation would require a realistic model of how voters cast their ballots. However, the best case scenario is when voters cast their multi-ballots randomly and so by assuming a uniform distribution, we can determine a lower bound on the number of voters required. That is, with a uniform distribution the expected number of voters needed to cover all possible multi-ballot patterns is: $n \cdot \sum_{i=1}^{n} \frac{1}{i}$ where $n = c \cdot 3^c$, is the number of possible multi-ballots. For five candidates, this comes out at 9331 voters; for ten candidates, 8.1 million voters are needed and for fifteen candidates, the number exceeds 4 billion.

For n possible multi-ballots, and a uniform distribution, the number of voters required to ensure that the probability of covering every multi-ballot at least once exceeds a given threshold can be calculated. However, since the security of ThreeBallot relies on the SBA, there would need to be confidence that (the correct interpretation of) it is satisfied by achieving an acceptable probability level pertaining to the number of voters required. For n multi-ballots, and v voters, the probability that the v voters will cover all of the n possibilities is:

$$1 - \sum_{j=1}^{n-1} (-1)^{j+1} \binom{n}{j} \left(\frac{n-j}{n} \right)^v$$

This summary is difficult to calculate precisely but easy to calculate approximately because the first few terms dominate for large v.

For five candidates, to reach 95% probability of full coverage, around 12,250 voters are needed, whereas six candidates need around 50,000 voters and by the time ten candidates are running, 9.6 million voters are required to give 95% confidence that every multi-ballot turns up at least once. Note that these figures are rather

conservative lower bounds: the distribution will not in fact be uniform, which will lower the probability and in any case 95% confidence is perhaps insufficient for a critical security assumption.

However, because these numbers are so high, they are considered to be unrealistic. Therefore, as this version of the short ballot assumption is suitable only for a very small number of candidates or an extremely large numbers of voters; it will not be considered further in this paper.

Assumption 2 (SBA-mini)**.** *Every possible mini-ballot will be cast at least once.*

Under Assumption 2, each mini-ballot is required to be cast at least once, rather than each multi-ballot and clearly this is more likely to be satisfied than Assumption 1. For c candidates, there are only 2^c distinct mini-ballots, against $c \cdot 3^c$ distinct multi-ballots. For ten candidates, coverage of only 1024 mini-ballots is needed, rather than nearly 600,000 multi-ballots. It will be shown later that this interpretation of the SBA is insufficient to prevent attacks on ThreeBallot and since it is not a worthwhile formulation of the assumption, it is not necessary to calculate the likelihood that it will be satisfied.

Assumption 3 (SBA-mini-n)**.** *Every possible mini-ballot will be cast at least n times (for some suitably chosen n).*

A slightly stronger interpretation Assumption 3 requires each mini-ballot to turn up at least a certain number of times and this, of course, needs more voters than Assumption 2. However, it will be shown later that this formulation is also insecure, regardless of the value of n.

5.2 Modelling the ThreeBallot Voting System

In this section, the CSP model of ThreeBallot is given by first defining data-types, sets and the functions, and then describing each process individually. Subsequently, each is then run in parallel so as to reflect the behaviour of the voting system.

In this modelling approach, the multi-ballot of the ThreeBallot voting system is treated as a board with coordinates different to the CVS model. Here, a coordinate (i, j) defines a bubble on a mini-ballot, which is to be filled in—although the bubbles at the end of the each mini-ballot are allocated for serial numbers, the separation between a bubble for a mark and the bubble for a serial number is ensured in the process definitions. Thus, a multi-ballot consists of three columns

		j		
	○	○	○	
i	○	○	○	
	○	○	○	
	56248	04578	31489	

Table 5.1: Bubbles returned by $\mathsf{Row}(i)$ and $\mathsf{Col}(j)$

		j		
	○	○	○	
i	○	○	○	
	○	○	○	
	56248	04578	31489	

Table 5.2: Bubbles returned by $\mathsf{nhdAll}(i, j)$

		j		
	○	○	○	
i	○	○	○	
	○	○	○	
	56248	04578	31489	

Table 5.3: Bubbles returned by $\mathsf{adjR}(i, j)$

		j		
	○	○	○	
i	○	○	○	
	○	○	○	
	56248	04578	31489	

Table 5.4: Bubbles returned by $\mathsf{adjC}(i, j)$

each representing: a mini-ballot; as many rows as the number of candidates racing in the election; and a single row at the end of the ballot allocated for serial numbers. Hence, the size of the board is determined by these parameters: the number of voters and the number of candidates. These parameters define the sets of voters, candidates and serial numbers (there are three times as many serial numbers as there are voters). Conventionally, the data-types for voters, candidates and serial numbers are denoted v, c and s, respectively. Note that the candidate order is predetermined numerically, i.e., the order of the candidate is fixed as c_1,c_2, ..., c_n for all ballots.

In order to return a specific part of the board in a process description, several functions are used in the model. In more detail, the function $\mathsf{Row}(i)$ returns the ith row of a multi-ballot and $\mathsf{Col}(j)$ is the set of bubbles on the jth column (Table 5.1). Likewise, some other functions are used to return the neighbouring bubbles of a given coordinate, such as, the function $\mathsf{nhdAll}(i, j)$, which returns all the neighbours of (i, j) in the current multi-ballot coordinates (Table 5.2). Similarly, $\mathsf{adjR}(i, j)$ returns the coordinates adjacent to (i, j) in the same row (Table 5.3), and $\mathsf{adjC}(i, j)$ returns the coordinates adjacent to (i, j) in the same column (Table 5.4).

5.2.1 Modelling Assumptions

The assumptions made in the modelling of ThreeBallot are continuation of the ones in Section 4.4.1. That is, there is a limited number of voters, all of which

follow the protocol steps honestly and choose the candidates to vote for before the registration phase. The rest of the assumptions regarding ThreeBallot is as follows. The voter is not restricted in how she votes, but the filling of the ballot form needs to be done efficiently in a left to right fashion (i.e., to limit the non-determinism in the CSP). Therefore, this modelling assumption does not impact on the analysis, but facilitates the mechanised verification by reducing the state space (more details are given when describing the voter process in the following subsection).

All cast multi-ballots are valid. That is, in the original ThreeBallot system there exists a checker machine in the booth, which confirms that the multi-ballot inserted by the voter has been correctly completed. In modelling of ThreeBallot, this behaviour is already modelled in the voter process—the voter never makes a mistake when completing a multi-ballot, which are ensured by processes synchronisation. This also means that we do not model incorrect voter behaviours either by mistake or intentionally as in all voting system models analysed in this thesis.

5.2.2 Honest Participants

In this subsection, the individual processes are defined and the channels that connect these processes as well as the information carried on each are explained. The ThreeBallot system model is formed by the parallel composition of the following processes (see Figure 5.2 illustrating the network for the ThreeBallot CSP model).

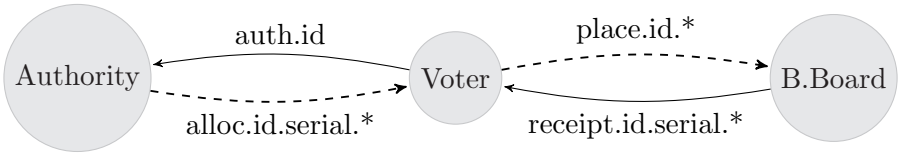

Figure 5.2: ThreeBallot CSP model communication channels (⇢ private channel) and note that the counter process is considered a part of the bulletin board process.

Voter Process

The voter chooses the candidate that she wants to vote for before the election. She then authenticates herself to the election authority, and collects her multi-ballot on the *alloc* channel. In the booth, she fills out two bubbles for the chosen candidate and one for the other candidates with the *place* events. Afterwards, she gets her receipt by choosing one of the mini-ballots allocated to her on the channel *receipt*, and leaves the booth before the election is closed.

The *Voter*() process performs *place* events in an efficient way, such that although it allows any legitimate completed multi-ballot, it reduces the state space by constraining the order in which bubbles are filled. That is, first a bubble from the first or second column is chosen for the candidate the voter wants to vote for (the set F_1 under the second non-deterministic choice in the process definition below determines what bubbles the voter can use first, where $F_1 = \mathsf{Row}(x-1) \setminus \mathsf{Col}(2)$) and x is the chosen candidate's id, such as 2 for the candidate c_2, thus, the value $x-1$ determines the row of the chosen candidate. Following this, the voter chooses another bubble for her chosen candidate in the same row, but in a left to right fashion, e.g., first marking the bubble $(0,1)$, and then $(0,2)$, thus eliminating the option of first marking $(0,2)$ and then $(0,1)$. This is illustrated by the voter choosing a bubble from the set F_2, where $F_2 = \mathsf{adjR}(i,j)$. Afterwards, the process performs a *place* event in a top to bottom manner for the other candidates—again the way that the voters fill in the bubbles is restricted in order to reduce the state space required for the analysis. All empty bubbles that can be filled in by the voter in the next phase are transferred to the set *Rest*, where $Rest = \mathsf{nhdAll}(i,j) \setminus (\mathsf{Row}(i) \cup \mathsf{Row}(n))$ and n is the number of candidates, also corresponds to the serial numbers row on a ballot form. Once the voter has finished voting for the chosen candidate, she marks one bubble for the other candidates, and the row belonging to the particular one is discarded from the set (modelled as $aset \setminus \mathsf{Row}(k)$ in the last line of the process definition). Finally, the voter determines one of the serial numbers as her receipt. That is, the mini-ballot with that serial number and the marked bubbles form her receipt.

$$
\begin{array}{l}
Voter(v) \mathrel{\widehat{=}} \\
\quad \prod\limits_{c \in \mathcal{C}} \left(\begin{array}{l}
choose.v.c \rightarrow openElection \rightarrow auth.v \rightarrow \\
alloc.v?s_1?(i_1,j_1) \rightarrow alloc.v?s_2?(i_2,j_2) \rightarrow alloc.v?s_3?(i_3,j_3) \rightarrow \\
enterBooth.v \rightarrow \\
\quad \prod\limits_{(i_4,j_4) \in F1} \left(\begin{array}{l}
place.v.(i_4,j_4) \rightarrow \\
\quad \prod\limits_{(i_5,j_5) \in F2} \left(\begin{array}{l}
place.v.(i_5,j_5) \rightarrow \\
Voter_1(v, Rest, \{s_1,s_2,s_3\}, n-1)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
Voter_1(v, aset, serials, 0) \mathrel{\widehat{=}} \\
\quad \prod\limits_{s \in serials} \left(\begin{array}{l}
receipt.v.s?(i,j) \rightarrow leaveBooth.v \rightarrow \\
closeElection \rightarrow STOP
\end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
Voter_1(v, aset, serials, r) \mathrel{\widehat{=}} \\
\quad place.v?(k,l) \rightarrow \\
\qquad Voter_1(v, aset \setminus \mathsf{Row}(k), serials, r-1)
\end{array}
$$

Thus the process representing all voters is described by the interleaving of the voters as follows:

$$Voters \mathrel{\widehat{=}} \|_{v \in \mathcal{V}} Voter(v)$$

Election Authority Process

The authority (electoral official) in the polling station is responsible for authenticating voters on the channel *auth* and assigning the pre-printed multi-ballots (three unique serial numbers from the set $\mathcal{S}$) to the voters with the *alloc* events. Moreover, the serial number allocations are performed non-deterministically and it is ensured that the serial number allocated is never assigned to another mini-ballot. Because the last line of a mini-ballot is allocated for serial numbers, each is placed on the nth column, where n is the number of candidates on the board (note that the row numbers range between 0 and n, thus ensuring that the serial numbers are always allocated at the bottom of the ballot forms). Additionally, the allocation occurs three times for each multi-ballot form, and once three are allocated, the remaining serial numbers are carried forward to the next voter. When the election is closed through the *closeElection* event, no more ballots are allocated for any voter. Here we do not present the most efficient authority model used in the analysis for simplicity. Section 5.4 provides details of the need to define efficient model.

$$\begin{aligned}
&Authority \mathrel{\widehat{=}} openElection \rightarrow Authority_1(\mathcal{S}) \\
&Authority_1(serials) \mathrel{\widehat{=}} \\
&\qquad auth?v \rightarrow \prod_{s \in serials} \left(\begin{array}{l} alloc.v.s.(n,0) \rightarrow \\ Authority_2(v,(n,0),serials \setminus \{s\}) \end{array} \right) \\
&Authority_2(v, coord, \emptyset) \quad \mathrel{\widehat{=}} closeElection \rightarrow STOP \\
&Authority_2(v,(n,2),serials) \mathrel{\widehat{=}} Authority_1(setSerials) \\
&Authority_2(v,(n,i),serials) \mathrel{\widehat{=}} \\
&\qquad \prod_{s \in serials} \left(alloc.v.s.(n,i+1) \rightarrow Authority_2(v,(n,i+1),serials \setminus \{s\}) \right)
\end{aligned}$$

The Bulletin Board Process

The process *B_Board* operates as a bulletin board where the cast mini-ballots are published. In detail, the votes are collected by this process as the voters cast their mini-ballots and a record is kept of the serial numbers and the bubbles that are filled in on that particular mini-ballot with the set *bag*. Once a serial number is allocated to a coordinate (i, j) for any voter, the process traces the jth column, whenever a *place* event happens on that column, because the event $place.v.(i, j)$ models a vote for the ith candidate on the candidate list. Subsequently, the value i is then stored in the set *bag*. Additionally, the process is also ready to

give any of the mini-ballots as the voter's receipt, with the serial number and bubbles filled in, if requested by them. After the election, the cast mini-ballots are published with the *pub* event. The whole process *B_Board* is modelled as the parallel composition of individual mini-ballot processes as follows.

$$
\begin{aligned}
&Board(s) \mathrel{\widehat{=}} alloc?v.s?(i,j) \rightarrow Board_1(\emptyset, s, (i,j)) \\
&Board_1(bag, s, (i,j)) \mathrel{\widehat{=}} \\
&\quad \Box_{(m,n)\in \mathsf{Col}(j)} \Big(place.v.(m,n) \rightarrow Board_1(bag \cup \{m\}, s, (i,j)) \Big) \\
&\quad \Box\ receipt?v.s.bag \rightarrow Board_2(s, bag) \\
&\quad \Box\ Board_2(s, bag) \\
&Board_2(s, bag) \mathrel{\widehat{=}} closeElection \rightarrow pub.s.bag \rightarrow bagempty \rightarrow STOP
\end{aligned}
$$

$$
B\_Board \mathrel{\widehat{=}} openElection \rightarrow \|_{s\in\mathcal{S}} Board(s)
$$

Counter Process

The last process is *Counters*, which works as an electoral official counting the votes published on the bulletin board, keeping a record of *place* events for each candidate by following each row of ballot forms. That is, when there exists a mark on the ith row of a mini-ballot, it is counted as a vote for the candidate c_{i+1}. Moreover, when no more *place* events are happening, the number of total votes for each candidate is published on the channel *total*.

$$
\begin{aligned}
&Counter(c_x, r) \mathrel{\widehat{=}} \\
&\quad \Box_{\substack{v\in\mathcal{V} \\ (i,j)\in \mathsf{Row}(x-1)}} \Big(place.v.(i,j) \rightarrow Counter(c_x, r+1) \Big) \\
&\quad \Box\ bagempty \rightarrow total.c_x.r \rightarrow STOP
\end{aligned}
$$

$$
Counters \mathrel{\widehat{=}} \|_{c\in\mathcal{C}} Counter(c, 0)
$$

System Process

The following parallel composition of the processes defined previously models the behaviour of the ThreeBallot voting system.

$$
System_{3B} \mathrel{\widehat{=}} Voters \parallel Authority \parallel Booth \parallel B\_Board \parallel Counters
$$

In the overall flow of the system process, as all voters are run in parallel and synchronise on *openElection* and *closeElection* events with the election authority pairwise. Thus, each voter performs an *openElection* event to begin her voting process. Each voter must also perform a *closeElection* event after casting their

individual vote and leaving the polling station. Additionally, the WBB publishes cast votes when all voters have finished vote casting and subsequently the election closed. Finally, the counter process announces total vote for each candidate when there is no ballot left to count in the ballot box.

A number of sanity checks are deployed in order to gain more confidence in the correct behaviour of the CSP model of the ThreeBallot voting system and the details can be found in Appendix A.2. In the following subsection, the passive intruder model is defined for the analysis of the ThreeBallot CSP model.

5.2.3 The Passive Attacker

Although the passive intruder model defined in the previous chapter is a generic framework, a few modifications need to be made for individual voting system models, because the channels and data-types can be modelled differently for each. Regarding this, the intruder model in this analysis is similar to the one defined before, however, the vulnerable data on the channels needs to be hidden according to the needs of the ThreeBallot voting system model. More specifically, the intruder is able to see all the public channels, but not the private ones: *alloc* and *place*. Hence, these private channels need to be hidden from the intruder, forming the abstracted system process, $System_{3B}^{\mathsf{abs}}$ below:

$$System_{3B}^{\mathsf{abs}} \mathrel{\widehat{=}} System_{3B} \setminus \{|\, alloc, place \,|\}$$

Note that the intruder in this model analysis is able to see all *receipt* events, i.e., he can see all the receipts taken by the voters in an election (this is a strong assumption; however, it is safe in the sense that if the system is secure under this assumption, it will also be secure for an adversary who sees only some receipts.). In the following section the automated analysis of this voting system model is presented.

5.3 Automated Anonymity Verification

The verification of this voting system is checked against the anonymity specification defined in Section 4.3 Definition 2. To recall, when the two channels $c.x$ and $d.x$ are swapped over for all values of x, if the resulting process is indistinguishable from the original one, P, from an observer's point of view, then it provides anonymity.

It is over channel *choose* that the voter determines a choice of candidate and consequently, the channels that need to be swapped over are: $choose.v_1.c_x$ and

$choose.v_2.c_x$ for $c_x \in \mathcal{C}$. Consequently, the anonymity specification for the ThreeBallot CSP model $System_{3B}^{\mathsf{abs}}$ over the set $A = \{|\, choose \,|\}$ can be described as:

$$Spec_A(System_{3B}^{\mathsf{abs}}) \mathrel{\widehat{=}} System_{3B}^{\mathsf{abs}}[\![choose.v_1.c_x, choose.v_2.c_x / choose.v_2.c_x, choose.v_1.c_x]\!]$$

Following this, the anonymity requirement of this voting system model is verified with the following trace equivalence.

$$Spec_A(System_{3B}^{\mathsf{abs}}) \equiv_{\mathrm{T}} System_{3B}^{\mathsf{abs}}$$

5.3.1 Results with no SBA

In this subsection, some of the results produced by the automated anonymity analysis of ThreeBallot without the SBA assumption are given.

Not unexpectedly, the previous trace equivalence does not hold for the ThreeBallot voting system model, because there are situations in which a reconstruction attack is possible. That is, a coercer who has seen the receipts for v_1 and v_2 can deduce that they voted respectively for c_1 and c_2, because there is no way of constructing a complete set of valid multi-ballots in which the voters voted other way around. The analysis shows that for the ThreeBallot voting system, whether the election run provides anonymity entirely depends on how the voters fill their multi-ballots, and also on which mini-ballots they choose as receipts. The following counter-examples produced by FDR in different voting scenarios give useful intuition about the situations in which anonymity is violated.

Examples of Privacy Violations of ThreeBallot

The FDR model checker returns several counter-examples which violate anonymity and the following illustrated ones are the election runs from the observer's point of view. Note that the receipts shown in the following figures are taken in sequence by the voters, i.e., the first receipt is taken by the voter v_1, and the second by v_2, and so forth.

Example 1. *The first counter-example is taken from a protocol run with two voters, v_1 and v_2, and two candidates, c_1 and c_2. It shows that in a voting scenario where the public information is displayed as in Figure 5.3 and v_1 gets the mini-ballot s_2 as her receipt, and v_2 chooses s_3, the observer is able to reconstruct the multi-ballots, thus violating their anonymity. This is because there is only one possible way of forming valid multi-ballots with the public information shown in Figure 5.4 and therefore, the observer is able to deduce who voted for whom in*

this ThreeBallot election run. Although, the voters may have used different mini-ballots to cast their votes, such as instead of s_0, v_1 may have used s_5, this does not affect the candidate that v_1 has voted for.

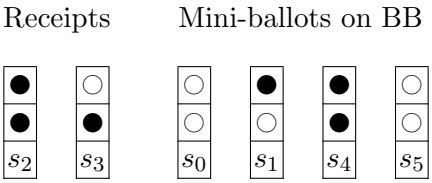

Figure 5.3: Voting scenario 1

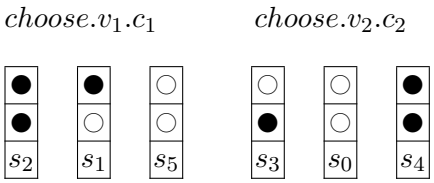

Figure 5.4: Reconstruction attack 1

Example 2. *Figure 5.5 involves a counter-example in a voting scenario with two voters and three candidates. The counter-example trace shows that when the voters v_1 and v_2 vote for c_3 and c_1, respectively, taking the mini-ballots with the serial numbers s_0 and s_2 as their receipts, the observer can break their anonymity, because the only way of constructing valid multi-ballots is illustrated in Figure 5.6. In other words, there is no possible combination of these mini-ballots shown on the BB such that the voter v_1 can vote for the candidate c_1 with the receipt $s.0$, and v_2 can vote for c_3 with the receipt $s.2$.*

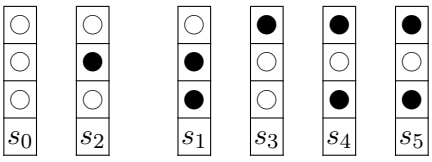

Figure 5.5: Voting scenario 2

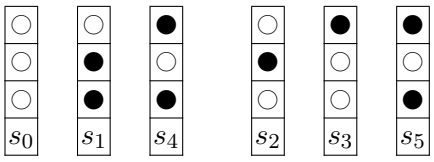

Figure 5.6: Reconstruction attack 2

Example 3. *The last counter-example involves an election with three voters and two candidates, as depicted in Figure 5.7. When the voter v_1 votes for c_1, v_2 votes for c_2, and v_3 votes for c_1, with the receipts s_1, s_2 and s_0, respectively, the intruder is sure about the voter v_1 not voting for c_2, but for c_1. Figure 5.8 shows the only possible way of reconstructing the valid multi-ballots by comparing the public mini-ballots and the receipts taken by the voters.*

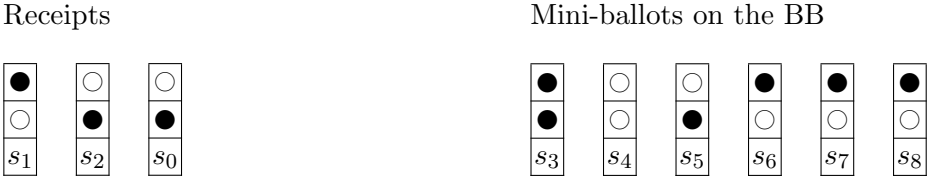

Figure 5.7: Voting scenario 3

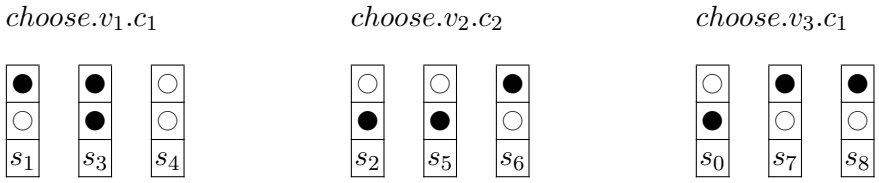

Figure 5.8: Reconstruction attack 3

Previously Suggested Modifications to ThreeBallot

So far it has been shown with the previous counter-examples that the ThreeBallot voting system does not provide anonymity. The following contains analysis of some of the proposed modifications for ThreeBallot in the literature that are claimed to be robust against such privacy attacks.

Taking Receipt Before Expressing Preference In this proposed modification by [dMPQ07], the voter chooses her receipt just before expressing her preference. That is, she fills one bubble for each candidate, and decides which mini-ballot to copy as her receipt. Then, she fills out one extra bubble for the chosen candidate, but not on the mini-ballot that was chosen as her receipt. It is contended in [KTV11] that this new version of ThreeBallot provides a better level of privacy than the primary scheme (probabilistic privacy). However, the automated analysis of this modified protocol reveals that it is also vulnerable to reconstruction attacks without the SBA being met. In particular, although it might appear that a voter's preference is not linked with her receipt as she has chosen it beforehand, the vote expressed through all three mini-ballots on the multi-ballot includes the one taken as a receipt.

Example 4. *By way of an example regarding this weakness, Figure 5.9 illustrates a voting scenario with this modified version of the ThreeBallot model involving two voters and two candidates. The public data on the BB and the existence of receipts mandate that the receipt s_4 can only be combined with a fully-filled mini-ballot and an empty one. Hence, the voter v_1 holding the receipt s_4 has definitely voted for candidate c_1, thus violating v_1's anonymity. Figure 5.10 shows a possible composition of these public mini-ballots.*

Figure 5.9: Voting scenario 4

Figure 5.10: Reconstruction attack 4

FourBallot The FourBallot scheme was proposed by Kutylowski *et al.* [KZ10], who claimed that it is immune to Strauss' like attacks [Str06b, Str06a] (reconstruction and pattern-matching (Italian) attacks[†]) under the SBA. In this scheme a multi-ballot consists of four mini-ballots and a voter fills in exactly three bubbles for the chosen candidate and two for the others (see Figure 5.11). The tallying is also similar to the original system, with the only difference being that for the FourBallot scheme two times the number of voters should be subtracted from the total tally for each candidate. The following counter-example illustrates that the

Alice	●	Alice	●	Alice	●	Alice	○
Bob	○	Bob	●	Bob	●	Bob	○
Chris	●	Chris	○	Chris	○	Chris	●
David	○	David	○	David	●	David	●
	56248		04578		31489		58201

Figure 5.11: A FourBallot multi-ballot form, filled in as a vote for Alice

proposed scheme without the SBA is vulnerable to reconstruction attacks for any number of candidates as it can be generalised.

Example 5. *In an election run with two voters and two candidates where the public data on the BB is as in Figure 5.12, a reconstruction attack can be illustrated as in Figure 5.13. With the public information on the BB and the receipts taken, the intruder violates the voters' anonymity, because the receipts s_0 and s_3 can only be combined with a fully-filled (such as s_1, s_2) mini-ballot and an empty one, like s_4, hence, deducing that v_1 has voted for c_1 and v_2 for c_2.*

In an election where the only voter who votes for c_1 is v_1, and all the others vote for the candidate c_2, the observer could deduce some information about the vote for c_1, if he keeps track of all the allocated mini-ballots to reconstruct valid multi-ballots. This is simply because of the fact that if the only voter who votes for *Alice* is v_1 with the multi-ballot filled as in Figure 5.14, there is no anonymity unless there is another voter, v_2 who votes for *Bob* with the mini-ballots filled as

[†]The coercer asks the voter to fill the mini-ballots in an uncommon way (e.g., voting for the candidate who has the least possible chance to win the election), so he can then check whether the unusual mini-ballots appear on the bulletin board.

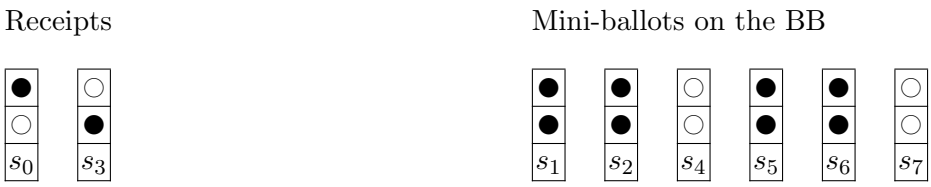

Figure 5.12: Voting scenario 5: FourBallot

Figure 5.13: Reconstruction attack 5

in Figure 5.15. Therefore, no matter how many other voters vote for *Bob*, when there is no such multi-ballot filled out as in Figure 5.15, there is no anonymity for the voter v_1.

Alice	●	Alice	●	Alice	○
Bob	○	Bob	●	Bob	○
	56248		04578		31489

Figure 5.14: A ThreeBallot multi-ballot voted for Alice

Alice	●	Alice	○	Alice	○
Bob	○	Bob	●	Bob	●
	78452		90732		13797

Figure 5.15: A ThreeBallot multi-ballot voted for Bob

Therefore, there is no absolute anonymity unless another voter exists who holds the same pattern on their receipt, but has voted for another candidate. Nevertheless, no one can guarantee that such a ballot appears on the bulletin board. As a result, although the proposed alternative schemes may provide a better level of probabilistic privacy, they are, however, not powerful enough to guarantee anonymity.

5.3.2 The Short Ballot Assumption

This section investigates the short ballot assumption by analysing the ThreeBallot voting system under two of the three possible interpretations of the SBA that were given earlier: Assumptions 2 and 3 (Recall that Assumption 1 seems to

be implausible unless there are only very few candidates.). In this section, the analysis of ThreeBallot under the SBA versions is made manually except under the SBA-mini because the SBA versions require too many voters to be involved in an election run, which cannot be checked using model checking tools. First, the ThreeBallot model under the SBA-mini is analysed mechanically using FDR, and subsequently, analysis of the model under the SBA-mini-n is provided using a hand-proof treatment. Finally, a better formulation the SBA-pro is presented.

Analysis under the SBA-mini

Suppose Assumption 2 is adopted, under which all possible mini-ballots are assumed to appear on the bulletin board at least once at the end of the election. The following counter-example shows that under such an assumption, the ThreeBallot voting system model does not provide anonymity.

Example 6. *Figure 5.16 is the voting scenario with three voters and three candidates, holding the Assumption 2. That is, all possible mini-ballots appear on the BB at least once. Under such assumption, however, the receipt s_0 has two possible completion methods: it could be combined with s_2 and s_4 or s_8 (as depicted in Figure 5.17), or with s_5 and s_7, but in either case it represents a vote for the third candidate.*

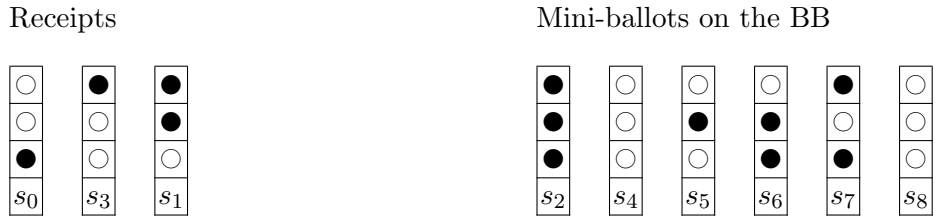

Figure 5.16: Voting scenario 6: All possible mini-ballots appear on the bulletin board

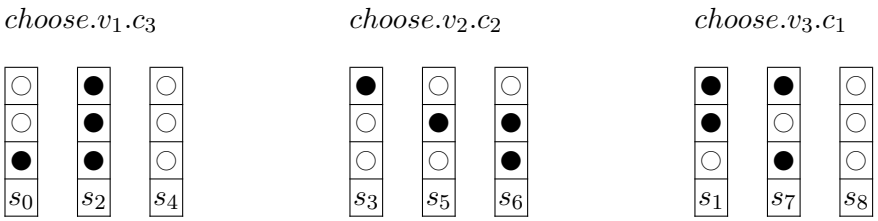

Figure 5.17: Reconstruction attack 6

Analysis under the SBA-mini-n

We now analyse the system under the Assumption 3, which ensures that every possible mini-ballot will appear on the bulletin board at least n times for some suitable value of n in the following lemma:

Lemma 3. *The Assumption 3 is insufficient for ThreeBallot to provide anonymity regardless of how many times every possible mini-ballot appears on the BB.*

Proof. The proof relies on a special construction of mini-ballots published on the bulletin board using the following two observations.

Observation 1. *A fully-filled mini-ballot can be combined only with an empty mini-ballot and a singleton (a mini-ballot with only one bubble marked) as in Figure 5.18.*

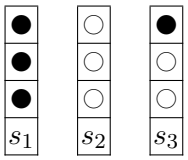

Figure 5.18: A completion of a multi-ballot with a fully-filled s_1, an empty s_2, and a singleton mini-ballot s_3

Observation 2. *Any possible mini-ballot m that is not empty, fully-filled or a singleton, can be turned into a completed multi-ballot that does not contain a fully-filled mini-ballot or a singleton such as the mini-ballot s_1 in Figure 5.19. This can be done by combining it with another mini-ballot that is the complement of m, but with one extra bubble and an empty mini-ballot, as in the following.*

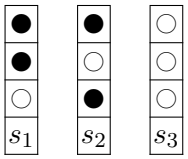

Figure 5.19: A completion of a mini-ballot s_1 that is not empty, fully-filled or a singleton

A bulletin board that displays at least n copies of every possible mini-ballot can be reached in the following way using these two observations above.

Each possible mini-ballot that is not empty, fully-filled or a singleton, like s_1 in Figure 5.19, can be turned into a multi-ballot as described in Observation 2, and added to the BB. This gives at least n copies of everything except singletons and

fully-filled mini-ballots. Now each possible singleton should be combined with a fully-filled mini-ballot and an empty mini-ballot as in Observation 1. Thus, n copies of each such multi-ballot are added to the BB, meaning that every possible mini-ballot now appears at least n times on the BB.

With such construction of the bulletin board, however, any voter taking a singleton as a receipt will have no anonymity, because the number of fully-filled mini-ballots is the same as the number of singletons. That is, since each fully-filled ballot must be combined with a singleton and a blank one from Observation 1, it follows that the voter's receipt must have been part of such a multi-ballot. However, in that case the mini-ballot reveals the candidate that the voter selected and hence, no value of n is sufficient to guarantee anonymity in ThreeBallot. □

SBA-pro: A better formulation

The possible interpretations of the SBA previously given are either not sufficient or too unrealistic to provide anonymity and hence a much more plausible definition, one that is demonstrably strong enough for ThreeBallot, is given next.

Assumption 4 (SBA-pro)**.** *Let M be the set of all mini-ballots cast during the election, where $R \subset M$ is the set of all receipts that are known to the adversary, and* vote *is a partial function, such that* $\mathsf{vote}(m_1, m_2, m_3) = c$ *whenever the three mini-ballots m_1, m_2 and m_3 together form a valid multi-ballot that represents a vote for c. Additionally, for any two mini-ballots m_1 and m_2, $m_1 \sim m_2$, if and only if, they contain the same sequence of vote marks (i.e., $m_1 = m_2$, but the serial numbers are different).*

For every $r \in R$ and every candidate c, there is a vote cast consisting of three (unordered) mini-ballots, m_1, m_2, m_3, such that:

1. $r \sim m_1$;
2. $\mathsf{vote}(m_1, m_2, m_3) = c$;
3. $m_2, m_3 \in M \setminus R$.

Informally, under this interpretation, for every receipt known to the adversary there is an equivalent one used in a multi-ballot for each of the candidates in the election.

Theorem 1. *The assumption 4 is strong enough to prevent reconstruction attacks in ThreeBallot.*

Proof. The key to the proof is the observation that if $m \sim m'$ then $\mathsf{vote}(m, m_2, m_3) = \mathsf{vote}(m', m_2, m_3)$. This is clear from the fact that m and m'

differ only regarding their serial numbers, which are not relevant for determining which candidate received the vote cast by a multi-ballot.

Suppose that $r \in R$, and the adversary wishes to determine which candidate received the vote cast that included r, this not possible because r could be a vote for any of them. By way of explanation, if r did in fact occur in a multi-ballot along with m_1 and m_2, as a vote for c, for any other candidate c', there was a multi-ballot cast containing m_3, m_4, m_5, such that $\mathsf{vote}(m_3, m_4, m_5) = c'$ and $r \sim m_3$, and with m_4 and m_5 not known to the adversary. Consequently, this means that the adversary cannot distinguish the following two possibilities:

1. a ballot of (r, m_1, m_2) for c, and a ballot of (m_3, m_4, m_5) for c';
2. a ballot of (m_3, m_1, m_2) for c, and a ballot of (r, m_4, m_5) for c'.

In each case, the set of mini-ballots used by this partial reconstruction is the same, so it cannot affect further reconstruction of the remaining mini-ballots. In one case, r was used to vote for c, and in another, for c' and since c' was arbitrarily chosen, it is concluded that r could equally have been used to vote for any candidate. □

To see the improved plausibility of this interpretation, suppose the adversary has knowledge of r receipts in an election run with n candidates. The SBA-pro requires at least $n \cdot r$ multi-ballots of the right type have been cast to protect anonymity. By contrast, the SBA-multi requires at least $n \cdot 3^n$ other appropriate multi-ballots and as long as r is small, the SBA-pro is much less demanding compared with the SBA-multi. For instance, in an election with 10 candidates, the SBA-multi needs at least 590,490 multi-ballots and unless the adversary has seen somewhere in the order of 59,000 receipts, the SBA-pro is much more likely to be satisfied.

This efficiency argument is not absolute and to formalise it would require a full voter model; that is, it would need a probability distribution for the multi-ballots cast in an election. Producing such a model is probably unrealistic, since it would be affected by the prevailing political landscape at the time of the election and it is in any case outside the scope of this thesis.

5.3.3 Verified Privacy Cases

Apart from the SBA, several slight modifications for ThreeBallot have been proposed to help the system provide absolute anonymity and by using FDR these modified systems can automatically be verified against reconstruction attacks.

The first proposed modification allows the voters to exchange their receipts, and the second mandates that voters must fill in at least one bubble in every column.

Floating or Exchanging Receipts

Rivest [Riv06] has suggested a possible improvement to the original ThreeBallot scheme, with the idea of exchanging receipts in the polling station— the idea is originally known as the Farnel protocol and it considers exchanging ballot forms rather than receipts (see [ACW+06] for a full description of the idea in English). Here, each voter puts her receipt in a box, and takes someone else's and indeed, this idea can be used in any paper-based election system. If voters are allowed to take a random receipt from the box in the polling station, then this eliminates reconstruction attacks as well as pattern-matching attacks, because the adversary does not have any knowledge of any part of the voter's ballot.

This is modelled in CSP, such that all cast mini-ballots are collected in the polling station outside the booth and once the voters have finished casting votes, each collects a mini-ballot from the box as their receipt before the election is closed, i.e., the voters wait for each other casting their votes outside the booth and pick a mini-ballot from a collection of them. Although the adversary may be able to reconstruct valid multi-ballots, he cannot link them to the voters, as any of the voters could have voted for any candidate, because they can get any mini-ballot as their receipt providing that each candidate has received at least one vote. Hence, the automated analysis here using FDR has confirmed prior research [Riv06] that modified schemes, where voters are allowed to take any cast mini-ballots as their receipts or exchange them, provide guaranteed anonymity.

No Single Mini-ballot Left Blank

In this version of ThreeBallot, the condition is that voters must fill out at least one bubble on each mini-ballot. Under this modified model, it has been automatically verified that this condition is sufficient to guarantee anonymity with a two candidate election run in which case there are only two ways of filling a mini-ballot, and thus only two different receipts that can be taken by voters.

In the modelling of such voter behaviour, the voter process is forced to place a mark on all three columns with an extra line. Hence, as in the original process, the voter places two marks for the chosen candidate, say on the coordinates $(i, 1)$ and $(i, 2)$, with the extra line in the process, a *place* event, forced to happen on the 3rd column and any of the rows apart from the ith, such as on the coordinate $(i + 1, 3)$. Following this, the voter fills in the multi-ballot form placing a mark for all the other candidates as in the original voter process.

However, in an election where there are more than two candidates, although

intuitively the system provides better probabilistic anonymity than the original, it cannot guarantee voter anonymity. The following illustrates a counter-example produced by FDR:

Example 7. *Figure 5.20 involves a counter-example in a voting scenario, where no mini-ballot is left blank, with two voters and three candidates. The counter-example demonstrates that when the voters v_1 and v_2 vote for c_3 and c_1, respectively, taking the mini-ballots with the serial numbers s_3 and s_1 as their receipts, the observer can break their anonymity, because the only way of constructing valid multi-ballots is as illustrated in Figure 5.21. In more detail, the mini ballot s_1 cannot be combined with that of s_5 and hence, according to the protocol, the only combination for the receipt s_3 to form a valid multi-ballot is s_3, s_0 and s_5. Therefore, the intruder can deduce how each voter has voted.*

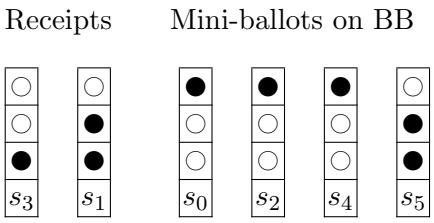

Figure 5.20: Voting scenario 7: no mini-ballot left blank

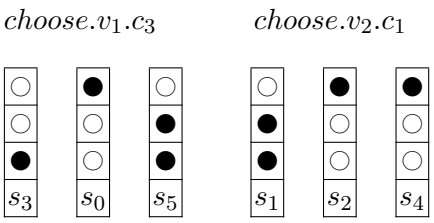

Figure 5.21: Reconstruction attack 7

5.4 Challenges Faced in the Modelling and Analysis

This section presents the challenges faced during the modelling of ThreeBallot and anonymity analysis of it. The ultimate challenge was to model the voting system in a way that is efficient for automated verification because the number of states of a CSP system can increase very quickly—especially for voting systems as they are complex and consist of a number of parallel components. In order to overcome this difficulty, the two methods, control processes and compression functions, were used.

The control processes helped us to reduce the states of the individual processes when the control process were applied. For instance, the voter process was restricted in a way that a voter could mark her ballot in a left to right fashion when filling bubbles for the chosen candidate and top to bottom when filling bubbles for the other candidates. Similarly, the voters were made to authenticate themselves in a descending order, i.e., first the voter v_n, then v_{n-1} and so on. Moreover, there were some limitations for the authority process too. For instance, the authority process was forced to allocate serial numbers for each mini-ballot form in a left

	Original		No mini-b empty		All mini-b appear	
	States	Time	States	Time	States	Time
2v 2c	$239,905$	$7.8s$	$56,841$	$5.3s$	$240,055$	$7.0s$
2v 3c	$4,139,347$	$1m41.8s$	$1,435,926$	$38.3s$	$4,165,428$	$1m40.1s$
3v 2c	–	–	$67,409,391$	$22m49.3s$	–	–
3v 3c	–	–	–	–	–	–

Table 5.5: The FDR verification times for the ThreeBallot versions. As the required state space grows quickly with the number of voters and candidates, it was not possible to produce results in some cases as FDR cannot handle with such huge states. Those are denoted as "–" in the table.

to right style with respect to the coordinates allocated for serial numbers, e.g., for 2 candidate race, first serial is allocated to $(2,0)$, then the second to $(2,1)$ and the last one to $(2,2)$. This reduces state space dramatically. Likewise, the occurrence of *pub* and *total* events could be performed in a descending order with suitable control processes.

The other method is the compression functions. We found two compression functions; sbisim() and diamond() useful to reduce the states to check the following refinement for automated anonymity analysis of ThreeBallot.

$$\mathsf{sbdia}(Spec_A(System^{\mathsf{abs}}_{3B})) \equiv_{\mathrm{T}} System^{\mathsf{abs}}_{3B}$$

where $\mathsf{sbdia}(P) = \mathsf{sbisim}(\mathsf{diamond}(P))$.

sbisim and diamond are two compression functions that do not modify the semantics of the processes when they are applied to them. The former one is the strong bisimulation, which is an equivalence over labelled transition systems (LTS) (a set of nodes and a relation for each event in some set). The latter, diamond, is called diamond elimination. It is a function that removes all τ actions from an LTS, produces an LTS with minimal acceptance and divergence information and never increases the number of nodes (see [Ros97] for further reading for a variety of compression functions that may be used in different CSP models).

Moreover, to give an idea to the reader about the verification times, see Table 5.5 (the results were produced using the efficient models). In the table, there are verification times for three different versions of ThreeBallot: the original ThreeBallot, a version of ThreeBallot in which voter fills in all mini-ballots of her multi-ballot (leaving no empty mini-ballots), and another version in which all possible mini-ballots appear on the bulletin board.

5.5 Summary

In this chapter, it has been demonstrated that the ThreeBallot voting system is vulnerable to privacy-related attacks, especially reconstruction attacks, even under some plausible interpretations of the short ballot assumption.

In the analysis, an abstracted CSP model of ThreeBallot has been used, which is defined as the parallel composition of agents in the system, and the passive intruder model defined in the previous chapter as pertaining to a person who can see all the public channels, including what each voter takes as a receipt.

Throughout a number of examples for different voting scenarios it was demonstrated that ThreeBallot does not provide anonymity under various formulations of the short ballot assumption. However, given a reasonable and plausible interpretation of this assumption, ThreeBallot is in fact protected from reconstruction attacks. Moreover, previously suggested modifications to ThreeBallot, such as, taking receipt before expressing her preference and FourBallot was shown that they are not adequate to prevent from reconstruction attacks. Finally, two different versions of ThreeBallot were analysed automatically using FDR, namely, exchanging receipts and no mini-ballot left blank.

Because of the state space problem, as mentioned in Chapter 2, a limited number of agents were considered. In most cases, the restriction did not affect the analysis of the systems and assumptions; however, as under the short ballot assumption a large number of mini-ballots are required, it was not possible to demonstrate automatic verification in such cases, but hand proofs were supplied where appropriate.

In the next chapter, the CSP approach to automated analysis of voting systems will be extended and applied to a cryptographic voting system, Prêt à Voter, thereby presenting the first automated formal analysis of this voting system.

Chapter 6

Modelling and Analysis of a Cryptographic Voting System

This chapter* extends the framework introduced in Chapter 4 and used in the non-cryptographic voting system analysis in Chapters 4 and 5 to cope with cryptographic voting systems and aims to demonstrate how this framework can be applied to the paper-based voter verifiable cryptographic voting system Prêt à Voter. This is the first automated analysis of this promising voting system using a reasonably abstracted model of it. Similar to the previous analyses, first, the behaviour of the honest participants is modelled and then the capabilities and limitations of the passive attacker as given in Chapter 4 are once again determined. Finally, automated formal analysis of this model of Prêt à Voter is performed against the specification defined in Chapter 4.

The chapter is structured as follows. Firstly, in Section 6.1, the Prêt à Voter voting system is introduced, covering the voting ceremony, system properties and components. Secondly, in Section 6.2 the modelling and following that in Section 6.3 automated analysis of Prêt à Voter as well as the CSP approach to cryptographic voting systems are presented. Section 6.4 investigates the corrupt agent scenarios in this voting protocol. Finally, Section 6.5 concludes the chapter with a discussion and a summary of the findings.

6.1 The Prêt à Voter Voting System

Prêt à Voter is a paper-based, voter-verifiable cryptographic e-voting system, introduced by Ryan [Rya04, Rya05] as an improvement on Chaum's scheme [Cha04]. Since then, it has been further improved and enhanced in

*This chapter is mainly based on the published work [MHS12].

many different ways [CRS05, RP05, RS06, Rya06, Hea07, Rya08, RBH+09, RP10, XCH+10, BCH+12b]. In the modelling and analysis in CSP for this work the focus is on the re-encryption mixes version of Prêt à Voter, as proposed by Ryan and Schneider [RS06]. Hereby, this version of Prêt à Voter will be abbreviated as Prêt à Voter.

Figure 6.1 illustrates a simple Prêt à Voter ballot form cast for Alice in a single-winner electoral method as in FPTP—although, Prêt à Voter can be used for preferential voting systems, these are not covered here. In the left-hand column of the ballot form is printed a random permutation of the candidate names and an encrypted onion called *registrar onion*. It embeds the candidate order, so that once it is scanned, the voting machine can decrypt the value and print the candidate list on the ballot form on demand. Moreover, in the right-hand column are the boxes in which the voter can mark her choice as well as a unique serial number at the top and the cryptographic value at the bottom of this column. It is called *teller onion* and embeds the candidate ordering on the left-hand side, and is encrypted under the tellers' public-key. This public-key is a threshold key for an appropriate homomorphic encryption algorithm, such as ElGamal [ElG84] or Paillier [Pai99]; the precise algorithm is not important for the purposes of this thesis. Moreover, there is a perforation line between the two halves to facilitate separating them. The following subsections present the voting ceremony for Prêt à Voter in general and the re-encryption version in detail.

	9X3ht
Bob	
Alice	X
Chris	
i5yTf	2iP6d2

Figure 6.1: Prêt à Voter ballot form

Voting with Prêt à Voter

Voting with Prêt à Voter is (by design) quite similar to voting in the conventional voting system. That is, an eligible voter goes to a polling station, authenticates herself to the election official and takes a random ballot form in an envelope. These ballot forms have been produced by the election authorities before the election day and are kept sealed (Although there are forms of Prêt à Voter that use on-demand printing, they are not considered here.). Once the voter goes into the booth, she then marks her choice with a cross on the right-hand column, tears the ballot form down the perforation line, and shreds the candidate list. Finally, she scans her ballot and takes the right-hand column as her receipt. Later, all the scanned right-hand sides will be published on a web bulletin board (WBB), and

she will be able to use her receipt to check that her vote has not been changed or deleted. As the candidate name order in the left-hand column is random, the right-hand column reveals nothing about her vote.

Ballot Generation: Prêt à Voter introduces a distributed ballot generation by a number of election authorities or clerks using the ElGamal encryption system and re-encryption mixes in order to eliminate single point failures.

For the distributed generation of ballots, several clerks are used jointly, with each of them contributing to the construction of the cryptographic seeds, which form the onions at the bottom of ballot forms. The registrar onion (on the bottom left) can be decrypted by the voting machine in the polling station, however, the teller onion (on the bottom right) can only be decrypted by a threshold of tellers, because it is encrypted under a distributed secret-key among a number of tellers. The distributed ballot generation is performed as follows.

Suppose that the ElGamal public key parameters (α, γ, p, q) are chosen and made public according to the ElGamal protocol, where p and q are two large primes, such that $p = 2q + 1$, and α and γ are generators of the cyclic group $\mathbb{Z}_p$. Additionally, a set of tellers generate the secret key $x_T \in \mathbb{Z}_q^*$ in a threshold fashion and publish the public key $(p, \alpha, \alpha^{x_T})$ in order to generate the teller onion. Similarly, another secret key $x_R \in \mathbb{Z}_q^*$ is chosen by the voting machine, and the public key $(p, \alpha, \alpha^{x_R})$ is revealed. For convenience, the registrar's public key is denoted as $\beta_R = \alpha^{x_R}$ and the tellers' public key $\beta_T = \alpha^{x_T}$ in the following.

Initially, clerk C_0 (one of the l clerks) generates a batch of seeds r_i^0 randomly from a binomial distribution and a batch of pairs of onions by encrypting each r_i^0 with the public keys of registrars and tellers in the form $\gamma^{-r_i^0}$. Thus, the pair $(\mathsf{E}_{pk_R}(\gamma^{-r_i^0}), \mathsf{E}_{pk_T}(\gamma^{-r_i^0}))$ is the encryption of a randomly chosen r_i^0 under the corresponding public keys. In the case of ElGamal, the pair is expressed as:

$(\alpha^{x_i{}^0}, \beta_R^{x_i{}^0}.\gamma^{-r_i^0}), (\alpha^{y_i{}^0}, \beta_T^{y_i{}^0}.\gamma^{-r_i^0})$, where x_i^0, y_i^0 are chosen from $\mathcal{F}_p^*$

The next clerks will perform a combined re-encryption with freshly injected entropy seed values, such that for each pair of the onion the same entropy is added so as to ensure that these values still continue to match. Assume $\overline{x}$, $\overline{y}$ are fresh random values taken from $\mathcal{F}_p^*$ and $\overline{r}$ are random values independently generated by the clerk C_j with a binomial distribution mean 0 and standard deviation σ. Then the transformation for each onion pair in the batch is as follows.

$$
\begin{array}{rl}
(\alpha^{{x_i}^{j-1}}, \beta_R^{{x_i}^{j-1}}.\gamma^{-r_i^{j-1}}) & (\alpha^{{y_i}^{j-1}}, \beta_T^{{y_i}^{j-1}}.\gamma^{-r_i^{j-1}}) \\
\downarrow & \downarrow \\
(\alpha^{{x_i}^{j-1}}.\alpha^{\overline{x}^j}, \beta_R^{{x_i}^{j-1}}.\beta_R^{\overline{x}_i{}^j}.\gamma^{-r_i^{j-1}}.\gamma^{-\overline{r}_i^j}) & (\alpha^{{y_i}^{j-1}}.\alpha^{\overline{y}^j}, \beta_R^{{y_i}^{j-1}}.\beta_R^{\overline{y}_i{}^j}.\gamma^{-r_i^{j-1}}.\gamma^{-\overline{r}_i^j}) \\
\downarrow & \downarrow \\
(\alpha^{{x_i}^{j-1}+\overline{x}^j}, \beta_R^{{x_i}^{j-1}+\overline{x}^j}.\gamma^{-(r_i^{j-1}+\overline{r}_i^j)}) & (\alpha^{{y_i}^{j-1}+\overline{y}^j}, \beta_R^{{y_i}^{j-1}+\overline{y}^j}.\gamma^{-(r_i^{j-1}+\overline{r}_i^j)}) \\
\downarrow & \downarrow \\
(\alpha^{{x_i}^j}, \beta_R^{{x_i}^j}.\gamma^{-r_i^j}) & (\alpha^{{y_i}^j}, \beta_T^{{y_i}^j}.\gamma^{-r_i^j})
\end{array}
$$

, where

$$
\begin{aligned}
x_i^j &= x_i^{j-1} + \overline{x}_i^j (\mathrm{mod}\, q) \\
y_i^j &= y_i^{j-1} + \overline{y}_i^j (\mathrm{mod}\, q) \\
r_i^j &= r_i^{j-1} + \overline{r}_i^j (\mathrm{mod}\, q)
\end{aligned}
$$

After having transformed each onion pair, clerk C_j performs a secret shuffle and forwards the result to the next clerk C_{j+1} and so forth. Thus, the final output after $l-1$ mixes (l is the number of clerks) is $(\alpha^{{x_i}^l}, \beta_R^{{x_i}^l}.\gamma^{-r_i^l})$ for the registrar onion, and $(\alpha^{{y_i}^l}, \beta_T^{{y_i}^l}.\gamma^{-r_i^l})$ for the teller one. As the teller onion is encrypted under the random seed values r_i, the encrypted values can only be revealed when all clerks collude. In the case of a ballot demand from a voter in the booth, these values can be revealed by a threshold set of registrars or the onion encoding candidate list can be decrypted by the voting machine using the corresponding secret key. Once the seed values are revealed, the candidate order π can be derived and the Prêt à Voter ballot form Figure 6.1 can be printed out in the booth on demand.

Tallying: In the tabulation phase, the existence of the index value as well as the onion may be used by the intruder to partition the mix and hence, the index values need to be absorbed into the onion value [RS06]. Suppose, the candidate order is determined by the cyclic shift rather than the full permutation π as previously, and r_i is the cyclic shift for the ith ballot and s is the index value. Now, the teller onions in the form of $(\alpha^{y_i}, \beta_T^{y_i}.\gamma^{-r_i})$ can be transformed into $(\alpha^{y_i}, \beta_T^{y_i}.\gamma^{s}.\gamma^{-r_i})$ after the election and before the tabulation phase. Subsequently, the new terms are sent to a re-encryption mixnet, where the new terms are re-encrypted and posted to the WBB in random order. Following this, a threshold set of decryption tellers take these terms and decrypt them to extract the plaintext value in the form $\gamma^{s-r_i} (\mathrm{mod}\, p)$. The original vote is then computed as $s - r_i (\mathrm{mod}\, n)$, where n is the number of candidates.

Auditing: As Prêt à Voter is intended to be a transparent and trustworthy voting system, auditing is an important counter-measure against incorrectly con-

structed ballots, incorrect recording of the votes and corrupt tellers. In order to audit ballot construction, a two sided ballot form mechanism is adopted in [RS06], which requires some modifications to generic Prêt à Voter ballot forms (see [RS06] for further information). However, there are other ways of auditing ballot construction, such as, using a single dummy vote or given the onion value, in which case the tellers are expected to return the candidate list on that ballot form. Moreover, in order to audit any malfunction or corruption by the mix tellers a mechanism called randomised partial checking (RPC), as proposed in [JJR02], can be performed. That is, the tellers reveal a randomly chosen half of their input and output links in such a way that there is no complete route from input to output, so that no ballot receipt can be traced, and the remaining links are hidden. A modification of a single value has a 50% chance of being caught and a corrupt teller therefore has only a $\frac{1}{2^n}$ chance of getting away with modifying n votes.

An Abstract Prêt à Voter System Structure

Figure 6.2 illustrates an election run with voter v and her interaction with the voting system, which also shows the approach and abstraction level taken when modelling Prêt à Voter. In detail, the protocol operates as follows:

- The authority in the voting system chooses a random value r from a seed space, R, computes the candidate list permutation π, using a publicly agreed function f, (so $f(r) = \pi$), and finally encrypts the random value r using the tellers' public key, $\mathsf{E}_{pk_T}(r)$, finally sending the data $\{\pi, \mathsf{E}_{pk_T}(r)\}$ to the voter.

- The voter chooses a candidate c, marks the ballot form finding the corresponding index value, i, and sends $\{i, \mathsf{E}_{pk_T}(r)\}$ to the WBB, discarding the permutation, π.

- The election official signs the receipt $\{i, \mathsf{E}_{pk_T}(r)\}$ and sends it back to the voter.

- The WBB first mixes the cast votes using re-encryption mixes and then publishes $\{i, \mathsf{E}_{pk_T}(r)\}$. The teller takes over the vote and reveals the random value r using its secret key, sk_T, then it calculates π candidate permutation as $f(r) = \pi$.

In the following subsection, a reasonably abstracted CSP model of Prêt à Voter is presented.

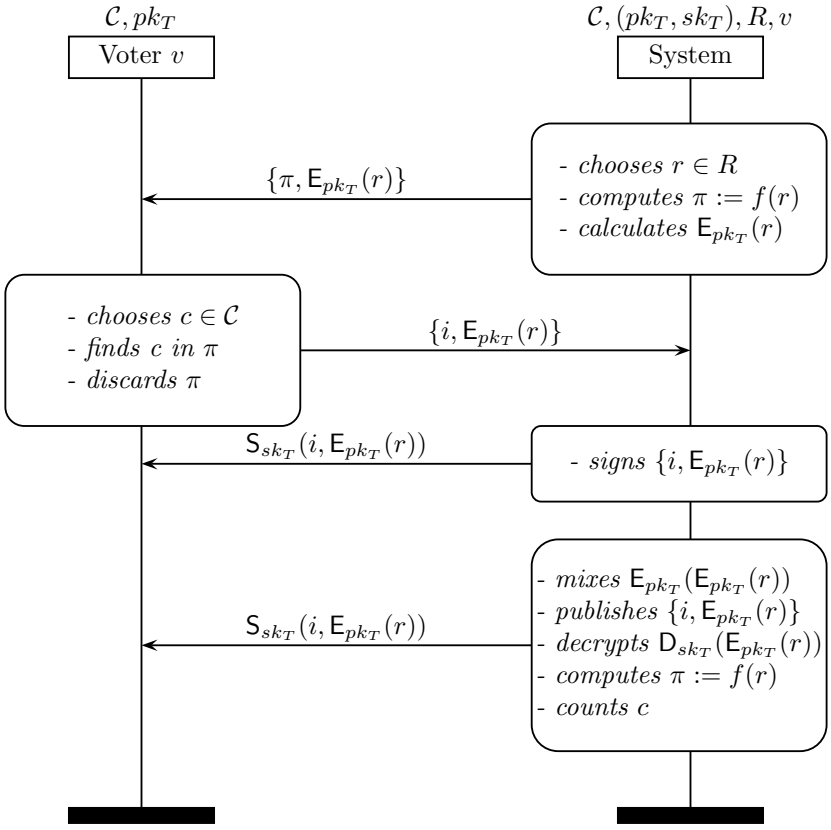

Figure 6.2: Prêt à Voter voter interaction with the system

6.2 Modelling Prêt à Voter

This section presents the CSP model of Prêt à Voter by first defining data-types, sets and the functions used, and then describing each process as in Chapter 5. Although, some of the processes are similar to the previous voting systems analysed for this research, each process behaves slightly differently depending on the modelling approach. For instance, in the previous chapter, ThreeBallot was modelled with the help of coordinates, however, in the Prêt à Voter modelling the approach used will be that introduced in Chapter 4.

To begin with, Figure 6.3 illustrates the message sequences between the protocol agents that form the Prêt à Voter voting system. In order to achieve the abstract behaviour of the voting system shown in this figure, the individual processes need to be modelled as follows:

In the following paragraphs, the functions, data-types and sets used to construct the events and the processes are described. Similar to the framework used in

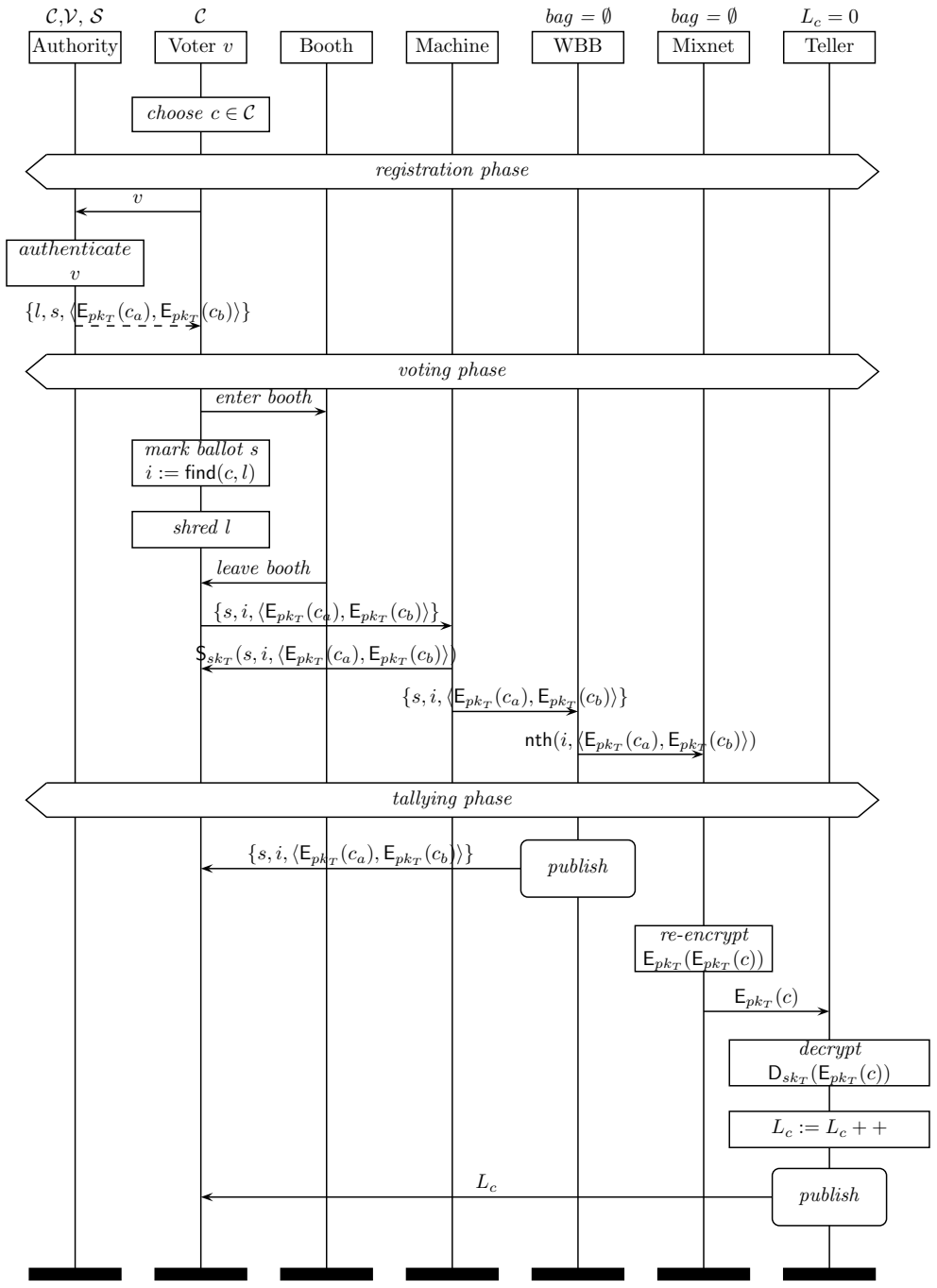

Figure 6.3: Prêt à Voter system model: the election process for a voter v, and candidate c.

Section 3.3, encryption is considered as a formal symbolic operation. The encryption function $\mathsf{encrypt}(pk, m)$ is the public-key encryption of the message m under the public key pk, and it uses a constructor $Encrypt.(f, f)$, which is denoted as $\mathsf{E}_{pk}(f)$, where $\mathcal{F}$ is the set of facts and $f \in \mathcal{F}$. In a similar manner, the re-encryption is modelled as a null operation (i.e., encryption does not change), so, $\mathsf{reencrypt}(pk, \mathsf{E}_{pk}(m)) = \mathsf{E}_{pk}(m)$ and therefore, the property of re-encryption is abstracted away in this model. Additionally, the key pair formed by pk_T and sk_T is the teller's encryption key pair and the public key is used to construct the ballot forms by the authority during the election. It is also used in the re-encryption phase by the mixnet. As the teller knows the secret key, he can extract the re-encrypted shuffled values and the onion values, which embed the actual vote, in the tallying phase. Moreover, in order to avoid state explosion in the model checker, the number of agents is limited in the model, with only two candidates, two voters and two serial numbers being used. For convenience, the names of the sets are abbreviated as follows: *candidates* as $\mathcal{C}$, *voters* as $\mathcal{V}$, *serials* as $\mathcal{S}$. Some of the other sets that are useful in modelling messages as data-types are:

$$
\begin{aligned}
candidates &= \{c_1, c_2\} \\
voters &= \{v_1, v_2\} \\
serials &= \{s_1, s_2\} \\
keypairs &= \{(pk_T, sk_T)\}
\end{aligned}
$$

A Prêt à Voter ballot form (Figure 6.4) consists of a LHS and a RHS. On the LHS, there is a candidate list, and on the RHS, there is a serial number, a grid that the voter places her mark in and an onion, representing the encrypted candidate list. The set *indices* consists of natural numbers, such as $\{1, 2\}$ here in order to symbolise marking action by the voter. For instance, suppose that two candidates are running in the election and the candidate list on the LHS is $\langle c_1, c_2 \rangle$. When the ballot form is given to the voter, the marking boxes are initially empty, so they are modelled with the data-type *emptylist*. If she wants to vote for c_1, she chooses 1, whereas if her preference is for candidate c_2, she fills in 2 and so on. Note that the voter can mark a ballot form only for the chosen candidate.

	s_1
c_1	
c_2	2
	$\langle \mathsf{E}_{pk_T}(c_1), \mathsf{E}_{pk_T}(c_2) \rangle$

Figure 6.4: Prêt à Voter ballot model in CSP: a vote for c_2 is expressed with the index value 2.

The function $lists()$ below produces the set of all possible candidate lists on a ballot form taking the candidate list $\mathcal{C}$ as a parameter. Similarly, the set *onions* covers all possible onion values, which embed the encrypted candidate lists under the teller's public key, pk_T, and are located at the bottom right of a ballot form. The sets of indices, lists and onions are abbreviated as $\mathcal{I}$, $\mathcal{L}$ and $\mathcal{O}$, respectively.

$$lists(\mathcal{C}) = \{\langle c\rangle\hat{}a \mid c \in \mathcal{C}, a \in lists(\mathcal{C} \setminus \{c\})\}$$
$$onions = \{\langle \mathsf{E}_{pk_T}(c_i), \mathsf{E}_{pk_T}(c_j)\rangle \mid c_i, c_j \in \mathcal{C}\}$$

Two special functions called find and nth as defined next are used to describe the actions taken by the agents. The find function is used by the voter process to see the corresponding grid for the candidate she has chosen. The function is defined by means of the head and tail functions; the former returns the first element of a non-empty sequence, the latter returns all but the first element of a non-empty sequence, for instance; $\mathsf{head}(\langle c_1, c_2, c_3\rangle) = \langle c_1\rangle$ and $\mathsf{tail}(\langle c_1, c_2, c_3\rangle) = \langle c_2, c_3\rangle$. The function nth is also defined using head and tail to extract the nth element of a sequence.

$$\mathsf{find}(c, l) = \begin{cases} 1 & \text{if } c = \mathsf{head}(l) \\ 1 + \mathsf{find}(c, \mathsf{tail}(l)) & \text{if } c \neq \mathsf{head}(l) \end{cases}$$

$$\mathsf{nth}(i, m) = \begin{cases} \mathsf{head}(m) & \text{if } i = 1 \\ \mathsf{nth}(i-1, \mathsf{tail}(m)) & \text{if } i \neq 1 \end{cases}$$

Subsequently, the message formats transmitted on the network can be modelled using the sets, functions and data-types above as follows.

$$\begin{aligned}
ballotforms &= \{\langle l, \langle s, i, o\rangle\rangle && \mid l \leftarrow \mathcal{L}, \\
&&& \quad s \leftarrow \mathcal{S}, \\
&&& \quad i \leftarrow \mathcal{I}, \\
&&& \quad o \leftarrow \mathcal{O}\} \\
castrhs &= \{\langle s, i, o\rangle && \mid s \leftarrow \mathcal{S}, \\
&&& \quad i \leftarrow \mathcal{I}, \\
&&& \quad o \leftarrow \mathcal{O}\} \\
encryptions &= \{\mathsf{E}_{pk_T}(c), \mathsf{E}_{pk_T}(\mathsf{E}_{pk_T}(c)) && \mid c \leftarrow \mathcal{C}\} \\
atomicfacts &= \{f && \mid f \leftarrow \cup\{\mathcal{V}, \mathcal{I}\}\}
\end{aligned}$$

6.2.1 Modelling Assumptions

The assumptions made in the modelling of CVS in Section 4.4.1 are also valid in the modelling of Prêt à Voter in this section. In addition to those, a real Prêt à Voter ballot form consists of a registrar and a teller onion. These are encrypted values embedding security sensitive informations and assumed not to be decrypted without the corresponding secret key. Additionally, the purpose of the registrar onion is to audit for malfunctioning voting machines and such like, it is not within the ambit of this thesis. Hence, in the modelling of Prêt à Voter, the ballot form includes only the teller onion, but not the registrar one as in the real system.

Additionally, the mixnet is treated as one single entity because the mixnet, which consists of a number of mix tellers, is assumed to be honest in the original proposal. Thus, in terms of modelling an honest mixnet there is no difference between an honest mixnet having multiple mix tellers and a mixnet with only one mix teller behaving honestly by shuffling all its inputs non-deterministically. Consequently, the mechanism for the threshold decryption of the encrypted votes cannot be used in the modelling of Prêt à Voter here as the onion values are only encrypted once by the honest mixnet using its public key. Thus, a single decryption teller knowing the corresponding public key can decrypt the encrypted values. Therefore, the decryption tellers working in a threshold way in the real system is not considered in the modelling of Prêt à Voter in this chapter. For the assumptions regarding cryptographic primitives and CSP, see Chapter 2.

In the re-encryption version of Prêt à Voter, the ballots can be pre-printed or can be printed on-demand in the booth machine. The former is considered here, whereby the election authority creates the ballot forms using the teller's public key and forwards pre-printed ballot forms to the voters (in practice, the ballot generation code is run on a diskless workstation, which generates the ballots, prints them, and then shuts down, keeping no record of its actions. The candidate lists are kept in only two places: printed on the ballot papers and on the WBB encrypted under the threshold public key). However, the election authority in this modelling behaves as an electoral official who creates ballot forms and issues them to the voters directly. Moreover, information about the candidate list is considered as flowing over a private channel, *collectform*, modelling an envelope, which ensures the privacy of the ballot form. Furthermore, the candidate list on the LHS is modelled here as a full permutation of candidates rather than cyclic shift—there is no difference in the case of two candidates as the set of permutations for these is the same as the set of cyclic shifts. However for the number of candidates more than two cyclic shifts should be used in the model to reflect the correct behaviour of the system. Using a cyclic shift, which is a subset of the set of permuted candidate lists, decreases the cardinality of the set of all

possible candidate lists, cutting down the required state space for the automated verification.

In the next section, the honest participant processes are defined.

6.2.2 Honest Participants

In the following subsections the individual processes that comprise the Prêt à Voter voting system model are expressed in CSP. Additionally, for each definition the abstraction level will be clarified further by comparing the real system and the CSP model of it.

Election Authority Process

The election authority first opens the election with the event *openElection*. Having taken possession of the list of eligible voters, serial numbers to assign, and the list of candidates, the authority authenticates the voters with their identification upon request from them, and issues each with an empty ballot form, containing a non-deterministically chosen serial number and a candidate list, over the channel *collectform*. The authority can perform these actions for as long as there are eligible voters and serial numbers to allocate. Finally, he closes the election with the *closeElection* event.

$$
\begin{aligned}
&Authority \mathrel{\widehat{=}} openElection \rightarrow Authority_1(\mathcal{V}, \mathcal{S}, \mathcal{L}) \\
&Authority_1(ids, serials, lists) \mathrel{\widehat{=}} \\
&\quad closeElection \rightarrow STOP \\
&\quad \Box \\
&\quad \Box_{v \in ids} \left(\begin{array}{l} auth.v \rightarrow \\ \quad \sqcap_{\substack{s \in serials \\ l \in lists}} \left(\begin{array}{l} collectform.v.\langle l, s, emptylist, \mathsf{E}_{pk_T}(l) \rangle \rightarrow \\ Authority_1(ids \setminus \{v\}, serials \setminus \{s\}, l) \end{array} \right) \end{array} \right)
\end{aligned}
$$

As in the conventional voting system model, the alphabet of each process is the set of all events that a process may perform. Thus, the alphabet of *Authority* is as below; the alphabets for the remaining processes in the system can be inferred from the CSP definitions.

$$\alpha Authority = \{| openElection,\ auth,\ collectform,\ closeElection |\}$$

Voter Process

Having chosen a candidate to vote for, the voter authenticates herself and accepts any ballot form given by the election authority. Then, she goes into the booth to select a candidate on the channel *mark*, and after destroying the candidate

list, she leaves the booth. Afterwards, she casts her vote under the supervision of the election authority using the voting machine supplied. Having cast the ballot form, the voter is provided with a receipt of her vote, which is the RHS of the ballot form, including the serial number—in the real system, the receipt is signed by the electoral official as a proof of casting the vote, hence, if the voter cannot find a matching receipt on the WBB, she can appeal using the signed receipt as her proof. However, in this modelling it is omitted as its purpose, i.e., verification, is not within the scope of this thesis. Once the voter gets her receipt and leaves the polling station, voting finishes for her. The serial numbers on the receipts are used by the voter for verifying whether her ballot appears on the WBB unchanged.

$$
\begin{aligned}
&Voter(v) \mathrel{\widehat{=}} \\
&\prod_{c \in \mathcal{C}} \left(\begin{array}{l} choose.v.c \rightarrow openElection \rightarrow auth.v \rightarrow \\ \quad \Box_{\substack{l \in \mathcal{L} \\ s \in \mathcal{S} \\ i := \mathsf{find}(c,l)}} \left(\begin{array}{l} collectform.v.\langle l, s, emptylist, \mathsf{E}_{pk_T}(l) \rangle \rightarrow \\ enterBooth.v \rightarrow \\ mark.\langle l, s, i, \mathsf{E}_{pk_T}(l) \rangle \rightarrow \\ shredLHS.\langle s, i, \mathsf{E}_{pk_T}(l) \rangle \rightarrow \\ leaveBooth.v \rightarrow \\ cast.\langle s, i, \mathsf{E}_{pk_T}(l) \rangle \rightarrow \\ receipt.\langle s, i, \mathsf{E}_{pk_T}(l) \rangle \rightarrow \\ closeElection \rightarrow STOP \end{array} \right) \end{array} \right)
\end{aligned}
$$

The above process should be followed by all eligible voters and hence the voter processes for each voter are put in parallel in order to model the behaviour of all voters in an election run.

$$Voters \mathrel{\widehat{=}} \|_{v \in \mathcal{V}} Voter(v)$$

Machine Process

The *Machine* process models the voting machine located in the polling station outside the booth, which is a multi-function machine that can scan the votes and print the receipts for the voters. That is, it synchronises on the event *openElection* with the authority and the voters, then starts receiving the cast right-hand sides of the ballot forms and printing out the receipts for the voters before the election is closed. Because the machine accepts any RHS cast by the voters, the external choice operator is used. When the process is combined with the other agents on the channel *receipt*, it will send the copies of the receipts to the WBB.

$$
\begin{aligned}
&Machine \mathrel{\widehat{=}} openElection \rightarrow Machine_1 \\
&Machine_1 \mathrel{\widehat{=}} closeElection \rightarrow STOP \\
&\qquad\qquad \Box \\
&\qquad\qquad \Box_{r \in castrhs} \Big(\; cast.r \rightarrow receipt.r \rightarrow Machine_1 \; \Big)
\end{aligned}
$$

Web Bulletin Board Process

The WBB stores and publishes all cast ballots for verification purposes as well as requesting shuffling from the mixnet, which anonymises the cast votes.

The following process *Wbb* starts receiving the digital copies of the cast right-hand sides returned by the *Machine* process on the channel *receipt*, once the *openElection* event has occurred. It keeps track of the receipts in a set called *bag*, which is initially an empty set. The WBB process can also request shuffling for the votes by sending them, one by one, to the mixnet process on the channel *mixReq*—in the Prêt à Voter voting system, the mix requests can also be sent as a batch of votes rather than one by one. This does not affect the analysis here as the votes are shuffled and output by the mixnet non-deterministically. However, serial numbers are stripped off beforehand. Once the election is closed and all votes have been sent for shuffling, the process publishes all cast votes kept in the set *bag*. These consist of a serial number, an index indicating where the mark is and an onion value, which is the encryption of the candidate list on the LHS of the ballot form.

$$
\begin{aligned}
&Wbb \mathrel{\widehat{=}} openElection \rightarrow Wbb_1(\emptyset) \\
&Wbb_1(bag) \mathrel{\widehat{=}} closeElection \rightarrow Wbb_2(bag) \\
&\qquad\qquad \Box \\
&\qquad\qquad \Box_{\substack{s \in \mathcal{S} \\ i \in \mathcal{I} \\ o \in \mathcal{O}}} \left(\begin{array}{l} receipt.\langle s, i, o \rangle \rightarrow \\ mixReq.\mathsf{nth}(i, o) \rightarrow \\ Wbb_1(bag \cup \{\langle s, i, o \rangle\}) \end{array} \right)
\end{aligned}
$$

$$
\begin{aligned}
&Wbb_2(\emptyset) \quad \mathrel{\widehat{=}} bagempty \rightarrow STOP \\
&Wbb_2(bag) \mathrel{\widehat{=}} \quad \sqcap_{r \in castrhs} \quad pub.r \rightarrow Wbb_2(bag \setminus \{r\})
\end{aligned}
$$

Mixnet Process

The *Mix* process behaves as a mixnet, which performs a mix for the digital copies of the receipts. Although a number of mix tellers exist in the real system, only a single honest mix server is considered. The following process below behaves as a perfect mixnet, which anonymises the cast votes arbitrarily. In more detail, as the cast votes arrive to the mixnet, one by one, the process keeps them in a

set called *Batch*, which is initially empty. Before shuffling the votes, the process *Mix* re-encrypts each encrypted vote with the teller's public key, pk_T, which does not change the actual vote. When all the votes have been re-encrypted (no vote left to mix as indicated by the event *bagempty*), the teller takes over the shuffled votes non-deterministically and one by one.

In this model of the mixnet, it is assumed that the mix is honest and does not reveal any information about the mapping from input to output. As having more than one mixnet would not make any difference as a consequence of the non-deterministic construction of the mixnet, one is enough to re-encrypt and shuffle the votes here. In the process, the parameter i is used to create a bag-like set, where the same data can be stored many times unlike normal sets.

$$
\begin{aligned}
&Mix \mathrel{\widehat{=}} openElection \rightarrow Mix_1(0, \emptyset)\\
&Mix_1(i, Batch) \mathrel{\widehat{=}} closeElection \rightarrow Mix_2(Batch)\\
&\qquad\qquad \Box\\
&\qquad\qquad \Box_{c \in \mathcal{C}} \left(\begin{array}{l} mixReq.\mathsf{E}_{pk_T}(c) \rightarrow \\ reencrypt.\mathsf{E}_{pk_T}(\mathsf{E}_{pk_T}(c)) \rightarrow \\ Mix_1(i+1, Batch \cup \{(i, \mathsf{E}_{pk_T}(c))\}) \end{array} \right)
\end{aligned}
$$

$$
\begin{aligned}
&Mix_2(\emptyset) \mathrel{\widehat{=}} bagempty \rightarrow STOP\\
&Mix_2(bag) \mathrel{\widehat{=}} \prod_{(i,e) \in bag} mixOut.e \rightarrow Mix_2(bag \setminus \{(i, e)\})
\end{aligned}
$$

Decryption Teller Process

As remarked previously, the existence of a threshold set of decryption tellers is present in the real system, however, here, only a single honest teller is considered enough to model the CSP decryption of teller process as given below.

The *Teller* process takes over the shuffled re-encrypted onion values from the mixnet, transferred on the channel *mixOut*. Because re-encryption and ballot generation are performed under the teller's public key pk_T, he can decrypt each of the encrypted cast votes, and tally the plaintext values according to the candidates. Finally, the process announces the result for each candidate on the channel *total*.

$$
\begin{array}{l}
\mathit{Teller} \mathrel{\widehat{=}} \mathit{openElection} \rightarrow \mathit{Teller}_1(0,0) \\
\mathit{Teller}_1(m,n) \mathrel{\widehat{=}} \\
\quad \Box_{c \in \mathcal{C}} \left(\begin{array}{l} \mathit{mixOut}.\mathsf{E}_{pk_T}(c) \rightarrow \\ \mathit{decrypt}.\mathsf{D}_{sk_T}(\mathsf{E}_{pk_T}(c)) \rightarrow \\ \mathit{tally}.c \rightarrow \\ \left(\begin{array}{l} \text{if } c = c_1 \text{ then } \mathit{Teller}_1(m+1,n) \\ \quad \text{else} \left(\begin{array}{l} \text{if } c = c_2 \text{ then} \\ \quad \mathit{Teller}_1(m,n+1) \text{ else } \mathit{STOP} \end{array} \right) \end{array} \right) \end{array} \right) \\
\quad \Box \\
\quad \mathit{bagempty} \rightarrow \mathit{total}.c_1.m \rightarrow \mathit{total}.c_2.n \rightarrow \mathit{SKIP}
\end{array}
$$

System Process

The following process System_{PaV} models the abstract behaviour of the Prêt à Voter voting system. Similar to previous systems modelling, a number of sanity checks are deployed in order to gain more confidence in the correct behaviour of the CSP model of this voting system and the details regarding these are presented in Appendix A.3.

$$\mathit{System}_{PaV} \mathrel{\widehat{=}} \mathit{Voters} \parallel \mathit{Authority} \parallel \mathit{Machine} \parallel \mathit{Booth} \parallel \mathit{Wbb} \parallel \mathit{Mix} \parallel \mathit{Teller}$$

In the overall flow of the system process, all voters synchronise on *openElection* and *closeElection* events with the election authority pairwise. Thus, each voter performs an *openElection* event to begin her voting process. Each voter must also perform a *closeElection* event after casting their individual vote and leaving the polling station. Not only all voters synchronise on election opening and closing with the authority, but also other system components, such as, the WBB and mixnet processes. In addition, the decryption teller process announces the result for each candidate once there is no ballot data left to decrypt and count.

The following subsection defines the intruder model for the analysis of this voting system.

6.2.3 The Passive Attacker

The passive intruder framework defined in Chapter 4 will be deployed in this analysis and similar to the previous analysis, the capabilities of the intruder need to be specified. Regarding this, as usual, the intruder is able to observe any public data. As mentioned in the *Authority* process, the authority or electoral official should not get to know the candidate order on the Prêt à Voter ballot form that is used by a particular voter because this could then be used to violate her privacy. Note that the voter anonymity or vote privacy in Prêt à Voter depends on the link between the candidate order and the receipt. Hence, no one should be able

to know the corresponding candidate order for a specific receipt. Moreover, the event *mark*, which models the voter marking her ballot form, also needs to be unobservable. Therefore, the channels; *collectform* and *mark*, need to be private and hence, any data on these channels should be hidden from the intruder. The following reflects the system model with the private channels abstracted and the rest of the channels in the model are public including the receipts taken away by the voters.

$$System'_{PaV} \mathrel{\widehat{=}} System_{PaV} \setminus \{| \, collectform, mark \, |\}$$

The previous assumptions made above are model-specific, i.e., depending on the model, channels that need to be hidden may change. Moreover, the generic assumptions that can be made for any cryptographic voting system are as follows. Firstly, it is assumed here that the public-key infrastructure is secure, and the observer does not have enough computational power to break the key pairs. As encryption and decryption are modelled symbolically, any cryptographic vulnerabilities and attacks are not considered. Similarly, two ciphertexts encoding different plaintext should look the same to the intruder as long as he does not possess the secret key that he can use to decrypt and compare them. In other words, the encryptions on the channels need to be masked, so the intruder cannot distinguish them. To this end, the special function $\mathsf{maskFact}()$ is deployed, which converts all encrypted data to one single value, *ciphertext*, but leaves other data unchanged. Note that in CSP a function can be defined as the following due to its pattern matching property. Hence, the data item is tried in top to bottom order, i.e., if it fails to match the pattern in the first function definition, then the second is tried.

$$\begin{aligned} \mathsf{maskFact}(\mathsf{E}_{pk_T}(m)) &= ciphertext \\ \mathsf{maskFact}(x) &= x \end{aligned}$$

Similarly, as the onions on the ballot forms are sequences of ciphertexts, $\mathsf{mask}()$ is defined as follows:

$$\begin{aligned} \mathsf{mask}(\langle\rangle) &= \langle\rangle \\ \mathsf{mask}(\langle x \rangle \hat{} xs) &= \langle \mathsf{maskFact}(x) \rangle \hat{} \mathsf{mask}(xs) \end{aligned}$$

As a result, the abstracted model of the Prêt à Voter voting system, $System^{\mathsf{abs}}_{PaV}$, can be defined as follows.

$$System^{\mathsf{abs}}_{PaV} \mathrel{\widehat{=}} System'_{PaV} [\![shred.\langle s, x, \mathsf{mask}(o)\rangle / shred.\langle s, x, o\rangle]\!] [\![cast.\langle s, x, \mathsf{mask}(o)\rangle / cast.\langle s, x, o\rangle]\!] [\![receipt.\langle s, x, \mathsf{mask}(o)\rangle / receipt.\langle s, x, o\rangle]\!] [\![pub.\langle s, x, \mathsf{mask}(o)\rangle / pub.\langle s, x, o\rangle]\!] [\![mixReq.\mathsf{mask}(enc) / mixReq.enc]\!] [\![mixOut.\mathsf{mask}(enc) / mixOut.enc]\!] [\![reencrypt.\mathsf{mask}(enc) / reencrypt.enc]\!]$$

It is further assumed that the computing devices are legitimate unless it is stated that the device is corrupted. The assumptions relating to the channels *collect-form, mark* are that the booth is private, and the envelope with the ballot form inside that is given to the voter hides the candidate order from the intruder. The following section presents the first automated anonymity analysis of Prêt à Voter.

6.3 Automated Anonymity Verification

This section checks whether the Prêt à Voter model with honest participants, $System^{\mathsf{abs}}_{PaV}$, meets the anonymity requirement defined in Section 4.3 Definition 2 under the presence of the passive attacker model. Informally, according to the definition, the normal system and the system in which two votes are swapped over should be trace equivalent. The latter is the specification that needs to be defined in order to check the system against it. As the voter decides which candidate to vote with the *choose* event, these events need to be anonymised. Thus, the specification for $System^{\mathsf{abs}}_{PaV}$, where $A = \{| \, choose \, |\}$ is defined as follows:

$$Spec_A(System^{\mathsf{abs}}_{PaV}) \mathrel{\widehat{=}} System^{\mathsf{abs}}_{PaV} [\![choose.v_1.c_x, choose.v_2.c_x / choose.v_2.c_x, choose.v_1.c_x]\!]$$

Consequently, in order for the Prêt à Voter model to provide anonymity, the following trace equivalence should hold.

$$Spec_A(System^{\mathsf{abs}}_{PaV}) \equiv_{\mathrm{T}} System^{\mathsf{abs}}_{PaV}$$

FDR verifies that the trace equivalence holds, meaning that the intruder cannot distinguish between these two systems and hence, the Prêt à Voter model provides anonymity.

6.4 Analysis under Alternative Assumptions

In the previous section, Prêt à Voter was analysed with respect to a passive attacker who can observe the public information on the channels that are available

to him. This section investigates anonymity from the insider point of view, such as that of a corrupt electoral official. The framework is flexible enough to model such behaviours in cryptographic voting system models too with the renaming and hiding operators.

In the case of a corrupt authority or electoral official, it is expected that the private information held by them is no longer secret. That is, the private channel *collectform*, on which all ballot information is transferred from the official to the voter, is no longer private, and can be used to deduce how the voter has voted. In order to verify whether the anonymity is broken in such a case, it is necessary that *collectform* is not hidden any more. Hence, the new abstracted model from authority's point of view can be defined using the following process rather than the process ${System'}_{PaV}$ used before.

$${System''}_{PaV} \mathrel{\widehat{=}} System_{PaV} \setminus \{| \, mark \, |\}$$

Consequently, the abstracted process $System^{\mathsf{abs}}_{PaV}$ should be defined in terms of ${System''}_{PaV}$ exactly the same way as described above and in order to verify whether the requirement is held by this model the same trace equivalence needs to be checked with this new abstracted model.

As expected, FDR produces a number of counter-example traces, meaning that the trace equivalence does not hold. One of the traces is as follows:

$\langle choose.v_1.c_2,$
$choose.v_2.c_1,$
$openElection,$
$auth.v_1,$
$collectform.v_1.\langle\langle c_1, c_2\rangle, s_1, emptylist, \langle \mathsf{E}_{pk_T}(c_1), \mathsf{E}_{pk_T}(c_2)\rangle\rangle,$
$enterBooth.v_1,$
$shredLHS.\langle s_1, 2, \langle ciphertext, ciphertext\rangle\rangle\rangle$

The trace illustrates that when the voter v_1 votes for c_2 casting the ballot form with the candidate order $\langle c_1, c_2\rangle$, the intruder deduces for whom the voter v_1 has voted. This is because the first time that the intruder can see v_1's RHS on the *shredLHS* channel, he can observe that the RHS of the ballot form has a mark for the second candidate (shown as 2), which corresponds to the candidate c_2 on the candidate list he observed before. Therefore, the intruder, with the additional information from the corrupt electoral official, can break the anonymity of the voters in this modelling of Prêt à Voter.

A similar approach can be taken to model any corrupt agents on the system in the sense that they follow the protocol, but leak information. For instance, the

intruder may be allowed to see one of the voter's candidates list or the voter can share what she knows with the intruder, which can be modelled by only hiding v_1's private channels, but not v_2. However, this would create a corner case, where in an election there exists only one honest voter, which was discussed in Section 4.6. Moreover, a corrupt mixnet may also be modelled and to do so, its non-deterministic behaviour needs to be made deterministic. For instance, the mixnet can output the re-encrypted votes in a first-in-first-out (FIFO) fashion or last-in-first-out (LIFO). In such cases, the intruder would notice the difference between the normal system and the system where two votes are swapped over, because a sequence of mixnet output for one is not possible for the other. Such scenarios can easily be modelled in CSP, and checked using FDR.

6.5 Conclusion

Although, the analysis in this chapter has been performed with a reasonably abstracted model of Prêt à Voter, which does not cover some of its aspects, such as a threshold set of tellers, registrar onion and so on, it has sufficed to show how some of the cryptographic primitives in voting systems can be modelled using the framework defined in Chapter 4. Moreover, this analysis further validates the suitability of the anonymity definition given in Section 4.3 Definition 2. Additionally, it was demonstrated that the passive attacker model is very flexible to capturing any information leakage from the corrupt system participants.

6.6 Challenges Faced in the Modelling and Analysis

This section presents the challenges faced during the modelling of Prêt à Voter and anonymity analysis of it. The main issues was to come up with a model of Prêt à Voter that behaves as close to the real system as possible and that could be used for the automated verification. Thus, a number of abstractions were needed in the model due to state explosion problem. For instance, a perfect mixnet was modelled consists of only one mix server unlike the real system in which the mixnet consists of at least 5 mix servers. These abstraction are explained in detail in Section 6.2.1.

Another difficulty was to get rid of false-positives during the automated analysis. For instance, when the encrypted messages are left as they are in the model, the intruder can distinguish two different ciphertexts as if he possesses the corresponding secret key. Thus, a masking function, maskFact(), was needed in order to hide the differences between two ciphertexts. This function renames a number

of events, which carry confidential information that should be hidden from the observer (see Section 6.2.3 for further details about this masking function).

6.7 Summary

In this section, the paper-based voter verifiable cryptographic voting system Prêt à Voter was modelled in CSP and its first automated analysis with respect to the anonymity requirement was performed. It was shown that Prêt à Voter provides anonymity under the assumption of honest behaviour of the participants in the system. Furthermore, the model has also been investigated under different trust assumptions, such as, the presence of a misbehaving electoral official and mixnet. In both cases, it was verified that the abstract model of Prêt à Voter does not provide anonymity. However, in the real system, these two assumptions are not an issue. That is, for the former, as the election authority in this modelling of Prêt à Voter is a combination of a diskless distributed ballot generation engine and a poll worker, who does not see the candidate orders on the ballot forms, in the real system, the secret information on ballot forms can only be leaked by the envelopes, which do not leak by assumption. For the latter, assuming a corrupt mixnet would be against the protocol design as it is a distributed process and assumed not to be so.

In the next chapter, automated analysis of cryptographic voting systems under an active and more powerful intruder than the passive attacker along the lines of a Dolev-Yao intruder [DY83] as defined in Section 3.3 will be investigated.

Chapter 7

Adapting the Dolev-Yao Intruder Model to Voting Systems

This chapter* presents a novel intruder model for automated reasoning about anonymity and secrecy properties of voting systems. It is much stronger than the passive attacker used in the previous chapters as it behaves as a Dolev-Yao intruder model [DY83]. This type of intruder not only observes a protocol run, but also interacts with the protocol participants, overhears communication channels, intercepts and spoofs any messages that he has learned or generated from any prior knowledge. This approach is inspired by lazy spy [RG97], which is designed for cryptographic protocol analysis and called "lazy" as it avoids the eagerness of pre-computation of unnecessary inferences [Ros97]. The approach has already been introduced in Chapter 3, however, in order to apply this intruder model to voting systems, a few modifications are needed in relation to existing channel types and the deductive system. For the former, we benefit from Creese *et al.* [CGRZ03, CGH$^+$05], who defined various channels for different threat models in ubiquitous computing environments. For the latter, a large deductive system is constructed regarding the messages transmitted on the voting system model channels.

When describing the approach followed in this chapter, the vVote voting system will be used, which is based on Prêt à Voter [Rya04, CRS05], and is under development for use in Victorian Electoral Commission (VEC) elections in 2014 [BCH$^+$12a, BCH$^+$12b, Cul13] in Australia. The intended system will be used in these elections, involving over three million voters, electing 88 legislative

*This chapter is mainly based on the paper accepted by the 13th International Workshop on Automated Verification of Critical Systems (AVOCS'13) [MH13].

assembly and 40 legislative council representatives, using a mixture of the alternative vote (AV), also called instant-runoff voting (IRV), and the single transferable vote (STV). Most of the key features of Prêt à Voter are retained in the vVote system. However, to adapt the system to such a complex election setup, a number of modifications have been necessary in the system design; for instance, the inclusion of distributed ballot generation, an electronic ballot marker to assist the voter in filling out the ballot, and print-on-demand ballots for voters who are voting away from their registered polling station.

In terms of voting system requirements addressed during the analysis of the vVote system, the anonymity definition given in Section 4.3 Definition 2, and an adaptation of the secrecy definition used in the analysis of the NSPK in Section 3.3.4 are covered.

The participants of the vVote voting system will be modelled in CSP as usual, however, the processes will involve more cryptographic primitives than the previous chapter and the intruder model will be much more complex than the passive attacker. That is, this chapter will involve analysis of a complex system that requires much more state space than the previous analyses of voting systems in this thesis. However, it will be demonstrated that the FDR model checker and the CSP formal language are suitable methods for mechanically analysing cryptographic voting systems.

The chapter is structured as follows. Section 7.1 presents an overview of the vVote voting system. In Section 7.2, the vVote voting system is modelled in CSP except for some parts of it, such as distributed key generation, and the lazy spy intruder model is further extended for the analysis of voting systems with additional data-types, channel types and deduction rules. Section 7.3 analyses the system model regarding the formal specification of anonymity, whilst Section 7.4 investigates the analysis of the model under alternative assumptions, such as, there being a corrupt election authority, who is preparing and distributing digital ballots. In Section 7.5, a formal specification of secrecy is given for voting systems. Section 7.6 presents a conclusion and discussion on the findings, with a chapter summary being provided in Section 7.8.

7.1 The vVote Voting System

Over the last few decades many trustworthy voting systems have been proposed. However, only a few have been deployed in large-scale real elections. Regarding these, Scantegrity II [CCC+08] was the first E2E voting system deployed in a binding governmental election on November 3, 2009 [CCC+10], involving a relatively small electorate with 1728 voters. Additionally, Norway used an internet voting protocol [Gjø10] in the municipal elections in September 2011, in

which more than 25,000 voters cast their votes using this protocol. Moreover, STAR-Vote [BBK+12] is another voter verifiable DRE-style voting system, which is going to be used in Travis County, Texas, the United States, involving over 450,000 registered voters.

The vVote voting system is an E2E paper-based electronic voting system based on Prêt à Voter [Rya04, CRS05]. However, a number of modifications have been made to the original Prêt à Voter system. The main difference is that an electronic device is deployed in order to facilitate accommodating a candidate list with over 30 candidates on the ballot forms. This also helps voters to indicate their preferences among many other candidates. However, deploying these computers requires further trust on them as they know about the voters' choice at the time of voting. Hence, a misbehaving device can violate voter anonymity or vote privacy and as a result, for further confidence in the design of this promising real-world voting system, formal verification is needed.

The vVote ballot form illustrated in Figure 7.1 is similar to a Prêt à Voter one. On one side there is a randomly permuted candidate list and a QR code at the bottom that records the permuted candidate order, on the other are marking boxes, a unique serial number and another QR code, corresponding to the onion that embeds the candidate order.

In the construction of onions, exponential and normal ElGamal public key algorithms are used. For instance, for the legislative council election, a value in G_q is chosen to represent each party name. Suppose the value is α, then the corresponding onion for this party is the ElGamal term $\mathsf{E}_{pk}(\alpha) = (g^r, \alpha h^r)$, where $h = g^x$, $r \in \mathbb{Z}_q^*$ is a random value and $x \in \mathbb{Z}_q^*$ is a random secret value.

7.1.1 Ballot Generation

Ballot generation can be realised on the machine that prints the ballot form in the booth or in a distributed fashion that is similar to that described in [Rya06, RT10], i.e., a number of candidate list mixers shuffle the encrypted candidate names for each vote, which ensures that the candidate ordering is random and not generated by a single party. This eliminates single point failures. In the distributed version, a list of encrypted ballots, including a serial number, the onion encoding the candidate list, and the list of encrypted candidate names for the printer with a proof of correspondence is produced. The printers' (print-on-demand (POD) client) candidate list is encrypted under a threshold key shared across a set of candidate list key sharers, called the POD service. Hence, in order for a printer to obtain the candidate list, it generates a blinding factor, encrypts it under the POD service public key, and sends it to the POD service with a proof of knowledge. Afterwards, having received the encrypted candidate list blinded

Figure 7.1: vVote ballot form [Cul13]

by itself, the printer removes the blinding factor, and prints the candidate list.

7.1.2 Election Phases

In the following paragraphs the election phases covering the voting ceremony and the vVote system components are explained.

Pre-election The pre-election phase covers the preparation of election material before the polling station opens. In this period, digital ballots are generated in a distributed fashion that are encrypted under the POD service's and the election authority's public key, before being committed to the public bulletin board. In order to speed up decrypting of the votes, a lookup table of all possible voting permutations is created. Additionally, mixnets are set up, and key generation is performed in this phase.

Vote Casting This phase starts with polling stations opening and lasts until the election is closed, with no further votes being allowed to be cast. The POD

service allocates and transfers pre-prepared digital ballots to a print station in the polling booth during the election. Having registered with the poll worker, the voter or the former interacts with the POD service to get a ballot paper as illustrated in Figure 7.1. The POD client will derive the permuted candidate list on the ballot form when it is actually being printed in the polling station and the ballot paper format in vVote is similar to the Prêt à Voter one. However, instead of an onion for the encrypted candidate list, it has a QR barcode that consists of a digitally signed serial number. Having scanned this barcode to the electronic ballot marker (EBM), the voter can see and cast her ballot form in an electronic environment. The EBM is a new front-end component that interacts with the WBB to submit the vote and receive a digitally signed receipt for it, which is then printed for the voter for verification purposes by a receipt printer in the booth. The WBB commits, records and broadcasts the ballot data generated during the election and also signs the serial numbers allocated by the ballot manager, thereby ensuring their uniqueness. Once the voter has cast her vote and received her receipt, she then leaves the polling station.

Post Election Post election is the phase where the cast votes are mixed by the mixnets, decrypted and tallied by a set of key sharers, such that only a threshold set of these sharers can perform decryption. The results are then announced by the bulletin board.

7.1.3 vVote POD Service and Protocol

The POD service (also called candidate list key sharers) provides distribution of digital ballots in a distributed manner to the polling stations in any district. As the digital ballots are prepared and committed to the WBB before the election, this service facilitates the print-on-demand ballot distribution in real time (The details about the POD service and any other part of the vVote system can be found in the software design technical report [Cul13]. Despite the fact that it is still being updated in respect of design changes, it is considered to be a natural stable description for use in the analysis in this chapter).

In the ballot generation procedure, the randomised candidate order of a ballot is encrypted under the election public key pk_{EA} and it is then transformed to an encryption under the POD service's public key pk_{PS} without revealing the underlying message, as described in [Jak99]. The same transformation technique is also used in the POD protocol illustrated in Figure 7.2 to transform the encryptions on the digital ballots into the designated POD client's public key pk_{PC} and these transformed ciphertexts cannot be decrypted by anyone other than the designated printer.

In more detail, when the voter authenticates in the polling station, the poll

worker requests a random nonce-like $sessionID$ from the administrator machine. Following this, the poll worker sends the $sessionID$ to the POD service, where it is signed and stored. Having received the signed $sessionID$, the poll worker hands it to the voter in barcode form. Then, the voter scans the barcode to the POD client, which signs the $sessionID$ and submits it to the POD service. Subsequently, the POD service signs the district and the $sessionID$ and submits them to the ballot manager. Following that, the ballot manager finds the next available serial number for that district, assigns it to the submitted $sessionID$, and notifies the WBB for this assignment by signing them using its secret key sk_{BM}. The WBB then sends a confirmation of this assignment to the ballot manager, which returns this to the POD service. Now, the transformation of the public keys that encrypt the ballot form takes place from pk_{PS} to pk_{PC}. Finally, the POD service signs the serial number and sends along with the transformed ciphertexts to the POD client, which can then decrypt these and print the actual ballot for the voter.

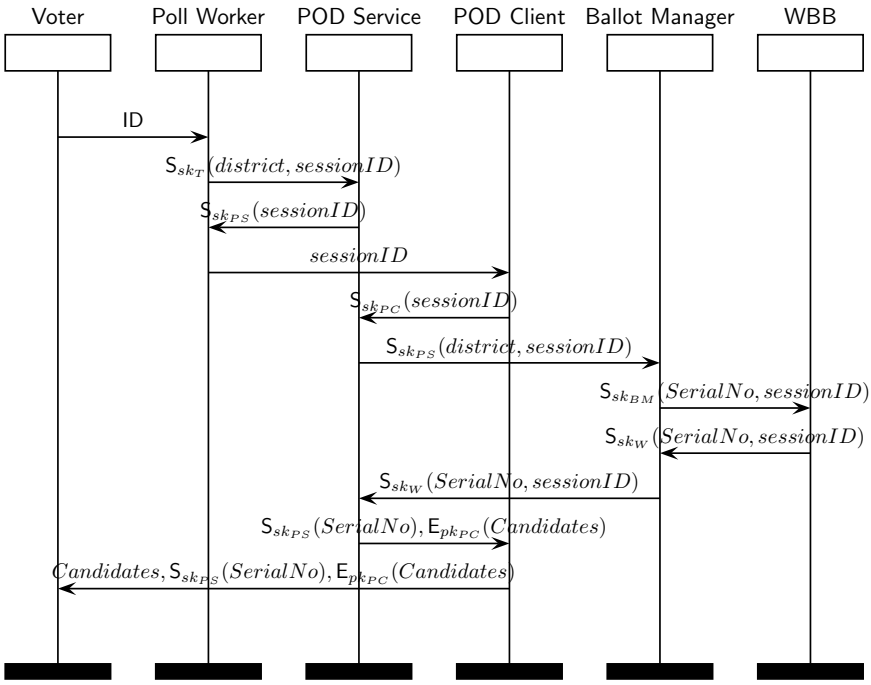

Figure 7.2: Message sequence chart of POD protocol

In this new system, the electronic ballot marker (EBM) is particularly interesting as it forms the key distinctive characteristic of vVote, being the only device in the system that knows how a particular voter has voted. That is, when the voter transfers her actual ballot form to the EBM, the candidate list on her form is also transferred to the EBM, while it is destroyed and kept secret in Prêt à

Voter. One of the assumptions made in [BCH+12a, BCH+12b, Cul13] is that the POD client, where physical ballot form is printed, and the EBM, are located in a private environment, such as, a voting booth.

In the next section the vVote voting system is modelled in CSP as well as there being description of the adaptation of the lazy spy intruder model for the analysis of voting systems.

7.2 Modelling vVote and Active Intruder

Conventionally, security protocols consist of several agents sending messages to each other on the medium they share or on direct communication channels. The vVote voting system is modelled in terms of a number of agent processes that run in parallel and these processes behave as the corresponding components of the voting system.

In the following sections, the messages sent on the channels of the model are defined. Secondly, the different kinds of channels that are needed for the analysis are introduced along with the process definitions for each agent, which compose the voting system model. Following this, the lazy spy intruder model acting as a Dolev-Yao intruder is adapted to analyse such voting systems. Finally, the system model and active intruder model are put together in order to reason about the system as a whole later in the analysis section.

7.2.1 Data-types and Messages

As introduced in Chapter 3, cryptographic primitives, such as encryptions and signatures, are modelled as symbolic objects like the agents, the public and secret keys, the nonces and serial numbers. For instance, encryption: $\mathsf{E}_{pk}(f)$, decryption: $\mathsf{D}_{sk}(f)$, signature: $\mathsf{S}_{sk}(f)$. Additionally, apart from these, the other messages, which can be a collection of these cryptographic primitives, are also modelled as the data-types. In this respect, the message including a serial number and an encrypted candidate list (called raw ballots here) is denoted as $\mathsf{Raw}(s, \mathsf{E}_{pk}(l))$, and a digital ballot message formed by a signed serial number and an encrypted candidate list is modelled as $\mathsf{DigB}(\mathsf{S}_{sk}(s), \mathsf{E}_{pk}(l))$. Similarly, a ballot form consisting of a candidate list, a serial number, and an index value, is $\mathsf{B}(l, s, \mathsf{Ind}.i)$; a message with a serial number and an index value forming the marking boxes on the ballot form, called *castrhs*, which is demonstrated as $\mathsf{RHS}(s, \mathsf{Ind}.i)$; and a receipt is the signed *castrhs* denoted as $\mathsf{R}(\mathsf{S}_{sk}(\mathsf{RHS}(s, \mathsf{Ind}.i)))$. Finally, a message consisting of an index value and an encrypted candidate list is called a *vote* and shown as $\mathsf{V}(\mathsf{Ind}.i, \mathsf{E}_{pk}(l))$. Figure 7.3 depicts how these messages are composed in the model.

In order to compose these messages, the model consists of several finite sets of facts, $\mathcal{F}$, as listed below. The abbreviation W stands for the web bulletin board, T is Tom, the poll worker, EA is the election authority, PS and PC are the POD service and client, respectively, and BM is the ballot manager. For convenience, names are abbreviated as follows: the set of candidates as $\mathcal{C}$, voters as $\mathcal{V}$, agents as $\mathcal{A}$, serial numbers as $\mathcal{S}$, nonces as $\mathcal{N}$, and public-keys and secret-keys as $\mathcal{PK}$ and $\mathcal{SK}$, respectively.

$$
\begin{aligned}
\mathcal{C} &= \{Zoe, Victor\} \\
\mathcal{V} &= \{Alice, Bob, James\} \\
\mathcal{A} &= \bigcup(\mathcal{V}, \{Tom, authority, wbb, teller, podservice, \\
&\qquad podclient, ballotmngr, ebm, printer\}) \\
\mathcal{S} &= \{s_1, s_2, s_3\} \\
\mathcal{N} &= \{n_a, n_b, n_c\} \\
\mathcal{PK} &= \{pk_A \mid A \in \{W, T, EA, PS, PC, BM\}\} \\
\mathcal{SK} &= \{sk_A \mid A \in \{W, T, EA, PS, PC, BM\}\}
\end{aligned}
$$

The agents send various kinds of messages to each other, which need to be defined in terms of data-types. The messages mentioned above and illustrated in Figure 7.3 form the message set $\mathcal{M}$. The names of the sets are indicative of what messages they represent. However, to remove the ambiguity; *castrhs* represents the cast ballots, $\mathcal{L}$ is the set of all possible candidate lists, and $\mathcal{I}$ is the set of indices with how the voter is modelled to fill in the marking boxes for her preferred candidate.

In the next section, the communication channels, on which these messages are transmitted, are described.

7.2.2 Channel Types

As in the NSPK analysis in Section 3.3, the channels have the form $\mathcal{A}.\mathcal{A}.\mathcal{M}$, where $\mathcal{A}$ is the set of agents and $\mathcal{M}$ is the set of messages that agents may wish to transmit over the channels and these are listed in Figure 7.3.

The framework introduced in Section 3.3 involves only (InS) *Insecure channels*, i.e., the whole network is not secure, and hence, any message can be manipulated in many ways by the intruder. The intruder can block, overhear and spoof any message transmitted on the insecure communication channels between the legitimate agents. This kind of communication channels are directly connected to the intruder using the renaming operator in CSP. Hence, there is no restriction in the intruder process about what he can or cannot perform on the insecure communication channels. In order words, he can act as the Dolev-Yao intruder model on such channels.

$$
\begin{array}{lll}
\mathit{signednonces} & = \{\mathsf{S}_{sk}(n) & \mid sk \leftarrow \mathcal{SK},\\
 & & \;\; n \leftarrow \mathcal{N}\}\\
\mathit{signednonsers} & = \{\mathsf{S}_{sk}(s,n) & \mid sk \leftarrow \mathcal{SK},\\
 & & \;\; s \leftarrow \mathcal{S},\\
 & & \;\; n \leftarrow \mathcal{N}\}\\
\mathit{rawballots} & = \{\mathsf{Raw}(s,\mathsf{E}_{pk}(l)) & \mid s \leftarrow \mathcal{S},\\
 & & \;\; pk \leftarrow \mathcal{PK},\\
 & & \;\; l \leftarrow \mathcal{L}\}\\
\mathit{digitalballots} & = \{\mathsf{DigB}(\mathsf{S}_{sk}(s),\mathsf{E}_{pk}(l)) & \mid sk \leftarrow \mathcal{SK},\\
 & & \;\; pk \leftarrow \mathcal{PK}\\
 & & \;\; s \leftarrow \mathcal{S},\\
 & & \;\; l \leftarrow \mathcal{L}\}\\
\mathit{indices} & = \{\mathsf{Ind}.i & \mid i \leftarrow \mathit{Int}\}\\
\mathit{ballotforms} & = \{\mathsf{B}(l,s,\mathsf{Ind}.i) & \mid l \leftarrow \mathcal{L},\\
 & & \;\; s \leftarrow \mathcal{S},\\
 & & \;\; \mathsf{Ind}.i \leftarrow \mathcal{I},\\
 & & \;\; a \leftarrow \mathcal{A}\}\\
\mathit{castrhs} & = \{\mathsf{RHS}(s,\mathsf{Ind}.i) & \mid s \leftarrow \mathcal{S},\\
 & & \;\; \mathsf{Ind}.i \leftarrow \mathcal{I}\}\\
\mathit{receipts} & = \{\mathsf{R}(\mathsf{S}_{sk}(\mathsf{RHS}(s,\mathsf{Ind}.i))) & \mid s \leftarrow \mathcal{S},\\
 & & \;\; \mathsf{Ind}.i \leftarrow \mathcal{I},\\
 & & \;\; sk \leftarrow \mathcal{SK}\}\\
\mathit{votes} & = \{\mathsf{V}(\mathsf{Ind}.i,\mathsf{E}_{pk}(l)) & \mid \mathsf{Ind}.i \leftarrow \mathcal{I},\\
 & & \;\; pk \leftarrow \mathcal{PK},\\
 & & \;\; l \leftarrow \mathcal{L}\}\\
\mathit{atomicfacts} & = \{f & \mid f \leftarrow \cup\{\mathcal{V},\mathcal{N},\mathcal{I}\}\}
\end{array}
$$

Figure 7.3: Message types used in the modelling

Such an assumption is too strong for voting systems that require an environment for the voters to be able to vote privately, such as a voting booth, at least if the action of receiving a ballot form is modelled as a message. This is also the case for most of the remote voting systems, where it is assumed that no one is watching over the voters' shoulder while she is casting her vote. Hence, this necessitates the existence of private channels in the voting system model. To this end, the agents in the model are enabled to communicate over a *secure channel* (S), called *scomm*, on which the intruder has no power at all. The intruder cannot block, overhear or spoof transmitted messages over the secure communication channels. For instance, when the voter is given the ballot form by the poll worker, messages including the sensitive data regarding the candidate order, are transmitted over *scomm* channels. If there was no private channels, the intruder would obviously violate the voter's privacy by overhearing communication channels on which vulnerable data flows. In the modelling of such channels, different channel

names, like *scomm*, are used to distinguish the secure channels from others in order to hide the crucial information from the Dolev-Yao intruder. As stated previously, the intruder's ability is modelled using the renaming operator in the process definition of the intruder. This is to say that the intruder can perform all his bad behaviour on the channels that are connected to his process definition using renaming. Hence, the secure channel *scomm* is shared only between honest agents, and not with the intruder. Therefore, as the intruder does not have any connection with secure channels, he cannot block, overhear or spoof messages on such channels.

From the observations made throughout the analysis in this thesis (it will be explained further in Section 7.6), it is assumed that at least two eligible honest voters are able to vote, and the cast votes are tallied at the end of the election. Otherwise an attack regarding the voter's privacy occurs in which the intruder blocks all the communication channels except the one on which the target voter communicates in order to cast her vote. Thus, the intruder would learn how the voter has voted. Therefore, at least two honest voters should be able to cast their votes without any blocking so that the intruder cannot deduce how each of them has voted. This assumption requires that there exists a channel in the voting system model such that the communications made by these two honest voters with the other agents are *No Spoofing and Blocking* (NSB) channels modelled as *nsbcomm* here, and they are combinations of two different channel types; No Blocking (NB) and No Spoofing (NS) channels. On such channels the intruder can overhear the communication, but cannot block its occurrence and spoof any messages. Creese *et al.* [CGH$^+$05] describe various kinds of channels for pervasive computing environments. For instance, the *No OverHearing* channel c (NOH_c) is that which cannot be overheard, the *No Blocking* channel c (NB_c) is the channel that cannot be blocked and the *No Spoofing* channel c (NS_c) is the channel type that cannot be spoofed. The three NOH_c, NB_c and NS_c form the secure channels *scomm* in the modelling. The NB channels in CSP are modelled when the intruder process is renamed to take/block messages from the channels on the network.

Using CSP the set of messages that make sense to the protocol (they are from real communications between agents), called *comms*, can be defined as the union of sets of data objects for each message type. For instance, the following defines the vote messages sent by one agent to another.

$$commVotes = \{a.b.m \mid m \leftarrow votes, a \leftarrow \mathcal{A}, b \leftarrow \mathcal{A}, a \neq b\}$$

These are also useful when the intruder is afforded the ability to modify the messages on the insecure channels or not to block and fake certain data from specific agents as it may be confusing as to whether the message is already known or

has just been learned from the real communication that the intruder overhears. This is used in modelling the intruder by defining the set of legitimate insecure messages sent from one agent to another *Ucomms*, i.e., they are from real communications of agents, which can be overheard, learned and said by the intruder. This set on which the intruder can behave as the Dolev-Yao intruder in the analysis of vVote in Section 7.2.5 is defined as the following.

$$Ucomms = \bigcup(\{q.q'.f \mid q.q'.f \leftarrow comms, q \leftarrow \{James\}, q' \leftarrow agents\}, \\ \{q.q'.f \mid q.q'.f \leftarrow comms, q \leftarrow agents, q' \leftarrow \{James\}\})$$

The set defines real communication messages between the agent James and other agents on the network, meaning that all messages that are sent by and to James are transmitted on the insecure communication channels (InS). Similarly, insecure NB messages *Nbcomms* from real communication can be defined so using such sets and later used to determine what the intruder can overhear, spoof but not block.

Although, the existence of NB channels solves one problem, which is the unwanted privacy attack previously mentioned, there is another plausible attack where the intruder does not block the messages on NB channels, but can later modify and spoof the messages, i.e., the intruder cannot take/block, but he can still fake messages overheard from the NB channels. Hence, if the intruder can modify and spoof one of the messages sent from one of those honest two voters, he can then deduce the other private message by looking at the election result as in the previous attack. Therefore, there is a need for a channel that cannot be spoofed, called *No Spoofing* (NS) channels. On NS channels, the intruder can overhear but cannot block or spoof messages. This is exactly what we need in order to allow two honest voters to cast their votes without any interruption and modification. As such a channel is no blocking and spoofing channel, it will be called as *No spoofing and blocking* (NSB) channel from now on, and in the CSP definitions of the voting system and intruder model it will be expressed as *nsbcomm*.

As mentioned earlier, secure channels are combinations of NB, NS and NOH channels. NOH channels are the channels that the intruder cannot overhear any messages on. On such channels the intruder can block and spoof messages, but cannot overhear the communication channel. The implementation of this channel in CSP is similar to the others'—it is modelled by restricting what the intruder can overhear with a defined set of network messages.

For the analysis of vVote later in this thesis, we need to define what information flows over: secure channels *scomm*, insecure channels *comm*, and no spoofing and blocking channels *nsbcomm*, because of the reasons explained previously. This can be done in two ways: the first one is that all agents on the network

work on insecure communication channels (*comm*) in which case secure, and no spoofing and blocking channels need to be defined. The second way is that all agents communicate over a NSB channel (*nsbcomm*), and the secure and insecure channels are defined accordingly. As we know what information should be shared with the intruder, it is easier to define the insecure communications than defining the others, meaning that the second way of defining channels is the one to follow for the ease of modelling. This will also help to reduce the size of the required state space for automated analysis. In terms of the deduction system that is used in the model, following the second way does not have any impact on the deductions that may be made by the intruder because the same set of information is given to the intruder and the deduction system remains the same in each case. Table 7.1 illustrates the intruder's capabilities on different channels used in this analysis.

	Secure (S)	No OverHearing (NOH)	No Spoofing and Blocking (NSB)	Insecure (InS)
overhear	X	X	✓	✓
block	X	✓	X	✓
spoof	X	✓	X	✓

Table 7.1: The intruder's capabilities on different channels

Finally, there exist a number of other channels that regulate the protocol run, such as; *openElection*, *closeElection*, *enterBooth*, *leaveBooth*, *bagempty* and *done*. However these will not be discussed any further.

7.2.3 Modelling Assumptions

Although the aim in the modelling of voting systems is to obtain a model that reflects real system behaviour, there are a few assumptions that need to be made in order to avoid state explosion, which also result in abstractions in some of the features of the vVote voting system. For instance, although vVote supports the AV and STV electoral methods, FPTP will be modelled due to its simplicity in this analysis. Thus, possible privacy attacks to the system that may occur in the AV and STV electoral methods are not considered here. Additionally, in the original vVote system, ballot generation is made in a distributed fashion, which allows verifiable generation of ballot forms by distributing the trust among various entities. However, in the modelling of vVote, it is assumed that there is one honest single entity, election authority, who generates the candidate lists and digital ballot papers. This assumption can also be read as the entities that are responsible for distributed generation of ballot forms are honest and work as a single process. Similarly the web bulletin board (WBB) is a threshold-based

service, which signs messages by co-operation. As in all threshold parties in the voting system models in this thesis, the threshold parties in the vVote model are treated as single entities too.

The vVote voting system uses a mixnet to shuffle the encrypted votes cast during the election as in Prêt à Voter. Previously, in Section 6.2.2, a CSP model of the mixnet has been given, which works as a perfect mixnet (no link between its inputs and outputs due to its non-deterministic behaviour). However, here in vVote modelling we omit this mixnet process as the WBB process already outputs the encrypted messages non-deterministically to the decryption tellers. This can also be thought as that the mixnet process is embedded in the WBB process, removing the communication between a WBB process and a mixnet one. Thus, there is no point of having two subsequent non-deterministic choices over the same inputs in terms of efficient and effective modelling. Regarding the analysis of this voting system model without a mixnet process, as the communication channels between WBB and mixnet is no blocking link because of the reasons given in Section 7.2.2 and the messages are encrypted under authorities public key, there is not much that the intruder can do over these channels. Additionally, everything that the intruder can perform over the channel from the mixnet to the WBB can also be realised over the channel from the WBB to the decryption tellers because the messages and channel types are of the same format.

The vVote voting system employs a district information for each voter in order to allow them to vote in different constituencies. Because the modelling and analysis of this voting system does not cover this aspect of voting, the district information, used in the POD protocol, is omitted. Hence, with this abstraction, the possible privacy-related attacks to the system that may emerge with if the district information was used are not touched here.

Finally, the assumptions made in Section 5.2.1 regarding the voter behaviour when choosing the candidate to vote for and the number of booths are also valid for this modelling too. Similar to the assumption made on the number of booths, it is assumed here that there exists only one poll worker, which opens a session for each voter with a fresh nonce. This would not impact our analysis as in the case of existence of multiple poll workers in a polling station, voters could only authenticate themselves without awaiting each other with different poll workers. However, if the cast votes were to be published on the BB one by one in the model, then there might have been issues regarding this assumption. This is because in the current model voters cast their vote in order and if the intruder could see the cast votes published on the BB in the same order, then he could violate voter anonymity.

The following subsection presents the honest participants' CSP definitions.

7.2.4 Honest Participants

The vVote voting system model developed for this work is defined by the processes illustrated at the top of Figure 7.4. All the processes are involved in the protocol by sending, receiving messages on the synchronised channels and the model behaves exactly as in Figure 7.4. Moreover, the model covers all phases of the vVote, including the POD protocol. The following subsections present the CSP descriptions of the individual protocol participants.

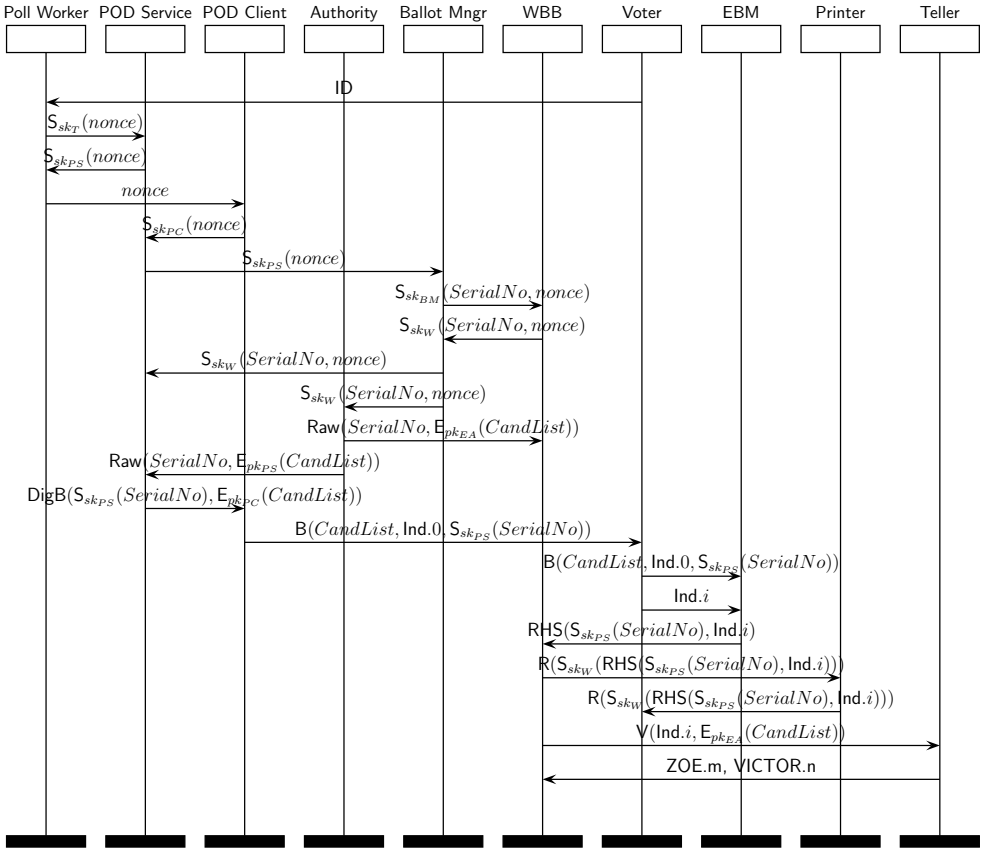

Figure 7.4: vVote system model

Voter Process

With the parameterised process $Voter(v, c)$, the behaviour of a voter $v \in \mathcal{V}$ voting for a chosen candidate $c \in \mathcal{C}$ is modelled. There exist two honest voters, Alice and Bob, and a misbehaving one, James, who behaves honestly in the model at first, but his secret will be shared with the intruder later on and whose communications, even the private and NSB ones, are used by the intruder.

Having authenticated herself on the NSB channel with the poll worker, Tom, the

voter receives a ballot form from the POD client with the candidate list printed on it and scans her ballot data to the EBM on the secure channel, where she can see her ballot in an electronic environment. After indicating her preference by sending the index value ($\mathsf{Ind}.i$) to the EBM that corresponds to the candidate she wants to vote for—the index i is found by using the function $\mathsf{find}(c, l)$, which finds the candidate c in the sequence of candidates l and is defined in Section 6.2—, she then receives her signed receipt and leaves the polling station.

$$
\begin{array}{l}
Voter(v, c) \mathrel{\widehat{=}} \\
\quad openElection \rightarrow nsbcomm.v.Tom.v \rightarrow \\
\quad \Box\limits_{\substack{l \in \mathcal{L} \\ s \in \mathcal{S}}} \left(\begin{array}{l}
scomm.podclient.v.\mathsf{B}(l, \mathsf{S}_{sk_{PS}}(s), \mathsf{Ind}.0) \rightarrow \\
scomm.v.ebm.\mathsf{B}(l, \mathsf{S}_{sk_{PS}}(s), \mathsf{Ind}.0) \rightarrow \\
\quad \Box\limits_{i := \mathsf{find}(c,l)} \left(\begin{array}{l}
nsbcomm.v.ebm.\mathsf{Ind}.i \rightarrow \\
nsbcomm.printer.v.\mathsf{R}(\mathsf{S}_{sk_W}(\mathsf{RHS}(\mathsf{S}_{sk_{PS}}(s), \mathsf{Ind}.i))) \rightarrow \\
closeElection \rightarrow STOP
\end{array} \right)
\end{array} \right)
\end{array}
$$

All eligible voters, Alice, Bob and James, follow this protocol, which is modelled as the parallel running of all individual voter processes synchronising on *openElection* and *closeElection* pairwise with the election authority. That is, each voter performs an *openElection* event to begin her voting process. Each voter must also perform a *closeElection* event after casting their individual vote and leaving the polling station.

$$Voters \mathrel{\widehat{=}} \|_{v,c} Voter(v, c)$$

Poll Worker Process

The poll worker, Tom, authenticates voters and starts a fresh session for each of them by choosing a nonce n from the set of nonces $\mathcal{N}$. He ensures that he always authenticates a different voter, and commences a new session with a fresh nonce. The poll worker is not involved in any private communication as he only sends and receives signed nonces from and to the POD service and sends nonces to the POD client.

$$
\begin{array}{l}
Pollworker(\mathcal{V}, \mathcal{N}) \mathrel{\widehat{=}} \\
\quad closeElection \rightarrow STOP \\
\quad \Box \\
\quad \Box\limits_{v \in \mathcal{V}} \left(\begin{array}{l}
nsbcomm.v.Tom.v \rightarrow \\
\quad \bigsqcap\limits_{n \in \mathcal{N}} \left(\begin{array}{l}
nsbcomm.Tom.podservice.\mathsf{S}_{sk_T}(n) \rightarrow \\
nsbcomm.podservice.Tom.\mathsf{S}_{sk_{PS}}(n) \rightarrow \\
nsbcomm.Tom.podclient.n \rightarrow \\
Pollworker(\mathcal{V} \setminus \{v\}, \mathcal{N} \setminus \{n\})
\end{array} \right)
\end{array} \right)
\end{array}
$$

Election Authority Process

Authority is the election authority process, which assigns a random candidate list from the list $\mathcal{L}$ for a particular serial number that has been asked for along with a nonce from the ballot manager. Following this, *Authority* submits two copies of the raw ballot form: one is encrypted under the election authority's public key pk_{EA} and sent to the WBB, the other is encrypted under the POD service public key pk_{PS} and sent to the POD service. Hence, the WBB and the POD service keep the same candidate list associated for a particular serial number, but encrypted under different keys.

$$
\begin{array}{l}
Authority \;\widehat{=}\; openElection \rightarrow Authority_1(\mathcal{S},\mathcal{L}) \\
Authority_1(\emptyset,\mathcal{L}) \;\widehat{=}\; closeElection \rightarrow STOP \\
Authority_1(\mathcal{S},\mathcal{L}) \;\widehat{=} \\
\quad \Box \\
\quad \Box_{\substack{s\in\mathcal{S}\\ n\in\mathcal{N}}} \left(\begin{array}{l} nsbcomm.ballotmngr.authority.\mathsf{S}_{sk_{BM}}(s,n) \rightarrow \\ \quad \bigsqcap_{l\in\mathcal{L}} \left(\begin{array}{l} nsbcomm.authority.wbb.\mathsf{Raw}(s,\mathsf{E}_{pk_{EA}}(l)) \\ nsbcomm.authority.podservice.\mathsf{Raw}(s,\mathsf{E}_{pk_{PS}}(l)) \\ Authority_1(\mathcal{S}\setminus\{s\},\mathcal{L}) \end{array} \right) \end{array} \right)
\end{array}
$$

POD Service Process

Following a fresh session, the POD service (candidate list key sharers) receives a serial number s from the ballot manager and the encrypted candidate list $\mathsf{E}_{pk_{PS}}(l)$ associated with s from the election authority, which is called a raw ballot. Subsequently, the digital ballot form consisting of a signed serial number and the encrypted candidate list is sent to the POD client after a transformation made on the encrypted candidate list from the POD service's public key pk_{PS} to POD client's public key pk_{PC}.

$$
\begin{array}{l}
Podservice \;\widehat{=} \\
\quad closeElection \rightarrow STOP \\
\quad \Box \\
\quad \Box_{n\in\mathcal{N}} \left(\begin{array}{l} nsbcomm.Tom.podservice.\mathsf{S}_{sk_T}(n) \rightarrow \\ nsbcomm.podservice.Tom.\mathsf{S}_{sk_{PS}}(n) \rightarrow \\ nsbcomm.podclient.podservice.\mathsf{S}_{sk_{PC}}(n) \rightarrow \\ nsbcomm.podservice.ballotmngr.\mathsf{S}_{sk_{PS}}(n) \rightarrow \\ \quad \Box_{s\in\mathcal{S}} \left(\begin{array}{l} nsbcomm.ballotmngr.podservice.\mathsf{S}_{sk_W}(s,n) \rightarrow \\ \quad \Box_{l\in\mathcal{L}} \left(\begin{array}{l} nsbcomm.authority.podservice. \\ \qquad \mathsf{Raw}(s,\mathsf{E}_{pk_{PS}}(l)) \\ nsbcomm.podservice.podclient. \\ \qquad \mathsf{DigB}(\mathsf{S}_{sk_{PS}}(s),\mathsf{E}_{pk_{PC}}(l)) \rightarrow \\ Podservice \end{array} \right) \end{array} \right) \end{array} \right)
\end{array}
$$

POD Client Process

The POD Client process is responsible for printing out the ballot form, which has been received as a digital ballot from the POD service (note that this should not be confused with the receipt printer). The candidate list on this digital ballot l is encrypted under the POD client's public key pk_{PC} with empty marking boxes denoted by $\mathsf{Ind}.0$. Having extracted the candidate list, the POD client prints the actual ballot form for the voter on the private channel.

$$
\begin{array}{l}
Podclient \mathrel{\widehat{=}} \\
\quad closeElection \rightarrow STOP \\
\quad \Box \\
\quad \displaystyle\Box_{n \in \mathcal{N}} \left(\begin{array}{l} nsbcomm.Tom.podclient.n \rightarrow \\ nsbcomm.podclient.podservice.\mathsf{S}_{sk_{PC}}(n) \rightarrow \\ \displaystyle\Box_{\substack{s \in \mathcal{S} \\ l \in \mathcal{L}}} \left(\begin{array}{l} nsbcomm.podservice.podclient. \\ \qquad \mathsf{DigB}(\mathsf{S}_{sk_{PS}}(s), \mathsf{E}_{pk_{PC}}(l)) \rightarrow \\ \displaystyle\bigsqcap_{v \in \mathcal{V}} \left(\begin{array}{l} scomm.podclient.v.\mathsf{B}(l, \mathsf{S}_{sk_{PS}}(s), \mathsf{Ind}.0) \rightarrow \\ Podclient \end{array} \right) \end{array} \right) \end{array} \right)
\end{array}
$$

Ballot Manager Process

The ballot manager apportions the serial numbers to each ballot form uniquely and commits them to the WBB. Additionally, it also notifies the election authority and the POD service about the serial number being used.

$$
\begin{array}{l}
Ballotmanager \quad \mathrel{\widehat{=}} openElection \rightarrow Ballotmanager(\mathcal{S}) \\
Ballotmanager(\emptyset) \mathrel{\widehat{=}} closeElection \rightarrow STOP \\
Ballotmanager(\mathcal{S}) \mathrel{\widehat{=}} \\
\quad closeElection \rightarrow STOP \\
\quad \Box \\
\quad \displaystyle\Box_{n \in \mathcal{N}} \left(\begin{array}{l} nsbcomm.podservice.ballotmngr.\mathsf{S}_{sk_{PS}}(n) \rightarrow \\ \displaystyle\bigsqcap_{s \in \mathcal{S}} \left(\begin{array}{l} nsbcomm.ballotmngr.wbb.\mathsf{S}_{sk_{BM}}(s,n) \rightarrow \\ nsbcomm.ballotmngr.authority.\mathsf{S}_{sk_{BM}}(s,n) \rightarrow \\ nsbcomm.wbb.ballotmngr.\mathsf{S}_{sk_{W}}(s,n) \rightarrow \\ nsbcomm.ballotmngr.podservice.\mathsf{S}_{sk_{W}}(s,n) \rightarrow \\ Ballotmanager(\mathcal{S} \setminus \{s\}) \end{array} \right) \end{array} \right)
\end{array}
$$

The Electronic Ballot Marker (EBM)

The EBM is a device to help voters to mark their preferences. Having received her ballot form from the POD client, the voter goes into the booth and scans her ballot form to transfer the ballot information to the EBM on the secure channel *scomm*. She then fills out the electronic ballot form on the screen by interacting

with the machine and choosing the index value $\mathsf{Ind}.i$ corresponding to her chosen candidate. Although no one is supposed to be observing voter interaction with the EBM, it is assumed here that the index value sent from voter to the EBM can be observed by the intruder, as it will be observed anyway once she takes her receipt from the receipt printer. Afterwards, the EBM sends the marking boxes side of the ballot form, $\mathsf{RHS}(\mathsf{S}_{sk_{PS}}(s), \mathsf{Ind}.i)$, to the WBB, which can then be checked by the voters.

$$EBM \mathrel{\widehat{=}} closeElection \rightarrow STOP$$
$$\Box$$
$$\Box_{\substack{l\in\mathcal{L}\\ s\in\mathcal{S}\\ v\in\mathcal{V}}} \left(\begin{array}{l} scomm.v.ebm.\mathsf{B}(l, \mathsf{S}_{sk_{PS}}(s), \mathsf{Ind}.0) \rightarrow \\ \Box_{i\in\mathcal{I}} \left(\begin{array}{l} nsbcomm.v.ebm.\mathsf{Ind}.i \rightarrow \\ nsbcomm.ebm.wbb.\mathsf{RHS}(\mathsf{S}_{sk_{PS}}(s), \mathsf{Ind}.i) \rightarrow EBM \end{array} \right) \end{array} \right)$$

Receipt Printer Process

The receipt printer process behaves as a typical printer, i.e., it receives the receipt r from the WBB, and prints it out for the voter v. Note that the POD client and this printer process are two different printers located in different places in the polling station.

$$Printer \mathrel{\widehat{=}} closeElection \rightarrow STOP$$
$$\Box$$
$$\Box_{\substack{v\in\mathcal{V}\\ r\in receipts}} \left(\begin{array}{l} nsbcomm.wbb.printer.r \rightarrow \\ nsbcomm.printer.v.r \rightarrow Printer \end{array} \right)$$

The Web Bulletin Board Process

The WBB is a public bulletin board that broadcasts the committed data during the election, such as submitted votes and signed serial numbers. Moreover, there is nothing private about this process as everything is publicly verifiable. Having received all the cast votes and sending the receipts for each voter, the WBB transfers them in the form of $\mathsf{V}(\mathsf{Ind}.i, \mathsf{E}_{pk_{EA}}(l))$ to the decryption teller (election key sharers) non-deterministically. Note that a mixnet, such as re-encryption mixnet used in Prêt à Voter, shuffles the cast votes arbitrarily and here a separate mixnet processes is not necessary as the WBB already outputs the cast votes non-deterministically. Hence, it can be thought of there being a perfect mixnet embedded in the WBB and this also helps in terms of the state space. Having finished the tallying, the teller then sends the result for each candidate (Zoe and Victor) to the WBB.

$$
\begin{aligned}
&WBB \quad \widehat{=} \; openElection \rightarrow WBB_1(\emptyset)\\
&WBB_1(bag) \; \widehat{=}\\
&\quad closeElection \rightarrow WBB_2(bag)\\
&\quad \Box\\
&\quad \Box_{\substack{n\in\mathcal{N}\\ s\in\mathcal{S}}} \left(\begin{array}{l} nsbcomm.ballotmngr.wbb.\mathsf{S}_{sk_{BM}}(s,n) \rightarrow \\ nsbcomm.wbb.ballotmngr.\mathsf{S}_{sk_W}(s,n) \rightarrow \\ \quad \Box_{l\in\mathcal{L}} \left(\begin{array}{l} nsbcomm.authority.wbb.\mathsf{Raw}(s, \mathsf{E}_{pk_{EA}}(l)) \\ \quad \Box_{i\in\mathcal{I}} \left(\begin{array}{l} nsbcomm.ebm.wbb.\mathsf{RHS}(\mathsf{S}_{sk_{PS}}(s), \mathsf{Ind}.i) \rightarrow \\ nsbcomm.wbb.printer. \\ \qquad \mathsf{S}_{sk_W}(\mathsf{RHS}(\mathsf{S}_{sk_{PS}}(s), \mathsf{Ind}.i)) \rightarrow \\ WBB_1(bag \cup \{\mathsf{V}(\mathsf{Ind}.i, \mathsf{E}_{pk_{EA}}(l))\}) \end{array} \right) \end{array} \right) \end{array} \right)
\end{aligned}
$$

$$
\begin{aligned}
&WBB_2(\emptyset) \quad \widehat{=} \; bagempty \rightarrow nsbcomm.Zoe?t_1 \rightarrow \\
&\qquad\qquad nsbcomm.Victor?t_2 \rightarrow done \rightarrow STOP\\
&WBB_2(bag) \; \widehat{=} \quad \sqcap_{v\in votes} \; nsbcomm.wbb.teller.v \rightarrow WBB_2(bag \setminus \{v\})
\end{aligned}
$$

Decryption Teller Process

The decryption teller process—it is a thresholded setup, called decryption key sharers, but here this property is abstracted away and modelled as a single CSP process—is responsible for decrypting the votes encrypted under the election authority's public key pk_{EA} and tallying them for each candidate. The results are then sent back to the WBB. What happens in the third line of the process is that because the decryption teller possesses the shared secret key sk_{EA} (shared in the real system), it can decrypt and extract the candidate list of the cast votes as in $l := \mathsf{D}_{sk_{EA}}(\mathsf{E}_{pk_{EA}}(l))$. The teller then identifies for whom the vote is by checking the ith element of the list l. Accordingly, it increments the total vote received by that particular candidate by one. Once there are no more votes to tally, the teller announces the total votes for each candidate.

$$
\begin{aligned}
&Teller \; \widehat{=} \; openElection \rightarrow Teller_1(0,0)\\
&Teller_1(m,n) \; \widehat{=}\\
&\quad \Box_{\substack{i\in\mathcal{I}\\ l\in\mathcal{L}}} \left(\begin{array}{l} nsbcomm.wbb.teller.\mathsf{V}(\mathsf{Ind}.i, \mathsf{E}_{pk_{EA}}(l)) \rightarrow \\ \left(\begin{array}{c} \text{if } nth(i, \mathsf{D}_{sk_{EA}}(\mathsf{E}_{pk_{EA}}(l))) = Zoe \text{ then } Teller_1(m+1, n) \\ \text{else} \left(\begin{array}{c} \text{if } nth(i, \mathsf{D}_{sk_{EA}}(\mathsf{E}_{pk_{EA}}(l))) = Victor \text{ then} \\ Teller_1(m, n+1) \text{ else } STOP \end{array} \right) \end{array} \right) \end{array} \right)\\
&\quad \Box\\
&\quad bagempty \rightarrow nsbcomm.Zoe.m \rightarrow nsbcomm.Victor.n \rightarrow SKIP
\end{aligned}
$$

7.2.5 Adapting Lazy Spy

Lazy spy [Ros97] is an efficient intruder model as it avoids state explosion by following only its findings (deductions through the messages he has seen or from his initial knowledge). This intruder model provides active attacks against the system by not only observing the communication channels, but also blocking messages or generating and sending fake messages to any agents on the system. The framework has already been introduced in Section 3.3 in detail. The framework should be altered so it can work with the cryptographic voting systems. In particular, the vVote voting system model is equipped with a number of voting system specific messages as well as the cryptographic ones (see Figure 7.3). Hence, the existence of these messages requires further deduction rules that need to be defined so that the intruder can act as he is supposed to regarding those messages. Secondly, the initial knowledge of the intruder $\mathcal{IK}$ is also model specific, hence, it needs to be defined according to the voting system model and as this set of knowledge is used to specify what the intruder knows and what he can learn, it needs to be defined carefully. Lastly, because of the introduction of various channel types in the analysis of voting systems, the intruder model needs to be amended so that the private channels stay private and NSB channels are, indeed, not blocked or spoofed by the intruder.

In order to allow the intruder to compose messages, there are a number of deduction rules. Recall that a deduction is a pair (X, f), where X is a finite set of facts and f is the fact that can be generated, providing that the intruder possesses X and these inferences are denoted as $X \vdash f$. It should be ensured that the intruder deals with a finite set of facts because FDR cannot handle with infinite number of states. Thus, nesting of encryptions and sequences need to be avoided. To do so, the set of data-types are limited to the types that are enough to build protocol messages. Although, the intruder can generate "bad" facts (objects that are not of the form real messages sent among protocol agents), these facts will do him no good [RSG+00]. That is because the agents in the protocol can only communicate with the messages that they understand—the messages need to be in the same form as they are expected. Hence, the deduction rules with which the intruder is able to build and decompose all protocol messages are adequate for the analysis.

The deduction rules regarding this analysis $\mathcal{D}$ are the union of deductions defined previously in Table 2.4, which are related to cryptographic primitives and in Table 7.2 are specific to the vVote voting system. The new deduction rule BALLOT-COMP enables ballot forms to be composed if the intruder possesses the set $\{l, \mathsf{S}_{sk}(s), \mathsf{Ind}.i\}$, where l is the candidate list, s is serial number and $\mathsf{Ind}.i$ is the index value, corresponding to the chosen candidate and conversely the deduction rule BALLOT-DCMP helps the intruder to decompose ballot forms

and obtain all the data on it. Similarly, the intruder can also work on any composition and decomposition of any other messages in the model. For instance, RHS-COMP and RHS-DCMP are the deduction rules related to cast ballot forms, consisting of an index value and a signed serial number $\{\mathsf{Ind}.i, \mathsf{S}_{sk}(s)\}$. VOTE-COMP and VOTE-DCMP are the deduction rules related to the votes, in the form of $\mathsf{V}(\mathsf{Ind}.i, \mathsf{E}_{pk}(l))$ (note that these do not contain a serial number). The deduction rules regarding the digital ballots, consisting of a signed serial number and an encrypted candidate list, $\{\mathsf{S}_{sk}(s), \mathsf{E}_{pk}(l)\}$, are DIG.BLT-COMP and DIG.BLT-DCMP. Similarly, RAW.BLT-COMP and RAW.BLT-DCMP are the two deduction rules that help the intruder compose and decompose the raw ballots,$\{s, \mathsf{E}_{pk}(l)\}$, and IND-COMP and IND-DCMP are the index related deduction rules. Hence, with this set of deduction rules, $\mathcal{D}$, the intruder is enabled to deduce messages that are used to attack the protocol.

BALLOT-COMP.	$\{l, \mathsf{S}_{sk}(s), \mathsf{Ind}.i\}$	$\vdash$	$\mathsf{B}(l, \mathsf{S}_{sk}(s), \mathsf{Ind}.i)$
BALLOT-DCMP.	$\{\mathsf{B}(l, \mathsf{S}_{sk}(s), \mathsf{Ind}.i)\}$	$\vdash$	$l, \mathsf{S}_{sk}(s), \mathsf{Ind}.i$
RHS-COMP.	$\{\mathsf{Ind}.i, \mathsf{S}_{sk}(s)\}$	$\vdash$	$\mathsf{RHS}(\mathsf{Ind}.i, \mathsf{S}_{sk}(s))$
RHS-DCMP.	$\{\mathsf{RHS}(\mathsf{S}_{sk}(s), \mathsf{Ind}.i)\}$	$\vdash$	$\mathsf{Ind}.i, \mathsf{S}_{sk}(s)$
VOTE-COMP.	$\{\mathsf{Ind}.i, \mathsf{E}_{pk}(l)\}$	$\vdash$	$\mathsf{V}(\mathsf{Ind}.i, \mathsf{E}_{pk}(l))$
VOTE-DCMP.	$\{\mathsf{V}(\mathsf{Ind}.i, \mathsf{E}_{pk}(l))\}$	$\vdash$	$\mathsf{Ind}.i, \mathsf{E}_{pk}(l)$
DIG.BLT-COMP.	$\{\mathsf{S}_{sk}(s), \mathsf{E}_{pk}(l)\}$	$\vdash$	$\mathsf{DigB}(\mathsf{S}_{sk}(s), \mathsf{E}_{pk}(l))$
DIG.BLT-DCMP.	$\{\mathsf{DigB}(\mathsf{S}_{sk}(s), \mathsf{E}_{pk}(l))\}$	$\vdash$	$\mathsf{S}_{sk}(s), \mathsf{E}_{pk}(l)$
RAW.BLT-COMP.	$\{s, \mathsf{E}_{pk}(l)\}$	$\vdash$	$\mathsf{Raw}(s, \mathsf{E}_{pk}(l))$
RAW.BLT-DCMP.	$\{\mathsf{Raw}(s, \mathsf{E}_{pk}(l))\}$	$\vdash$	$s, \mathsf{E}_{pk}(l)$
IND-COMP.	$\{i\}$	$\vdash$	$\mathsf{Ind}.i$
IND-DCMP.	$\{\mathsf{Ind}.i\}$	$\vdash$	i

Table 7.2: Deduction rules capturing the properties of vVote voting system messages

As mentioned earlier, the set *comms* needs to be defined for all messages in the model illustrated in Figure 7.3 so that the intruder can justify that a message being heard is actually from a real communication between agents. As in the protocol, no agent sends any message to himself, for such communications are ensured to be omitted with $a \neq b$ below, which also implies that if an agent sends a message to himself, it cannot be blocked or spoofed by the intruder.

$$comms = \{a.b.m \mid m \leftarrow \mathcal{M}, a \leftarrow \mathcal{A}, b \leftarrow \mathcal{A}, a \neq b\}$$

The messages in the model that make sense to the intruder are: *comms*, all the messages from real communications, *Nsbcomms*, the set of messages that cannot be blocked or spoofed by the intruder, and *Ucomms*, the set of insecure messages that the intruder can act as in the Dolev-Yao intruder model [DY83].

As all honest participants communicate on the NSB channels, not including a message type in the set *Nsbcomms* means that the intruder cannot even overhear that kind of message. Hence, the messages in the form of a ballot are not included in this set, as the intruder should not be able to observe any communication involving a ballot form between honest participants (denoted as *commBallots*). For example, a voter scanning her ballot form to the EBM should not be observed by the intruder and this is how he is prevented from overhearing and blocking the private channels.

$$Nsbcomms = comms \setminus commBallots$$

The insecure messages that the intruder can overhear, block or use in any way in the line of Dolev-Yao model, are defined with the set *Ucomms* as follows. It should be noted that the set in the analysis of vVote covers all the messages that are communicated by the dishonest voter James.

$$\begin{aligned} Ucomms = \bigcup(&\{q.q'.f \mid q.q'.f \leftarrow comms, q \leftarrow \{James\}, q' \leftarrow agents\}, \\ &\{q.q'.f \mid q.q'.f \leftarrow comms, q \leftarrow agents, q' \leftarrow \{James\}\}) \end{aligned}$$

The set *Ucomms* can be extended with any set of information. For instance, the insecure communications in *Ucomms* do not yet include the receipts taken by the voters during the election. Hence, the intruder cannot cannot block the voters' taking their receipts as they are still on the no blocking channel. However, if we add the set of receipts that can be taken away by any voters (denoted as *commReceipts*) to the set *Ucomms*, then the intruder could also block the voters taking their receipts because in such case receipt information would flow on the insecure communication channels. However, it should be noted that the more information is given to the intruder, the longer the automated verification takes due to the increased number of deductions made by the intruder.

The intruder also needs a way to identify the facts that are relevant to the messages, $\mathcal{M}$, and as in the NSPK analysis, those facts can be identified by using the *explode* function defined in Section 3.3. However, the function needs to be extended to all messages in the set of messages $\mathcal{M}$ as shown in *AllFacts* below.

$$AllFacts = \{explode(m) \mid m \leftarrow \mathcal{M}\}$$

The intruder's possible deductions now can be found by applying all these facts to the deductions function *deductions*(), which takes the message types as the parameter and returns the possible deductions according to the deduction rules displayed in Figure 7.2.

$$AllDeductions = deductions(Allfacts)$$

In order to reduce the cardinality of the set of possible deductions, the facts that cannot be reachable by the intruder are omitted, which depends on his initial knowledge $\mathcal{IK}$. This knowledge covers the names of the candidates, voters and any other agents, such as, the POD service, and the public keys that any honest agent may know $\mathcal{IK} = \bigcup(\mathcal{A}, \mathcal{C}, \{pk_a \mid a \leftarrow \mathcal{A}\})$. Subsequently, if the framework procedure as in Section 3.3 is followed, the learnable facts and then the deductions, $\mathcal{D}$, can be defined as follows:

$$\begin{aligned} PossibleBasicKnowledge &= Known \cup \mathcal{M} \\ KnowableFacts &= Close(PossibleBasicKnowledge) \\ Learnablefacts &= KnowableFacts \setminus Known \end{aligned}$$

$$\begin{aligned} \mathcal{D} = \{(X, f) \mid\ & (X, f) \in AllDeductions, \\ & f \in Learnablefacts, \\ & f \notin X, \\ & X \setminus Knowablefacts = \emptyset\} \end{aligned}$$

Now, the processes that form the intruder model can be defined. These processes use the channel *learn* to learn a fact, and the *say* channel to say the fact to the other agents. Moreover, the channel *infer* allows deductions between facts. Thus, the process *Ignorantof* below ensures that all facts in *Learnablefacts* have the state: the process can always learn a fact, but it can only say it once it is known. The process is the same as used in the NSPK analysis except for the line where it flags up when the intruder knows a secret (This will be discussed when studying the secrecy specification in Section 7.5).

$$\begin{aligned} Ignorantof(f) \mathrel{\widehat{=}} & f \in \mathcal{M} \,\&\, learn.f \rightarrow Knows(f) \\ & \Box\ infer?t \in \{(X, f') \mid (X, f') \in \mathcal{D}, f' = f\} \rightarrow Knows(f) \end{aligned}$$

$$\begin{aligned} Knows(f) \mathrel{\widehat{=}} & f \in \mathcal{M} \,\&\, say.f \rightarrow Knows(f) \\ & \Box\ f \in \mathcal{M} \,\&\, learn.f \rightarrow Knows(f) \\ & \Box\ infer?t \in \{(X, f') \mid (X, f') \in \mathcal{D}, f \in X\} \rightarrow Knows(f) \end{aligned}$$

Consequently, the intruder process can be defined as a parallel composition of facts, where $AlphaL(f)$ defines the alphabet of each fact and the internal events *infer* are hidden.

$$\begin{aligned} &Intruder \mathrel{\widehat{=}} \\ &\quad \mathsf{chase}((\underset{f \in Learnablefacts}{\|} (Ignorant(f), AlphaL(f)))\backslash\{|infer|\}) \\ &\qquad ||| \; SayKnown \end{aligned}$$

The process *SayKnown* defined below ensures that the proper messages that are already known are learned and said, since the facts that are of relevance to the initial knowledge $\mathcal{IK}$ are not included in the *Learnablefacts* and *AllDeductions*.

$$\begin{aligned} SayKnown \mathrel{\widehat{=}}\ & say.f \in Known \cap \mathcal{M} \rightarrow SayKnown \\ & \Box\ learn.f \in Known \cap \mathcal{M} \rightarrow SayKnown \end{aligned}$$

7.2.6 Putting the Network Together

Figure 7.5 illustrates how the intruder is connected to the dishonest voter James, and the honest voter Alice, whereby Alice's private channel *scomm* is kept private, but her insecure NSB channels can be observed by the intruder, whereas all the channels of James are under the control of the intruder. That is, the intruder can overhear all insecure communications acting as a medium, but he can only intercept and fake the messages in the form of insecure data *Ucomms* (it defines all communications from and to James) as defined in the previous section. Moreover, he has no power over the private channels of the honest voters.

The processes that construct the voting system model and the intruder model are connected by using the renaming operator. That is, *nsbcomm.a.b.m* and *learn* channels are renamed to a *take* channel, and the *nsbcomm.b.a.m* and *say* channels are renamed to a *fake* channel from the agent a's point of view. Similarly, the intruder process is also renamed and the aim is to connect them as is done in Figure 3.2. Hence, the intruder channel *learn.m* is mapped to the events of the form *take.a.b.m*, and *say.m* is renamed to *fake.a.b.m*. To this end, a renaming function for the process P and agent name p can be defined as follows:

$$\begin{aligned} r(P,p) \mathrel{\widehat{=}}\ & P[\![^{nsbcomm.p,\, take.p}/_{nsbcomm.p,\, nsbcomm.p}]\!] \\ & [\![^{nsbcomm.a.p,\, fake.a.p}/_{nsbcomm.a.p,\, nsbcomm.a.p \mid a \in \mathcal{A}}]\!] \\ & [\![^{scomm.p,\, take.p}/_{scomm.p,\, scomm.p}]\!] \\ & [\![^{scomm.a.p,\, fake.a.p}/_{scomm.a.p,\, scomm.a.p \mid a \in \mathcal{A}}]\!] \end{aligned}$$

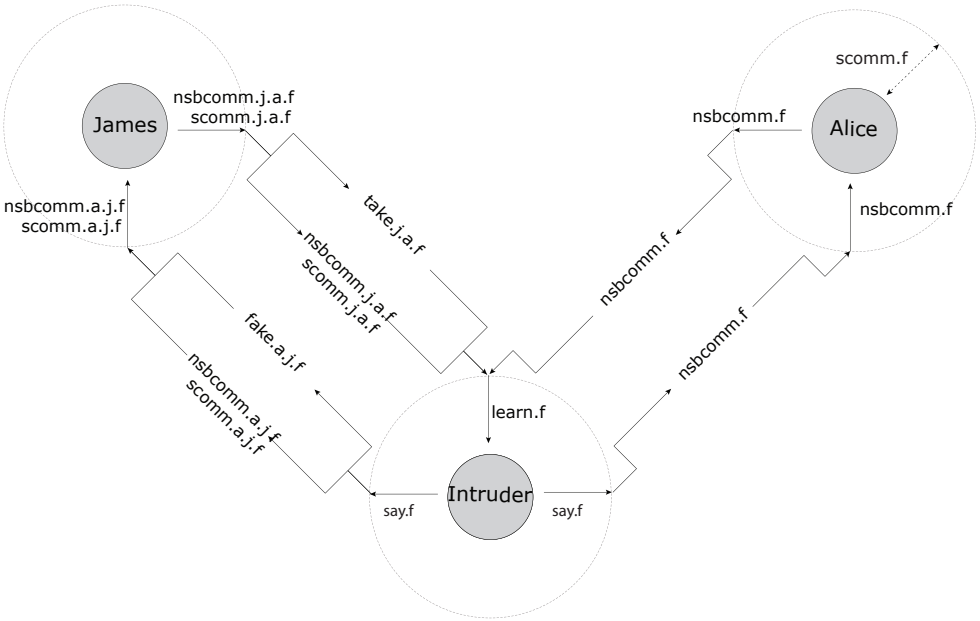

Figure 7.5: The lazy spy intruder model

Hence, the renamed voter process for the voter v, for instance, can be defined as follows. Note that, the private channel *scomm* is renamed to the *take* and $fake$ channels, because this models the malicious behaviour of a corrupt voter.

$$
\begin{aligned}
&r(\mathit{Voter}(v,c),v) \mathrel{\widehat{=}} \\
&\quad \mathit{Voter}(v,c) \\
&\qquad [\![{}^{nsbcomm.v,\, take.v}/_{nsbcomm.v,\, nsbcomm.v}]\!] \\
&\qquad [\![{}^{nsbcomm.a.v,\, fake.a.v}/_{nsbcomm.a.v,\, nsbcomm.a.v \mid a \in \mathcal{A}}]\!] \\
&\qquad [\![{}^{scomm.v,\, take.v}/_{scomm.v,\, scomm.v}]\!] \\
&\qquad [\![{}^{scomm.a.v,\, fake.a.v}/_{scomm.a.v,\, scomm.a.v \mid a \in \mathcal{A}}]\!]
\end{aligned}
$$

Similarly, the other processes that construct the vVote voting system model are renamed as in the above example. Consequently, the voting system model, *Model*, which is ready to be modified by the intruder, is defined as the parallel composition of all those renamed processes.

$$
\begin{aligned}
&\mathit{Model} \mathrel{\widehat{=}} \\
&\quad \mathit{rVoters} \parallel \mathit{rPollworker} \parallel \mathit{rAuthority} \parallel \mathit{rEBM} \parallel \mathit{rPodservice} \parallel \mathit{rPrinter} \\
&\quad \parallel \mathit{rBallotmanager} \parallel \mathit{rPodclient} \parallel \mathit{rWBB} \parallel \mathit{rTeller}
\end{aligned}
$$

The parallel composition (interface parallel) above is constructed in a way that the processes only synchronise on the *nsbcomm* and *scomm* channels on which

they send messages to each other, leaving the insecure channels ($take$, $fake$) vulnerable to be used by the intruder. The following shows how two processes are put in an interface parallel and therefore, the above parallel composition of *Model* should be constructed in this way.

$$Model \mathrel{\widehat{=}} rVoters \underset{X}{\parallel} rPollworker \parallel \ldots$$

where

$$X = \{| nsbcomm.v.Tom, nsbcomm.Tom.v, scomm.v.Tom, scomm.Tom.v \mid v \leftarrow \mathcal{V} |\}$$

Similarly, the intruder process is prepared by renaming as below so that the intruder can overhear the messages on the insecure NSB channels (*Nsbcomms*) and act as the Dolev-Yao intruder on the insecure channels (*Ucomms*). For the vVote analysis, these sets are defined in the previous section.

$$\begin{aligned} rIntruder \mathrel{\widehat{=}} & \\ & Intruder \\ & \quad [\![{}^{say, learn}/_{say, say}]\!] \\ & \quad [\![{}^{nsbcomm.p.p'.f,\, take.q.q'.f}/_{learn.f,\, learn.f \mid \substack{p.p'.f \in Nsbcomms,\\ q.q'.f \in Ucomms,\\ p \neq p', q \neq q'}}]\!] \\ & \quad [\![{}^{fake.p.p'.f}/_{say.f \mid p.p'.f \in Ucomms,\ p \neq p'}]\!] \end{aligned}$$

The process $System_{vVote}$ is then defined in terms of the parallel composition of *Model* and *rIntruder*, which synchronise on the channels they share.

$$System_{vVote} \mathrel{\widehat{=}} Model \underset{\{| nsbcomm, take, fake |\}}{\parallel} rIntruder$$

Having modelled the system, it is ready to be automatically analysed under the anonymity requirement in the next section.

7.3 Automated Anonymity Verification

In this section, the first fully-automated analysis of the vVote voting system is presented under a Dolev-Yao intruder model and using the anonymity definition given in Definition 2 as the specification. It requires that when the two channels $c.x$ and $d.x$ are swapped over for all values of x, if the resulting process is

indistinguishable from the original one, then the process provides anonymity. In the previous section, the system where the honest agents of the voting system model and the intruder interact has been modelled as the process $System_{vVote}$. In this system, the process $rVoters$ is defined as the parallel composition of the $rVoter()$ processes for each of the voters: Alice, Bob and James. However, the anonymity specification in this chapter is approached in a different way whereby the two systems, which are expected to be indistinguishable, are defined as two separate system behaviours without using the renaming operator. Hence, the following processes, namely, $rVoters_1$ and $rVoters_2$ model the two different voters' behaviour, which may result in two different system behaviours: on the one hand Alice votes for Zoe, Bob votes for Victor and James can vote for either Zoe or Victor, whereas on the other hand, Alice votes for Victor, Bob votes for Zoe and again James can vote for any of them. The resulting two different system behaviours are ${System'}_{vVote}$ and ${System''}_{vVote}$, respectively. Note that the misbehaving voter, James, shares his knowledge with the intruder, thus his behaviour is modelled with the renamed voter process $rVoter()$, which allows the intruder to use his knowledge, whereas the honest voters Alice and Bob are modelled using the honest voter model $Voter()$. However, the NSB channels can still be observed by the intruder, meaning that the intruder acts passively on these channels.

$$\begin{aligned} rVoters_1 \mathrel{\widehat{=}} & (Voter(Alice, Zoe) \;|||\; Voter(Bob, Victor)) \\ & ||| \; (rVoter(James, Zoe) \sqcap rVoter(James, Victor)) \end{aligned}$$

$$\begin{aligned} rVoters_2 \mathrel{\widehat{=}} & (Voter(Alice, Victor) \;|||\; Voter(Bob, Zoe)) \\ & ||| \; (rVoter(James, Zoe) \sqcap rVoter(James, Victor)) \end{aligned}$$

The systems are modelled in such a way that the intruder can see everything James does, including his private messages, whilst Bob and Alice can vote freely without any interception/blocking or spoofing. That is, although the intruder can still overhear the public channels and inference from those messages, Alice and Bob vote under the private and NSB channel assumptions.

As in the Prêt à Voter analysis, the observational equivalence that is used for the analysis necessitates masking of the encrypted values in order to avoid false positive attacks, as the intruder can distinguish two ciphertexts, even if he does not know the secret key—this was not a case in the NSPK [NS78] analysis as the secrecy specification is different from the anonymity used in this analysis. To this end, a masking function maskFact is deployed. The function renames all messages encrypted under a public key, whose corresponding secret key is not known by the voter, to a data *ciphertext* and if the secret key is in the intruder's initial knowledge, then he is allowed to differentiate two ciphertexts by not masking them.

$$\mathsf{maskFact}(\mathsf{E}_{pk}(m)) = \\ \quad \text{if } \mathsf{dual}(pk) \in \mathcal{IK} \text{ then } \mathsf{E}_{pk}(m) \text{ else } ciphertext$$

The masking function $\mathsf{mask}(P)$ can also be defined for the processes, which masks all encrypted facts of a given process, P, using the $\mathsf{maskFact}$ function for all the data that appears in this process. (No keys are ever sent over the network, so if the intruder does not know a secret key at the beginning, he will not learn it later.)

$$\begin{aligned} &mask(P) \mathrel{\widehat{=}} \\ &\quad P[\![achannel.a.a'.\mathsf{DigB}(\mathsf{S}_{sk}(s), \mathsf{maskFact}(\mathsf{E}_{pk}(l)))/achannel.a.a'.\mathsf{DigB}(\mathsf{S}_{sk}(s), \mathsf{E}_{pk}(l))]\!] \\ &\qquad [\![achannel.a.a'.\mathsf{Raw}(s, \mathsf{maskFact}(\mathsf{E}_{pk}(l)))/achannel.a.a'.\mathsf{Raw}(s, \mathsf{E}_{pk}(l))]\!] \\ &\qquad [\![achannel.a.a'.\mathsf{V}(\mathsf{Ind}.i, \mathsf{maskFact}(\mathsf{E}_{pk}(l)))/achannel.a.a'.\mathsf{V}(\mathsf{Ind}.i, \mathsf{E}_{pk}(l))]\!] \end{aligned}$$

where $achannel \in \{nsbcomm, take, fake\}$, the serial number $s \in \mathcal{S}$, the candidate list $l \in \mathcal{L}$ and the index value $i \in \mathcal{I}$.

After applying the masking function to both $System'_{vVote}$ and $System''_{vVote}$, they are ready for the analysis under the anonymity specification. To this end, the anonymity requirement of this voting system model is checked with the following trace equivalence in which the private channels are hidden.

$$\mathsf{mask}(System'_{vVote}) \setminus \{|\, scomm \,|\} \equiv_{\mathrm{T}} \mathsf{mask}(System''_{vVote}) \setminus \{|\, scomm \,|\}$$

FDR verifies that the two systems refine each other, meaning that they are trace equivalent and hence that the intruder cannot distinguish them. As a result, the vVote voting system model provides anonymity under the Dolev-Yao intruder model.

7.4 Analysis under Alternative Assumptions

In the previous section, the analysis was conducted under the assumption of honest protocol participants except of the third voter James, and of the existence of a Dolev-Yao intruder interacting with the participants. Although, the framework used in the previous section provides a firm comprehensive foundation for analysis of voting systems, it is also important to see whether the framework supports further extensions to those assumptions made previously, because one of the important challenges in electronic voting systems may be to maintain requirements even under the assumption of the corrupt agents, for instance, misbehaving participants. Such analyses are possible with slight modifications to the voting system and the intruder models. The following paragraphs present two of these analyses of the vVote under different assumptions.

Corrupt POD Service The POD service is an important part of the print-on-demand protocol. It receives raw ballot data including a serial number s and candidate list encrypted under pk_{PS}, and sends the digital ballot by signing the serial number to the POD client. If the POD service is corrupt, which is modelled as that the POD service's secret is possessed by the intruder, the raw ballot received by the POD service, say $\mathsf{Raw}(s_3, \mathsf{E}_{pk_{PS}}(\mathsf{Sq}.\langle Zoe, Victor\rangle))$, can be captured and decrypted by the intruder. Hence, the intruder can extract the candidate list $\mathsf{Sq}.\langle Zoe, Victor\rangle$ and deduce its association with the serial number s_3. Following this, when he observes that Alice's receipt with the index value $\mathsf{Ind}.1$ has the serial number s_3 on it, he is then able to infer that Alice has voted for the first candidate of the candidate list $\mathsf{Sq}.\langle Zoe, Victor\rangle$, which is Zoe. Therefore, the intruder distinguishes the two systems as Alice cannot have voted for Victor. This counter-example is produced by FDR automatically and illustrated by the following partial trace.

$\langle \ldots$
$nsbcomm.authority.wbb.\mathsf{Raw}(s_3, ciphertext)$,
$nsbcomm.authority.podservice.\mathsf{Raw}(s_3, \mathsf{E}_{pk_{PS}}(\mathsf{Sq}.\langle Zoe, Victor\rangle))$,
$nsbcomm.podservice.podclient.\mathsf{DigB}(\mathsf{S}_{sk_{PS}}(s_3), ciphertext)$,
$enterBooth.Alice$,
$nsbcomm.Alice.ebm.\mathsf{Ind}.1\rangle$

It can be observed from the above trace that the intruder cannot decrypt the ciphertext in the message sent from the authority to the WBB, as it is encrypted under the authority's public key pk_{EA}, which is seen as *ciphertext* in the trace.

This scenario emphasises the importance of the single point failure in the protocol security. In the real system, however, the POD service is thresholded, meaning that all threshold parties, sign, encrypt or decrypt messages jointly, without any party learning the ballot order. Therefore, the above would be a threat against vVote, should all threshold parties collude.

Corrupt Authority A similar approach can be taken to model a corrupt election authority, who leaks sensitive information that can be used by the intruder. Since the authority is responsible for assigning random candidate lists to each requested serial number from the ballot manager, the candidate list encrypted under the authority's public key will be revealed when he is corrupt. Therefore, the intruder's accurate deduction about the candidate lists would violate voter anonymity by revealing the candidate list of a ballot form used by a particular voter. The following trace produced by FDR demonstrates that when the authority is compromised, which is modelled as the intruder knows his secret key sk_{EA}, the intruder violates Alice's anonymity by deducing how she has voted. In

more detail, the intruder can overhear the candidate order $\mathsf{Sq}.\langle Zoe, Victor\rangle$ on Alice's ballot form before she casts her vote. Once Alice indicates her preference by the index value $\mathsf{Ind}.1$, the chosen candidate, Zoe, is revealed to the intruder.

$\langle \ldots$
$enterBooth.Alice$
$nsbcomm.authority.wbb.\mathsf{Raw}(s_1, \mathsf{E}_{pk_{PS}}(\mathsf{Sq}.\langle Zoe, Victor\rangle))$,
$nsbcomm.authority.podservice.\mathsf{Raw}(s_1, ciphertext)$,
$nsbcomm.podservice.podclient.\mathsf{DigB}(\mathsf{S}_{sk_{PS}}(s_1), ciphertext)$,
$nsbcomm.Alice.ebm.\mathsf{Ind}.1\rangle$

Although no one is supposed to be observing voter interaction with the EBM, it is assumed here that the index value $\mathsf{Ind}.i$ sent from voter to the EBM can be observed, as it will be observed anyway once she takes her receipt from the receipt printer. Thus, the two counter-examples above were found by FDR when the intruder could observe these index values. If the intruder was not allowed to do so, the counter-examples would still appear once the voter has taken her receipt in a protocol run. Moreover, the two counter-example traces above include only *nsbcomm* events, which illustrates that the intruder does not need to block or spoof messages on those channels and hence a passive observer possessing the corresponding secret keys would also be able to attack the system.

There are numerous corruption scenarios one can think of and that can be modelled and analysed using this framework. In particular, the two presented here emphasise the importance of the case of a corrupt single entity, such as the election authority and POD service, where the voters are at a high risk of losing their anonymity. The vVote voting system has a solution to these problems, to some extent, by having the ballot forms generated by the threshold election authorities. However, if the other trusted entities, like the EBM, are acting dishonestly, the system is vulnerable to various attacks. Additionally, it was observed that a corrupt WBB does not reveal anything useful for the intruder to break the anonymity requirement of the system, because the WBB is public anyway.

The next section investigates the modelling and analysis of the secrecy requirement for voting systems.

7.5 Secrecy Analysis using Lazy Spy

The lazy spy intruder model [RG97] was used to verify the authentication and secrecy requirements of security protocols (the latter is also introduced in Section 3.3.4). In the NSPK analysis in Chapter 3, a secret is defined as the terms $\{AtoB, BtoA\}$, and the intruder model is defined so that when the intruder learns

one of the secrets in a protocol run, the process flags it up using the channel *intruderknows*. When this event occurs in a protocol run, the secret is not secret any more. Previously, in the anonymity requirement analysis of vVote, this event was omitted because such an event was not needed for the formal specification of this requirement. However, perhaps not for the paper-based voting systems, where the voters are generally not required to use public key pairs to encrypt their votes, but especially in remote voting systems, this specification can be used to verify whether the voting systems maintain the secrecy of the votes. That is, it can be verified whether the intruder ever gets to know a secret originated by a particular voter or any other agent. Moreover, the secret data can be defined more specifically for each voting system, such as, a candidate encrypted and cast by the voter as in the FOO scheme [FOO92], in which a voter encrypts her vote, blinds the encrypted version and sends it to the registrar. To this end, the highlighted expression below is added to the intruder model to flag up the intruder's knowledge about a secret f from the set of secrets *Banned*.

$$\begin{aligned} Ignorantof(f) \mathrel{\widehat{=}} & f \in \mathcal{M} \,\&\, learn.f \rightarrow Knows(f) \\ & \Box\ infer?t \in \{(X, f') \mid (X, f') \in \mathcal{D}, f' = f\} \rightarrow Knows(f) \end{aligned}$$

$$\begin{aligned} Knows(f) \mathrel{\widehat{=}} & f \in \mathcal{M} \,\&\, say.f \rightarrow Knows(f) \\ & \Box\ f \in \mathcal{M} \,\&\, learn.f \rightarrow Knows(f) \\ & \Box\ infer?t \in \{(X, f') \mid (X, f') \in \mathcal{D}, f \in X\} \rightarrow Knows(f) \\ & \Box\ \boldsymbol{f \in Banned \,\&\, intruderknows.f \rightarrow Knows(f)} \end{aligned}$$

Consequently, for the secrecy specification of a voting system, the following trace refinement needs to be checked.

$$STOP \sqsubseteq_{\mathrm{T}} System \setminus \Sigma \setminus \{|\, intruderknows \,|\}$$

where Σ is the alphabet of the process *System*:

$$\Sigma = \{|\, nsbcomm, take, fake, intruderknows \,|\}$$

7.6 Discussion

In the beginning of the modelling of vVote and the intruder, a need for different channel types was mentioned. Regarding this, the need came out when the model was initially analysed under the full Dolev-Yao intruder model that can overhear, intercept and spoof any messages on all channels other than the private channels. From this initial analysis, the following counter-example was produced, which

shows that with such an intruder the vVote voting system is open to anonymity attacks, which verifies the observation made in [KR05] about the FOO voting system [FOO92].

$\langle \ldots$
$scomm.podclient.Alice.\mathsf{B}(\mathsf{Sq}.\langle Zoe, Victor\rangle, \mathsf{S}_{sk_{PS}}(s_1), \mathsf{Ind}.0),$
$comm.Alice.ebm.\mathsf{Ind}.1,$
$scomm.podclient.Bob.\mathsf{B}(\mathsf{Sq}.\langle Zoe, Victor\rangle, \mathsf{S}_{sk_{PS}}(s_2), \mathsf{Ind}.0),$
$comm.Bob.ebm.\mathsf{Ind}.2,$
$closeElection,$
$comm.wbb.teller.\mathsf{V}(\mathsf{Ind}.2, ciphertext),$
$take.wbb.teller.\mathsf{V}(\mathsf{Ind}.1, ciphertext),$
$comm.teller.wbb.Zoe.0\rangle$

What the intruder does here is to block or intercept with the channel *take* all the other votes except Bob's. In this case, Alice has voted for Zoe with the index value Ind.1 and Bob has voted for Victor with Ind.2 on the private channels—the candidate orders on the private channels *scomm* are hidden in the analysis and they are revealed here just for illustration. Once the election is closed, tallying starts and the votes are transferred from the WBB to the teller, the intruder intercepts the vote with the index value Ind.1, and waits until Bob's vote is counted. Having seen that no one has voted for Zoe, the intruder then deduces that Bob has voted for Victor. This is a genuine and generic attack—not only to vVote, but it is applicable to any voting system. However, as it is not possible in a real system that the intruder can block all votes but one, it was assumed in the analysis that at least two honest votes are tallied at the end of the election. This was modelled with the existence of NSB channels adapted from Creese *et al.* [CGRZ03, CGH+05] and therefore, there exist at least two honest voters who vote on the NSB channels *nsbcomm*. On the other hand, the intruder works fully on James' messages on the public and private channels as if he votes in public.

Having modified the system with the adaptation of insecure NSB channels, it was verified that the vVote voting system provides anonymity. This, together with the corrupt agent scenarios, demonstrated that the abstract models and formal definitions of requirements are adequate for the automatic verification of voting system protocols. Additionally, it was shown that the active intruder model used in this chapter is much more powerful in terms of mounting various kinds of attacks than the passive attacker model used in the previous analyses [MHS12, MHS13], which can only observe the messages on the public channels.

7.7 Challenges Faced in the Modelling and Analysis

This section presents the challenges faced during the modelling of vVote and automated verification of it. Similar issues in the previous voting system models occur in this modelling gathering around the state explosion problem.

Recall that in Section 7.2.3 we abstracted away some of the system components, such as; thresholded decryption teller, distributed ballot generation and mixnet (see 7.2.3 for the detailed information about the abstracted components). The framework adapted here is much more complex than the original lazy spy as we have more deduction rules in our framework than lazy spy. Hence, even with this abstracted model it requires the use of compression techniques in order it to be checked by FDR. Hence, the compression function $\mathsf{chase}()$ was used as it is the most beneficial part of the original lazy spy framework. However, although the process P may have a number of paths of τs to follow to a final state, $\mathsf{chase}(P)$ chooses one path to follow. Although, chase does not preserve the semantics of nondeterminism in the processes, it does not, however, impact on the processes, for example lazy spy [Ros97]. Hence, it is safe to use in the analysis of voting systems. The operator was used in the $Intruder$ process when the intruder learns new facts with the event, *infer*. The τs that come up by hiding the occurrences of *infer* events, and that are chased by the operator represent the facts learned by the intruder.

$$
\begin{aligned}
&Intruder \mathrel{\widehat{=}} \\
&\quad \mathsf{chase}((\underset{f \in Learnablefacts}{\|} (Ignorant(f), AlphaL(f)))\backslash\{|infer|\}) \\
&\qquad ||| \; SayKnown
\end{aligned}
$$

Additionally, as the states needed for analysis grow quickly, some of the system behaviour can also be omitted, however, this would need an extra effort in order not to lose possible system attacks. In the modelling of vVote here, the voter always takes her receipt—the intruder never blocks the channel on which the voter takes her receipt. However, if it is desired, then more power could be given to the intruder in order him to block these channels by extending *Ucomm* as demonstrated in Section 7.2.5. On the other hand, the impact of this for FDR would be huge. After giving such power to the intruder, the number of states that FDR needs to check increases from $16,063,214$ to $42,945,122$ million states. Therefore, some of the system behaviour that would give no advantage rather than postponing the verification may be restricted in order to get a small number of state search.

Similar to the analysis of Prêt à Voter in the previous chapter, a masking function was needed to be employed here too in order not to produce false-positives when the intruder can distinguish encrypted ciphertexts.

Table 7.3 illustrates the verification times of the automated analysis of vVote voting system based on the efficient models. In the table, the restricted Dolev-Yao (D-Y) is the intruder model that is restricted to only a subset (James's communications) of all messages, whereby he can act as in the Dolev-Yao intruder model. The restriction is modelled with the existence of private and NSB channels. Additionally, the full D-Y model is where the intruder can act maliciously on all channels (there does not exist any NSB channel), but the private ones—voters' privacy is still maintained. However, the refinement does not hold, which necessitates the NSB channels in the model (see Section 7.6 for a discussion about the need for the NSB channels and a counter-example trace). The restricted and full D-Y results cannot be compared with each other, as the verification times vary depending on the voters' being honest or dishonest, they give some idea about how large a model FDR can handle before state explosion for each test.

Restricted D-Y				Full D-Y			
	Refine	States	Time		Refine	States	Time
3v 2c	✓	16,063,214	$1h14m56s$	2v 2c	X	899,494	$1m45s$
3v 3c	–	–	–	2v 3c	X	5,040,658	$22m26s$
4v 2c	–	–	–	2v 4c	–	–	–
4v 3c	–	–	–	3v 2c	–	–	–

Table 7.3: The FDR verification times for vVote. As the required state space grows quickly with the number of voters and candidates, it was not possible to produce results in some cases as FDR cannot handle with such huge states. Those are denoted as "–" in the table.

7.8 Summary

In this chapter, a formal approach to modelling and analysis of cryptographic voting systems has been proposed. In addition to the previously defined anonymity requirement, a CSP approach for formal specification of secrecy has been given in terms of trace refinements. In order to validate the suitability of the framework, the vVote voting system was analysed against the anonymity requirement. To do so, in addition to the cryptographic deduction rules expressed in Section 3.3, an extensive number of other such rules regarding voting systems have been defined. These enable the intruder to learn and deduce further from his knowledge so as to able to use it to break the protocol objectives. Moreover, special channel types,

private and NSB channels, have been introduced in order to reason about voting systems under appropriate assumptions, as it has been observed that no voting system model is anonymous under the full Dolev-Yao intruder model. Moreover, as voting systems are too complex to model and analyse in full detail, some of the properties of vVote, such as, distributed ballot generation with a thresholded POD service, have been abstracted away by assuming that thresholded parties work as a single honest agent.

The automated verification of these refinements has been conducted using the FDR model checker. It was shown that even though the voting system models are too complex resulting in a number of deduction rules being needed for reasoning about them, FDR together with lazy spy is adequate in the analysis of such systems thanks to the latter's efficient structure, thus avoiding unnecessary inferences. It has also been seen that the framework used in this chapter is very efficient in terms of cutting down the unnecessary states and quite flexible for modelling any misbehaviour by corrupt agents, when analysing voting systems under different assumptions, for instance, with a corrupt authority and a POD service.

Chapter 8

Conclusion and Future Work

This chapter reviews the results that were achieved in this thesis, presents the limitations of this work that have been identified and the future direction that it might be useful to focus upon.

This thesis will help the evaluation of voting systems by defining some of the security requirements and providing novel approaches to automatic analysis in formal methods. In particular, having shown that for a rigorous security protocol analysis, suitable and concise specifications of the requirements are needed, an anonymity definition has been given that has been demonstrated to be appropriate for the analysis of voting systems through analysing of this feature for four cryptographic and non-cryptographic voting systems.

Throughout the analysis, it is has been shown that the underlying assumptions of voting system designs play a crucial role when claiming that the scheme provides certain desired properties. That is, an assumption made when designing a voting system should be precisely defined so as to understand whether the system truly provides what it is claimed. Moreover, a voting system may be secure under an assumption, which is not realistically satisfied in practice, thus requiring an efficient realistic assumption. In summary, the assumptions under which protocols are secure should be: realistic and precisely defined. This conclusion has been made clearer during the analysis of the ThreeBallot voting system. That is, it was demonstrated that this system does not provide anonymity under various formulations of the short ballot assumption. However, given a reasonable and plausible interpretation of this assumption, the ThreeBallot is, in fact, protected from reconstruction attacks.

Cryptographic voting systems, which are similar to security protocols for many aspects, may contain flaws in their design, requiring rigorous formal analyses. However, as such protocols are too complex for analysis at a full level of de-

tail with current methods, it is hard to provide a technique to analyse them. Hence, the thesis has provided a novel approach to the formal analysis of cryptographic voting systems using lazy spy [RG97], acting as the Dolev-Yao intruder model [DY83], with a number of modifications to the model so as to capture voting system properties. Moreover, the framework was also found to be flexible when analysing voting systems under alternative assumptions such as there being a corrupt mixnet or election authority. Additionally, the framework presented here has been shown to be applicable for secrecy analysis in general, but in particular for remote voting systems.

In the analyses throughout this thesis, the FDR model checker [GGH$^+$] has been used. It has been demonstrated that FDR, together with the lazy spy intruder model or the passive attacker model, was efficient for automatically finding possible attacks on voting systems, for instance, the CVS, ThreeBallot, Prêt à Voter and vVote voting systems, should they fail to meet the anonymity requirement.

8.1 Limitations

State space considerations meant that only relatively small models with a few voters and candidates could be verified as shown in Tables 5.5 and 7.3. Moreover, although automated analysis has been shown to be a successful approach as proved by finding counter-examples for a failure of the protocol requirement, proving directly that the system meets the claimed properties would need an infinite-state model. In order to generalise the verification to models of arbitrary size, there are several techniques in the literature that can be employed, such as *structural* and *data-independent* induction [Ros97, Laz99, Hea10, Ros10]. However, data-independence techniques do not easily apply to the models that have been developed here as the established results require rather strict conditions, which have not been satisfied in our models. It is not currently clear whether it is possible to manipulate them into the appropriate form, but it seems unlikely. For instance, using functions (such as card) on data-types and the replicated parallel operator is not allowed in the models so as to satisfy data-independence technique, which are key features in our models presented in this thesis. Moreover, inequality tests are also forbidden in CSP models, and these are implicit in determining a winner in our models. The most important limitation with these techniques is regarding the specifications: there should be no hiding or renaming operators used in the specification. However, the anonymity specifications applied in this work are based on these operators. Similarly, the structural induction technique also appears to be a promising approach, but there are a few limitations with this technique too. For instance, it will be necessary to be creative so to find a finite-state description of the behaviour for use in the inductive step [Ros10], and it is not clear that this is possible as larger models will have

more states unless a way can be found of abstracting them away. That is, this technique will not be applicable to any model that publishes a final tally, because the addition of extra voters increases the number of possible results and hence impedes clear finite description. On the other hand, a generalisation based on the following opinions can be made, along the lines of work by Ryan *et al.* [RSG+00]:

> "Any attack on Anna and Bob that could ever arise through the intruder using any number of third party identities could equally arise if all of these were replaced in the trace by a single identity other than Anna or Bob."

Moreover, Roscoe [Ros97] observes that

> "With the great majority of protocols it would not improve the intruder's prospects to have any more identities (Donald, Eve, etc.) to play with. This is because any attack which involved more than one of these identities acting would also work with them all being Cameron."

Similarly, Syverson *et al.* [SMC00] have established that the Dolev-Yao intruder with multiple identities can be reduced to one.

In the case of the vVote analysis, for instance, there existed two honest voters and a dishonest one James. Under the statements above, it therefore seems likely that the existence of other dishonest voters, like James, in an election run, would not change the analytical results achieved during the analysis of this voting system. This is because the existence of more than one dishonest voter would not give any further information to the intruder that could be used to attack the protocol requirement.

It should be noted that although the CSP models were aimed at reflecting the voting system protocols, it was not possible to model them completely and consequently, some parts of the voting systems had to be abstracted away, such as: a threshold set of agents, decryption key sharers and a complete mixnet. Nonetheless, they were sufficient to demonstrate the appropriateness of the framework in automated verification of cryptographic and non-cryptographic voting systems. Additionally, due to the abstracting away of cryptographic algorithms and assuming they work perfectly, the attacks that may be caused by them have not been covered in the analyses of cryptographic voting systems in this thesis.

8.2 Future Work

There are several directions which can broaden research on the analysis of voting system protocols. Firstly, the frameworks proposed in this thesis could also be used to analyse such protocols against a number of other requirements, such as fairness. Although, the term is used to describe different desirable properties, what is meant here is that no partial results, which could affect voters' preferences, are revealed before the tallying phase. This property is described in terms of reachability and observational equivalence as a secrecy property and has been used to analyse the voting system FOO [FOO92] in [KR05]. Hence, the framework proposed in this thesis for the secrecy property can also be used to verify voting systems regarding such properties that can be defined in terms of secrecy.

Additionally, other privacy-related voting system properties that are worth investigating are coercion-resistance and receipt-freeness. Heather and Schneider [HS12] proposed a framework for automated verification of these properties. It is stated that their framework does not seem possible for mechanised analysis due to the structure of the refinement used in their coercion-resistance definition. In more detail, each instruction that may be given to the voter by the coercer needs to be defined as a CSP process so as to compose the set of all possible instructions. However, it is not realistic or plausible to define all possible coercer instructions as a CSP process because the coercer can misbehave in so many ways that we may not foresee. Therefore, further research regarding these properties (coercion-resistance and receipt-freeness) is a future direction worth pursuing.

An attempt to automate the analysis of these properties was conducted by Backes *et al.* [BHM08] using the symbolic definitions of them by Delaune *et al.* [DKR06]. However, it was not fully mechanised as some human effort was needed when transforming each of the equivalences in the coercion-resistance definition into a biprocess. This researcher's intuition is that coercion-resistance can be automatically checked by modelling the most important instructions given by the intruder to the voters, such as randomisation, Italian and forced-abstention attacks, and analysing voting systems against these attacks individually.

As mentioned in the limitations section, the Prêt à Voter and vVote voting system models used in this thesis are not complete versions, because it was assumed that certain parts of the systems work honestly and correctly, such as there being: a distributed ballot generation, a threshold set of mixes and that the WBB works in a distributed fashion. Hence, a logical next step would be automated analysis of these complex mechanisms, in terms of such aspects as: robustness, correctness and anonymity. Moreover, because of the abstraction level in the analyses in this study, it was not possible to investigate some of the cryptographic primitives that are used in voting systems especially in mixnets, such as, zero-knowledge

proofs and homomorphic encryptions. Hence, the applicability of such primitives in the adapted approach remains an open research question requiring further investigation.

The four voting systems investigated in this thesis were all paper-based including two cryptographic and two non-cryptographic. In order to extend the application domain of the framework presented, some of the remote voting systems, such as JCJ [JCJ05] and Pretty Good Democracy (PGD) [RT09] requiring different kinds of trust assumptions, should be considered for analysing automatically. Although voting remotely is more convenient than voting in polling station, there exist issues in authentication between the voter and voting system server, and trustworthiness of the integrity of the election results as well as voter anonymity and secrecy of the vote. In terms of secrecy in remote voting systems, it can be defined in many ways depending on the methods that are used in system design. For instance, the secrecy of the vote in postal voting depends on the secrecy of the conventional mail system. Moreover, in an internet-based remote voting system where the voter uses her public key to encrypt her vote, the secrecy of the vote depends on the secrecy of her secret key and the strength of the public key algorithm. To sum up, for automated analysis of remote voting systems, definitions of the requirements, such as, secrecy, may differ from the ones in paper-based voting systems, and channel types that are different than the ones used in this thesis may be required.

It has been observed that other process calculus tools and approaches, such as spi-calculus [AG97] and the automated tool ProVerif [Bla01] as well as the process-algebraic language μCRL and its toolset [BFG$^+$01], have also been used in the analysis of voting systems. Moreover regarding this, a comparison between these tools has been occasionally asked for by the reviewers of the published papers and hence, further endeavours to this end could be undertaken. Similarly, as the approach in the analyses in this research was possibilistic, deploying a probabilistic framework in accordance with some of reviewers' comments of work previously submitted by this researcher could prove beneficial. For instance, in Chapter 5 when measuring the likelihood of the privacy assumptions' being fulfilled in the ThreeBallot voting system, a probabilistic approach might have been deployed, such as in the line of [DPP07], in order to capture levels of privacy rather than an absolute measure. However, ThreeBallot has already been subjected to such treatment in [Str06a, Str06b].

Appendices

Appendix A

Sanity Checks

In model checking, one can question the correctness of the system model. Such possible errors in designing the desired behaviour of systems can be detected by further reasoning via various sanity checks written as specifications. If the system model behaves correctly for all possible cases, then one can be sure that the model is correct. However, it is not an easy task to try all the possible cases, instead, a number of necessary sanity checks can help to gain confidence in the model. The following sections present sanity checks performed for each voting system modelled and analysed in this thesis. The CSP$_M$ codes for the CSP models and the analysis of the voting systems involved are available on [Mor13]. Note that they may differ slightly from the ones presented here in terms of the syntax used.

A.1 The Conventional Voting System Model

Before performing a formal analysis on the CVS model, it is wise to check that the voting system preserves some desired properties, by means of appropriate sanity checks (see Section 4.4 for the analysis of the model).

A.1.1 No one can vote after the election

The model should not allow a voter to vote after the election is closed. That is, no *cast* event should be observed following a *closeElection* event. The sanity specification and the assertion to be checked can be expressed as follows:

$$Sanity\_Spec_1 \mathrel{\widehat{=}} closeElection \rightarrow Closed \;\Box\; \Box_{\substack{v\in\mathcal{V}\\ s\in\mathcal{S}\\ c\in\mathcal{C}}} \Big(cast.v.s.c \rightarrow Sanity\_Spec_1 \Big)$$

$$Closed \mathrel{\widehat{=}} closeElection \rightarrow Closed$$

$$Sanity\_Spec_1 \sqsubseteq_{\mathrm{T}} System_{CVS} \setminus \Sigma \setminus \{| \, closeElection, cast \, |\}$$

A.1.2 The number of votes tallied corresponds to the number of cast votes

The model can also be checked as to whether the number of votes tallied at the end of the election run with the model corresponds to the number of votes cast during the election. The specification $Sanity\_Spec_2$ and the assertion for this test can be defined as follows:

$$Sanity\_Spec_2 \mathrel{\widehat{=}} Count(0)$$
$$Count(n) \mathrel{\widehat{=}}$$
$$\Box_{\substack{v \in \mathcal{V} \\ s \in \mathcal{S} \\ c \in \mathcal{C}}} \Big(\; cast.v.s.c \rightarrow \text{if } n \leq \mathsf{card}(\mathcal{V}) \text{ then } Count(n+1) \text{ else } STOP \; \Big)$$
$$\Box$$
$$\Box_{i \in \{0 \ldots \mathsf{card}(\mathcal{V})\}} \Big(\; total.c_1.i \rightarrow Count_1(n-i) \; \Big)$$

$$Count_1(s) \mathrel{\widehat{=}}$$
$$\Box_{j \in \{0 \ldots \mathsf{card}(\mathcal{V})\}} \left(\begin{array}{l} total.c_2.j \rightarrow \text{if } s = j \text{ then } total.c_3.0 \rightarrow SKIP \text{ else} \\ \left(\begin{array}{l} \text{if } 0 \leq s - j \text{ and } s - j \leq \mathsf{card}(\mathcal{V}) \text{ then} \\ \qquad total.c_3.(s-j) \rightarrow STOP \text{ else } STOP \end{array} \right) \end{array} \right)$$

Note that the processes are restricted in order to avoid an infinite number of states as FDR cannot handle this, such as, $s - j \leq \mathsf{card}(\mathcal{V})$ above. It is not necessary for the process itself, but useful for FDR.

$$Sanity\_Spec_2 \sqsubseteq_{\mathrm{T}} System_{CVS} \setminus \Sigma \setminus \{| \, cast, total \, |\}$$

As expected, the sanity checks are satisfied showing that the model does not allow votes after the election is closed, and nor does it miscount the total number of votes. There are other sanity checks that it is wise to perform on the model, but for brevity just these two above are discussed here.

A.2 The ThreeBallot Voting System Model

The following sanity checks are used to gain more confidence in the correct behaviour of the ThreeBallot model analysed in Chapter 5.

A.2.1 No one can be authenticated twice

It is ensured in the ThreeBallot model that a voter can be authenticated only once to prevent multiple voting attacks. Thus, the specification for this check is modelled as follows:

$$\begin{aligned} Auth(v) &\mathrel{\widehat{=}} auth.v \rightarrow STOP \\ Sanity\_Spec_3 &\mathrel{\widehat{=}} |||_{v \in \mathcal{V}} Auth(v) \end{aligned}$$

Hence, the assertion that needs to be checked is:

$$Sanity\_Spec_3 \sqsubseteq_{\mathrm{T}} System_{3B} \setminus \Sigma \setminus \{| \, auth \, |\}$$

A.2.2 No one can fill in a ballot form before being authenticated

With the first sanity check of the ThreeBallot model, only the eligible voters can vote and they can vote only once in an election. Because only the *auth* and the *place* events are involved in this sanity test, all the other events can be hidden. Thus, regarding this sanity check, the specification $Sanity\_Spec_4$ can be written as follows:

$$\begin{aligned} Check(v) &\mathrel{\widehat{=}} auth.v \rightarrow Authed(v) \\ Authed(v) &\mathrel{\widehat{=}} \Box_{(i,j) \in Coords} \Big(\, place.v.(i,j) \rightarrow Authed(v) \, \Big) \\ &\quad \Box \\ &\quad auth.v \rightarrow Authed(v) \\ Sanity\_Spec_4 &\mathrel{\widehat{=}} |||_{v \in \mathcal{V}} Check(v) \end{aligned}$$

and the refinement that needs to be held is:

$$Sanity\_Spec_4 \sqsubseteq_{\mathrm{T}} System_{3B} \setminus \Sigma \setminus \{| \, auth, place \, |\}$$

A.2.3 No one can vote after the election

For the sake of fairness and the correctness of the election results, all eligible voters should be able to vote during the election — once the election is closed, casting ballots should not be allowed. For this sanity check, all irrelevant events can be hidden except the *closeElection* and *place* events. Thus, the aim is not to let the $System_{3B}$ process allow any *place* events happening after a *closeElection* event. In consequence, the specification $Sanity\_Spec_5$ can be expressed as follows:

$$Sanity\_Spec_5 \mathrel{\widehat{=}} closeElection \rightarrow Closed \;\Box \Box_{\substack{v\in\mathcal{V}\\(i,j)\in Coords}} \Big(place.v.(i,j) \rightarrow Sanity\_Spec_5 \Big)$$
$$Closed \mathrel{\widehat{=}} closeElection \rightarrow Closed$$

The following is the refinement check for this sanity test.

$$Sanity\_Spec_5 \sqsubseteq_{\mathrm{T}} System_{3B} \setminus \Sigma \setminus \{| \, closeElection, place \, |\}$$

A.2.4 The number of votes tallied corresponds to the number of cast votes

It can be verified whether the model reflects the number of cast votes to the final tally. Hence, if the total number of *place* events is the same as the number of votes tallied for each candidate at the end of election (total number of votes is announced with the *total* events), the specification is satisfied. Because only the events *place* and *total* are dealt with, the rest of the events can be hidden. The corresponding specification $Sanity\_Spec_6$ can be defined as follows:

$$Sanity\_Spec_6 \mathrel{\widehat{=}} Count(0)$$
$$Count(p) \mathrel{\widehat{=}} \Box_{\substack{v\in\mathcal{V}\\(i,j)\in Coords}} \Big(place.v.(i,j) \rightarrow Count(p+1) \Big) \;\Box \Box_{c\in\mathcal{C}\setminus\{c_1\}} \Big(total.c \rightarrow Count_1(p-i) \Big)$$
$$Count_1(j) \mathrel{\widehat{=}} total.c_1.j \rightarrow STOP$$

Therefore, the following refinement check needs to be held in order for the model to satisfy this sanity test.

$$Sanity\_Spec_6 \sqsubseteq_{\mathrm{T}} System_{3B} \setminus \Sigma \setminus \{| \, total, place, done \, |\}$$

As expected, the sanity checks above were successfully satisfied by the ThreeBallot CSP model.

A.3 The Prêt à Voter Voting System Model

The following four sanity checks provide more confidence that the behaviour of Prêt à Votermodel is as expected, which is analysed in Chapter 6.

A.3.1 No one can be authenticated twice

The following is the specification for this sanity check, which ensures that no one can be authenticated multiple times in the model.

$$\begin{aligned} &\mathit{Auth(v)} &&\widehat{=} \mathit{auth.v} \rightarrow \mathit{STOP} \\ &\mathit{Sanity\_Spec_7} &&\widehat{=} |||_{v \in \mathcal{V}} \mathit{Auth(v)} \end{aligned}$$

Hence the refinement below, in which the events in Σ other than the event *auth* are hidden, should be satisfied by the Prêt à Voter voting system model. A violation of the specification would mean that the model allows a voter to be authenticated twice.

$$\mathit{Sanity\_Spec_7} \sqsubseteq_{\mathrm{T}} \mathit{System_{PaV}} \setminus \Sigma \setminus \{\!| \mathit{auth} |\!\}$$

A.3.2 No one can fill in a ballot form before being authenticated

This sanity is involved with the authentication and marking events *auth* and *mark*, respectively, and hence the other events from Σ can be hidden. Additionally, this specification is not concerned with how many times a voter can authenticate herself to the election official. Thus, the specification process $\mathit{Sanity\_Spec_8}$ can be modelled as:

$$\begin{aligned} &\mathit{Check(v)} &&\widehat{=} \mathit{auth.v} \rightarrow \mathit{Authed(v)} \\ &\mathit{Authed(v)} &&\widehat{=} \Box_{x \in \mathit{markedforms}} \Big(\mathit{mark.v.x} \rightarrow \mathit{Authed(v)} \Big) \\ &&&\quad \Box \\ &&&\quad \Big(\mathit{auth.v} \rightarrow \mathit{Authed(v)} \Big) \\ &\mathit{Sanity\_Spec_8} &&\widehat{=} |||_{v \in \mathcal{V}} \mathit{Check(v)} \end{aligned}$$

The following refinement check ensures that the Prêt à Voter model allows only authenticated voters to mark a ballot form.

$$\mathit{Sanity\_Spec_8} \sqsubseteq_{\mathrm{T}} \mathit{System_{PaV}} \setminus \Sigma \setminus \{\!| \mathit{auth}, \mathit{mark} |\!\}$$

A.3.3 No one can vote after the election

The Prêt à Voter CSP model $\mathit{System_{PaV}}$ process should not allow a *cast* event after a *closeElection* event. Hence, the following ensures that such a trace is not possible in the model. Similarly, the events in Σ except for the *closeElection* and *cast* events can be hidden.

$$Sanity\_Spec_9 \mathrel{\widehat{=}} closeElection \rightarrow Closed \;\Box \Box_{x \in markedRHSs} \Big(cast.x \rightarrow Sanity\_Spec_9 \Big)$$
$$Closed \mathrel{\widehat{=}} closeElection \rightarrow Closed$$

Hence, the following refinement needs to be checked for this sanity test.

$$Sanity\_Spec_9 \sqsubseteq_{\mathrm{T}} System_{PaV} \setminus \Sigma \setminus \{| \, closeElection, cast \, |\}$$

A.3.4 The number of votes tallied corresponds to the number of cast votes

What is checked here is whether the total number of *cast* votes is the same as the number of votes tallied at the end of the election. Because the events that are involved in this sanity check are the *cast* and *total* events, the rest of the events in Σ can be hidden. The specification $Sanity\_Spec_{10}$ and the refinement check can be defined as follows:

$$Sanity\_Spec_{10} \mathrel{\widehat{=}} Count(0)$$
$$Count(n) \mathrel{\widehat{=}} \Box_{x \in markedRHSs} \Big(cast.x \rightarrow Count(n+1) \Big) \;\Box \Box_{i \in \{0 \ldots \mathsf{card}(\mathcal{V})\}} \Big(total.c_1.i \rightarrow Count_1(n-i) \Big)$$
$$Count_1(j) \mathrel{\widehat{=}} total.c_2.j \rightarrow STOP$$

The following refinement verifies whether the Prêt à Voter system model satisfies this sanity test.

$$Sanity\_Spec_{10} \sqsubseteq_{\mathrm{T}} System_{PaV} \setminus \Sigma \setminus \{| \, cast, total \, |\}$$

FDR confirms that all the sanity checks defined above are satisfied by the Prêt à Voter voting system CSP model.

A.4 The vVote Voting System Model

The following four sanity checks provide more confidence that the behaviour of vVote model is as expected and the analysis of this is carried out in Chapter 7.

A.4.1 No one can be authenticated twice

The following is the specification for this sanity check, which ensures that no one can be authenticated twice in the model.

$$
\begin{aligned}
&\mathit{Auth(v)} &&\widehat{=} \mathit{nsbcomm.v.Tom.v} \rightarrow \mathit{STOP} \\
&\mathit{Sanity\_Spec_{11}} &&\widehat{=} |||_{v \in \mathcal{V}} \mathit{Auth(v)}
\end{aligned}
$$

The events in Σ other than the events where the voters authenticate themselves can be hidden and hence, the following refinement check should be satisfied by the vVote voting system model. A violation of the specification would mean that the model allows a voter to be authenticated twice.

$$\mathit{Sanity\_Spec_{11}} \sqsubseteq_{\mathrm{T}} \mathit{System_{vVote}} \setminus \Sigma \setminus \{| \mathit{nsbcomm.v.Tom.v} \mid v \in \mathcal{V} |\}$$

A.4.2 No one can fill in a ballot form before being authenticated

As the authentication and marking events, $\mathit{nsbcomm.v.Tom.v}$ and $\mathit{nsbcomm.v.ebm.i}$, respectively, are involved in this sanity check, the other events from Σ can be hidden. Additionally, this specification is not concerned with how many times a voter can be authenticated and thus, the specification process $\mathit{Sanity\_Spec_{12}}$ can be written as:

$$
\begin{aligned}
&\mathit{Check(v)} &&\widehat{=} \mathit{nsbcomm.v.Tom.v} \rightarrow \mathit{Authed(v)} \\
&\mathit{Authed(v)} &&\widehat{=} \Box_{i \in \mathcal{I}} \Big(\mathit{nsbcomm.v.ebm.i} \rightarrow \mathit{Authed(v)} \Big) \\
& && \quad \Box \\
& && \quad \Big(\mathit{nsbcomm.v.Tom.v} \rightarrow \mathit{Authed(v)} \Big) \\
&\mathit{Sanity\_Spec_{12}} &&\widehat{=} |||_{v \in \mathcal{V}} \mathit{Check(v)}
\end{aligned}
$$

The following refinement check ensures that the vVote model allows only authenticated voters to send an index value, corresponding to the chosen candidate, to the EBM.

$$\mathit{Sanity\_Spec_{12}} \sqsubseteq_{\mathrm{T}} \mathit{System_{vVote}} \setminus \Sigma \setminus A$$

, where $A = \{| \mathit{nsbcomm.v.Tom.v}, \mathit{nsbcomm.v.ebm} \mid v \in \mathcal{V} |\}$.

A.4.3 No one can vote after the election

The vVote CSP model, $System_{vVote}$, should not allow a vote casting event, modelled as $nsbcomm.v.ebm.i$ after a $closeElection$ event. Hence, the following ensures such a trace is not possible in the model.

$$Sanity\_Spec_{13} \mathrel{\widehat{=}} closeElection \rightarrow Closed \;\Box\; \Box_{\substack{v \in \mathcal{V} \\ i \in \mathcal{I}}} \Big(nsbcomm.v.ebm.i \rightarrow Sanity\_Spec_{13} \Big)$$
$$Closed \mathrel{\widehat{=}} closeElection \rightarrow Closed$$

Hence, the following verifies whether the model satisfies this sanity test.

$$Sanity\_Spec_{13} \sqsubseteq_{\mathrm{T}} System_{vVote} \setminus \Sigma \setminus A$$

, where $A = \{| \, nsbcomm.v.ebm, closeElection \mid v \in \mathcal{V} \, |\}$.

A.4.4 The number of votes tallied corresponds to the number of cast votes

What is checked here is whether the total number of casting events modelled as $nsbcomm.v.Ebm.i$ is the same as the number of votes tallied, which is the sum of votes received by the candidates Zoe and Victor. Hence, the specification $Sanity\_Spec_{14}$ can be defined as follows:

$$Sanity\_Spec_{14} \mathrel{\widehat{=}} Count(0)$$
$$Count(n) \mathrel{\widehat{=}} \Box_{\substack{v \in \mathcal{V} \\ i \in \mathcal{I}}} \Big(nsbcomm.v.ebm.i \rightarrow Count(n{+}1) \Big) \;\Box\; \Box_{i \in \{0 \ldots \mathsf{card}(\mathcal{V})\}} \Big(nsbcomm.teller.wbb.Zoe.i \rightarrow Count_1(n-i) \Big)$$
$$Count_1(j) \mathrel{\widehat{=}} nsbcomm.teller.wbb.Victor.j \rightarrow STOP$$

The following refinement verifies whether the vVote system model satisfies this sanity check.

$$Sanity\_Spec_{14} \sqsubseteq_{\mathrm{T}} System_{vVote} \setminus \Sigma \setminus A$$

, where $A = \{| \, nsbcomm.v.ebm, nsbcomm.teller.wbb \mid v \in \mathcal{V} \, |\}$.

FDR confirms that all the sanity checks defined above are satisfied by the vVote voting system CSP model.

Bibliography

[ACW+06] R. Araújo, R. Custódio, A. Wiesmaier, T. Takagi, and Technische Universitat Darmstadt. An electronic scheme for the Farnel paper-based voting protocol. In *ACNS*, 2006.

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th USENIX Security Symposium*, pages 335–348, 2008.

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, pages 36–47, New York, NY, USA, 1997. ACM.

[App07] Andrew W. Appel. How to defeat Rivest's ThreeBallot voting system. Unpublished, 2007.

[BBC04] BBC:website. Florida ballot papers go missing, October 2004. `http://news.bbc.co.uk/1/hi/world/americas/3960679.stm`.

[BBK+12] Josh Benaloh, Mike Byrne, Philip T. Kortum, Neal McBurnett, Olivier Pereira, Philip B. Stark, and Dan S. Wallach. STAR-Vote: A secure, transparent, auditable, and reliable voting system. *CoRR*, abs/1211.1904, 2012.

[BCH+12a] Craig Burton, Chris Culnane, James Heather, Thea Peacock, Peter Y. A. Ryan, Steve Schneider, Sriramkrishnan Srinivasan, Vanessa Teague, Roland Wen, and Zhe Xia. A supervised verifiable voting protocol for the Victorian Electoral Commission. In *Electronic Voting*, pages 81–94, 2012.

[BCH+12b] Craig Burton, Chris Culnane, James Heather, Thea Peacock, Peter Y. A. Ryan, Steve Schneider, Sriramkrishnan Srinivasan, Vanessa

Teague, Roland Wen, and Zhe Xia. Using Prêt à Voter in Victoria State elections. In *EVT / WOTE*, 2012.

[BFG+01] Stefan Blom, Wan Fokkink, Jan Friso Groote, Izak van Langevelde, Bert Lisser, and Jaco van de Pol. μCRL: A toolset for analysing algebraic specifications. In *CAV*, pages 250–254, 2001.

[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 '02, pages 68–77, New York, NY, USA, 2002. ACM.

[BHM08] Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *CSF*, pages 195–209, 2008.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW*, pages 82–96, 2001.

[Bla09] Matt Blaze. Is the e-voting honeymoon over?, March 2009. `http://www.crypto.com/blog/vote_fraud_in_kentucky/`.

[BP05] Mohit Bhargava and Catuscia Palamidessi. Probabilistic anonymity. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 - Concurrency Theory*, volume 3653 of *Lecture Notes in Computer Science*, pages 171–185. Springer Berlin / Heidelberg, 2005.

[BRS07] A. Baskar, R. Ramanujam, and S. P. Suresh. Knowledge-based modelling of voting protocols. In *TARK*, pages 62–71, 2007.

[BT94] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *STOC*, pages 544–553, 1994.

[Can01] Ran Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symp. Foundations of Computer Science*, pages 136–145, 2001.

[CCC+08] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity II: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *Proceedings of the Conference on Electronic Voting Technology*, pages 14:1–14:13, Berkeley, CA, USA, 2008. USENIX Association.

[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 Conference on Security*, USENIX Security'10, pages 291–306, Berkeley, CA, USA, 2010. USENIX Association.

[CCK12] Rohit Chadha, Stefan Ciobaca, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. In Helmut Seidl, editor, *Programming Languages and Systems*, volume 7211 of *Lecture Notes in Computer Science*, pages 108–127. Springer Berlin Heidelberg, 2012.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368, 2008.

[CEA07] Jeremy Clark, Aleks Essex, and Carlisle Adams. On the security of ballot receipts in E2E voting systems. In *IAVoSS Workshop On Trustworthy Elections (WOTE)*, july 2007.

[CEC+08] David Chaum, Aleksander Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan T. Sherman, and Poorvi L. Vora. Scantegrity: End-to-end voter-verifiable optical-scan voting. *IEEE Security & Privacy*, 6(3):40–46, 2008.

[CGH+05] Sadie Creese, Michael Goldsmith, Richard Harrison, Bill Roscoe, Paul Whittaker, and Irfan Zakiuddin. Exploiting empirical engagement in authentication protocol design. In *Proceedings of the Second International Conference on Security in Pervasive Computing*, SPC'05, pages 119–133, Berlin, Heidelberg, 2005. Springer-Verlag.

[CGRZ03] Sadie Creese, Michael Goldsmith, Bill Roscoe, and Irfan Zakiuddin. The attacker in ubiquitous computing environments: Formalising the threat model. In *Formal Aspects of Security*, 2003.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseu-donyms. *Communications of the ACM*, 24:84–90, February 1981.

[Cha88] David Chaum. The dining cryptographers problem - unconditinal sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988.

[Cha04] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, 2004.

[CKW08] Jacek Cichoń, Miroslaw Kutylowski, and Bogdan Weglorz. Short ballot assumption and Threeballot voting protocol. In *Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM'08, pages 585–598, Berlin, Heidelberg, 2008. Springer-Verlag.

[COPD06] Tom Chothia, Simona Orzan, Jun Pang, and Mohammad Torabi Dashti. A framework for automatically checking anonymity with μCRL. In *TGC*, pages 301–318, 2006.

[CPP06] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. In *Information and Computation*. Springer, 2006.

[CRS05] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. In *ESORICS*, pages 118–139, 2005.

[CS11] Veronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. In *Computer Security Foundations Symposium (CSF), 2011 IEEE 24th*, pages 297–311, 2011.

[Cul13] Chris Culnane. Software design for VEC vVote system. Technical Report CS-13-01, University of Surrey, 2013.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[DKR06] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *CSFW*, pages 28–42, 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, December 2009.

[DKR10] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Towards trustworthy elections. In David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter A. Ryan, and Josh Benaloh, editors, *Towards Trustworthy Elections*, chapter Verifying privacy-type properties of electronic voting protocols: a taster, pages 289–309. Springer-Verlag, Berlin, Heidelberg, 2010.

[dMPQ07] Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Simulation-based analysis of E2E voting systems. In *Proceedings of the 1st International Conference on E-voting and Identity*, VOTE-ID'07, pages 137–149, Berlin, Heidelberg, 2007. Springer-Verlag.

[DPP07] Yuxin Deng, Catuscia Palamidessi, and Jun Pang. Weak probabilistic anonymity. *Electronic Notes in Theoretical Computer Science*, 180(1):55–76, June 2007.

[DRS08] Stéphanie Delaune, Mark Ryan, and Ben Smyth. Automatic verification of privacy properties in the applied pi calculus. In Yucel Karabulut, John Mitchell, Peter Herrmann, and ChristianDamsgaard Jensen, editors, *Trust Management II*, volume 263 of *IFIP - The International Federation for Information Processing*, pages 263–278. Springer US, 2008.

[DSS03] David L. Dill, Bruce Schneier, and Barbara Simons. Voting and technology: who gets to count your vote? *Communications of the ACM*, 46(8):29–31, August 2003.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198 – 208, mar 1983.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.

[FA02] Cédric Fournet and Martín Abadi. Hiding names: Private authentication in the applied pi calculus. In *ISSS*, pages 317–338, 2002.

[FOO92] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *AUSCRYPT*, pages 244–251, 1992.

[GGH+] Paul Gardiner, Michael Goldsmith, Jason Hulance, David Jackson, Bill Roscoe, Brian Scattergood, and Bryan Armstrong. FDR2 user manual. `http://www.fsel.com/documentation/fdr2/html/index.html`.

[GHPv05] Flavio D. Garcia, Ichiro Hasuo, Wolter Pieters, and Peter van Rossum. Provable anonymity. In *Proceedings of the 2005 ACM Workshop on Formal Methods in Security Engineering*, FMSE '05, pages 63–72, New York, NY, USA, 2005. ACM.

[Gjø10] Kristian Gjøsteen. Analysis of an internet voting protocol. *IACR Cryptology ePrint Archive*, 2010:380, 2010.

[Goo08] Dan Goodin. E-voting outfit confesses vote-dropping software bug, August 2008. `http://www.theregister.co.uk/2008/08/26/decade_old_evoting_error/`.

[Gum05] Andrew Gumbel. *Steal this vote : dirty elections and the rotten history of democracy in America*. Nation Books, 2005.

[Hea07] James Heather. Implementing STV securely in Prêt à Voter. In *CSF*, pages 157–169, 2007.

[Hea10] James Heather. *Using rank functions to verify authentication protocols*. PhD thesis, Royal Holloway, University of London, 2010.

[HHK95] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 453–, Washington, DC, USA, 1995. IEEE Computer Society.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21:666–677, August 1978.

[HS04] Dominic Hughes and Vitaly Shmatikov. Information hiding, anonymity and privacy: a modular approach. *Journal of Computer Security*, 12(1):3–36, 2004.

[HS12] James Heather and Steve Schneider. A formal framework for modelling coercion resistance and receipt freeness. In *FM*, pages 217–231, 2012.

[HSS09] Kevin Henry, Douglas R. Stinson, and Jiayuan Sui. The effectiveness of receipt-based attacks on ThreeBallot. *IEEE Transactions on Information Forensics and Security*, 4(4):699–707, December 2009.

[Jak99] Markus Jakobsson. On quorum controlled asymmetric proxy reencryption. In *Proceedings of the Second International Workshop on Practice and Theory in Public Key Cryptography*, 1999.

[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 '05, pages 61–70, New York, NY, USA, 2005. ACM.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security Symposium*, pages 339–353, 2002.

[JMP09] Hugo Jonker, Sjouke Mauw, and Jun Pang. A formal framework for quantifying voter-controlled privacy. *Journal of Algorithms*, 64(2-3):89 – 105, 2009.

[Jon09] Hugo L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. PhD thesis, Eindhoven University of Technology and University of Luxembourg, 2009.

[JP11] Hugo Jonker and Jun Pang. Bulletin boards in voting systems: Modelling and measuring privacy. In *ARES*, pages 294–300, 2011.

[KR05] Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi calculus. In *ESOP*, pages 186–200, 2005.

[KSW05] Chris Karlof, Naveen Sastry, and David Wagner. Cryptographic voting protocols: a systems perspective. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM'05, pages 3–3, Berkeley, CA, USA, 2005. USENIX Association.

[KTV11] R. Küsters, T. Truderung, and A. Vogt. Verifiability, privacy, and coercion-resistance: New insights from a case study. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 538 –553, May 2011.

[KZ10] Miroslaw Kutylowski and Filip Zagorski. Scratch, click & vote: E2E voting over the internet. In David Chaum, Markus Jakobsson, Ronald Rivest, Peter Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 343–356. Springer Berlin / Heidelberg, 2010.

[Laz99] Ranko S. Lazic. *A Semantic Study of Data Independence with Applications to Model Checking*. D. phil. thesis, Oxford University Computing Laboratory, 1999.

[LB03] Michael Leuschel and Michael Butler. Prob: A model checker for b. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *Lecture Notes in Computer Science*, pages 855–874. Springer Berlin Heidelberg, 2003.

[LJP10] Barbara Lucie Langer, Hugo Jonker, and Wolter Pieters. Anonymity and verifiability in voting: Understanding (un)linkability. In *ICICS*, pages 296–310, 2010.

[Low95] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of the Second International*

Workshop on Tools and Algorithms for Construction and Analysis of Systems, pages 147–166, London, UK, 1996. Springer-Verlag.

[Mea03] Catherine Meadows. Formal methods for cryptographic protocol analysis: emerging issues and trends. *IEEE Journal on Selected Areas in Communications*, 21(1):44–54, 2003.

[Mer92] Rebecca T. Mercuri. Physical verifiability of computer systems. In *The Fifth International Computer Virus and Security Conference*, March 1992.

[MH13] Murat Moran and James Heather. Automated analysis of voting systems with Dolev-Yao intruder model. In *Automated Verification of Critical Systems AVOCS*, September 2013.

[MHS12] Murat Moran, James Heather, and Steve Schneider. Verifying anonymity in voting systems using CSP. *Formal Aspects of Computing*, pages 1–36, 2012.

[MHS13] Murat Moran, James Heather, and Steve A Schneider. Automated anonymity verification of the ThreeBallot voting system. In *IFM*, pages 94–108, June 2013.

[Mor13] Murat Moran. CSP codes for CVS, ThreeBallot, Prêt à Voter and vVote voting systems, May 2013. `http://muratmoran.wordpress.com/publications/`.

[MVd04] S. Mauw, J. Verschuren, and E. P. de Vink. A formalization of anonymity and onion routing. In *ESORICS*, pages 109–124, 2004.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[NAN05] Christoffer Rosenkilde Nielsen, Esben Heltoft Andersen, and Hanne Riis Nielson. Static validation of a voting protocol. *Electronic Notes in Theoretical Computer Science*, 135(1):115–134, July 2005.

[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*, CCS '01, pages 116–125, New York, NY, USA, 2001. ACM.

[NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, 1992. Springer-Verlag.

[PIK94] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In Tor Helleseth, editor, *Advances in Cryptology EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer Berlin Heidelberg, 1994.

[PK00] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity - a proposal for terminology. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 1–9, 2000.

[RBH+09] Peter Y. A. Ryan, David Bismark, James Heather, Steve A. Schneider, and Zhe Xia. Prêt à Voter: a voter-verifiable voting system. *IEEE Transactions on Information Theory*, 4(4):662–673, 2009.

[RG97] A.W. Roscoe and M.H. Goldsmith. The perfect "spy" for model-checking cryptoprotocols. In *DIMACS workshop on the design and formal verification of cryptographic protocols*, 1997.

[Riv06] Ronald L. Rivest. The ThreeBallot voting system, 2006. `http://people.csail.mit.edu/rivest/Rivest-TheThreeBallotVotingSystem.pdf`.

[Ros97] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[Ros10] A. W. Roscoe. *Understanding Concurrent Systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[RP05] Peter Y. A. Ryan and Thea Peacock. Prêt à Voter: a systems perspective. Technical report, Newcastle University, 2005.

[RP10] Peter Y. A. Ryan and Thea Peacock. A threat analysis of Prêt à Voter. In David Chaum, Markus Jakobsson, Ronald Rivest, Peter Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 200–215. Springer Berlin / Heidelberg, 2010.

[RRE+05] Andrew Reynolds, Ben Reilly, Andrew Ellis, Jose Antonio Cheibub, Karen Cox, Dong Lisheng, Jorgen Elklit, Michael Gallagher, Allen Hicken, Carlos Huneeus, Eugene Huskey, Stina Larserud, Vijay Patidar, Nigel S. Roberts, Richard Vengroff, and Jeffrey A. Weldon. Electoral system design: The new international IDEA handbook, 2005.

[RS06] Peter Y. A. Ryan and Steve A. Schneider. Prêt à Voter with re-encryption mixes. In *ESORICS*, pages 313–326, 2006.

[RS07] Ronald L. Rivest and Warren D. Smith. Three voting protocols: ThreeBallot, VAV, and Twin. In *Proceedings of USENIX/ACCURATE Electronic Voting Technology (EVT)*. Press, 2007.

[RSA78] Ronald L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[RSG+00] Peter Y. A. Ryan, Steve A. Schneider, Michael H. Goldsmith, Gavin Lowe, and A. W. Roscoe. *The Modelling and Analysis of Security Protocols : the CSP Approach*. Addison-Wesley Professional, first edition, 2000.

[RT09] Peter Y. A. Ryan and Vanessa Teague. Pretty good democracy. In *Security Protocols Workshop*, pages 111–130, 2009.

[RT10] Kim Ramchen and Vanessa Teague. Parallel shuffling and its application to Prêt à Voter. In *EVT / WOTE*, 2010.

[Rya04] Peter Y. A. Ryan. A variant of the Chaum voter-verifiable scheme. Technical Report CS-TR-864, University of Newcastle upon Tyne, 2004.

[Rya05] Peter Y. A. Ryan. A variant of the Chaum voter-verifiable scheme. In *Proc. 2005 Workshop on Issues in the Theory of Security*, pages 81–88, 2005.

[Rya06] Peter Y. A. Ryan. Putting the human back in voting protocols. In *Security Protocols Workshop*, pages 20–25, 2006.

[Rya08] Peter Y. A. Ryan. Prêt à Voter with paillier encryption. *Mathematical and Computer Modelling*, 48(1):1646–1662, 2008.

[Sch96] Steve A Schneider. Security properties and CSP. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 174–187, 1996.

[Sch99] Steve A. Schneider. *Concurrent and Real Time Systems: The CSP Approach*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1999.

[Sch04] Bruce Schneier. The problem with electronic voting machines, November 2004. `http://www.schneier.com/blog/archives/2004/11/the_problem_wit.html`.

[SLD09] Jun Sun, Yang Liu, and JinSong Dong. Model checking csp revisited: Introducing a process analysis toolkit. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science*, pages 307–322. Springer Berlin Heidelberg, 2009.

[SMC00] Paul Syverson, Catherine Meadows, and Iliano Cervesato. Dolev-Yao is no better than Machiavelli. In *First Workshop on Issues in the Theory of Security - WITS'00*, pages 87–92, 2000.

[Smy11] Ben Smyth. *Formal verification of cryptographic protocols with automated reasoning*. PhD thesis, School of Computer Science, University of Birmingham, 2011.

[SRKK10] Ben Smyth, Mark Ryan, Steve Kremer, and Mounira Kourjieh. Towards automatic analysis of election verifiability properties. In *ARSPA-WITS*, pages 146–163, 2010.

[SS96] Steve A. Schneider and Abraham Sidiropoulos. CSP and anonymity. In *ESORICS*, pages 198–218, 1996.

[Str06a] Charlie Strauss. A critical review of the triple ballot voting system, part2: Crack- ing the triple ballot encryption, 2006. `http://www.cs.princeton.edu/~appel/voting/Strauss-ThreeBallotCritique2v1.5.pdf`.

[Str06b] Charlie Strauss. The trouble with triples: A critical review of the triple ballot (3ballot) scheme part1, 2006. `http://www.cs.princeton.edu/~appel/voting/Strauss-TroubleWithTriples.pdf`.

[TPR07] Thomas Tjøstheim, Thea Peacock, and Peter Y. A. Ryan. A case study in system-based analysis: The ThreeBallot voting system and Prêt à Voter. In *VoComp*, 2007.

[XCH+10] Zhe Xia, Chris Culnane, James Heather, Hugo Jonker, Peter Y. A. Ryan, Steve A. Schneider, and Sriramkrishnan Srinivasan. Versatile

Prêt à Voter: Handling multiple election methods with a unified interface. In *INDOCRYPT*, pages 98–114, 2010.

Notation

Explanation of some notation, left this in as it might be useful.

$\mathcal{A}$	set of agents
$\mathcal{C}$	set of candidates
$\mathcal{F}$	set of facts
$\mathcal{I}$	set of indices
$\mathcal{L}$	set of candidate lists
$\mathcal{M}$	set of messages
$\mathcal{N}$	set of nonces
$\mathcal{S}$	set of serials
$\mathcal{V}$	set of voters
$\mathcal{IK}$	set of initial knowledge
$\mathcal{PK}$	set of all public keys
$\mathcal{SK}$	set of all secret keys
pk	public key variable
sk	secret key variable
k	symmetric key variable
(pk, sk)	public key pair variable
pk_a	public key variable belonging to agent a
sk_a	secret key variable belonging to agent a
f, f_i	fact variables
$\mathsf{Sq}.\langle f_1 \dots f_n \rangle$	sequence of facts
$\mathsf{E}_{pk}(f)$	public key encryption
$\mathsf{D}_{sk}(f)$	public key decryption
$\mathsf{E}_k(f)$	symmetric encryption
$\mathsf{D}_k(f)$	symmetric decryption
$\mathsf{S}_{sk}(f)$	digital signature
$\mathsf{H}(f)$	cryptographic hash
a	agent variable
v	voter variable
c	candidate variable

s	serial number variable
l	candidate list variable
m	message variable
n	nonce variable
i	index variable
$\mathsf{Ind}.i$	vVote index
$\mathsf{Raw}(s, \mathsf{E}_{pk}(f))$	vVote raw ballot
$\mathsf{DigB}(\mathsf{S}_{sk}(s), \mathsf{E}_{pk}(l))$	vVote digital ballot
$\mathsf{B}(l, s, \mathsf{Ind}.i)$	vVote ballot form
$\mathsf{RHS}(s, \mathsf{Ind}.i)$	vVote RHS of a ballot form
$\mathsf{R}(\mathsf{S}_{sk}(\mathsf{RHS}(s, \mathsf{Ind}.i)))$	vVote receipt
$\mathsf{V}(\mathsf{Ind}.i, \mathsf{E}_{pk}(l))$	vVote vote

Acronyms

AKISS Active Knowledge in Security Protocols.

AV alternative vote.

CSP Communicating Sequential Processes.

CVS conventional voting system.

DRE Direct Recording by Electronics.

E2E end-to-end.

EBM electronic ballot marker.

FDR Failures-Divergence Refinement.

FPTP first-past-the-post.

IND-CCA2 indistinguishability under adaptive chosen ciphertext attack.

IND-CPA indistinguishability under chosen plaintext attack.

IRV instant-runoff voting.

LTL linear temporal logic.

LTS labelled transition systems.

NSPK Needham-Schroeder Public-Key.

PAT Process Analysis Toolset.

PGD Pretty Good Democracy.

POD print-on-demand.

RPC randomised partial checking.

SBA short ballot assumption.

STV single transferable vote.

TTP trusted third party.

VEC Victorian Electoral Commission.

VVPAT voter-verified paper audit trail.

WBB web bulletin board.