FORMAL VERIFICATION OF CRYPTOGRAPHIC PROTOCOLS WITH AUTOMATED REASONING

by

BEN SMYTH

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

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
March 2011

Abstract

Cryptographic protocols form the backbone of our digital society. Unfortunately, the security of numerous critical components has been neglected. As a consequence, attacks have resulted in financial loss, violations of personal privacy, and threats to democracy. This thesis aids the secure design of cryptographic protocols and facilitates the evaluation of existing schemes.

Developing a secure cryptographic protocol is game-like in nature, and a good designer will consider attacks against key components. Unlike games, however, an adversary is not governed by *the rules* and may deviate from *expected behaviours*. Secure cryptographic protocols are therefore notoriously difficult to define. Accordingly, cryptographic protocols must be scrutinised by experts using procedures that can evaluate security properties.

This thesis advances verification techniques for cryptographic protocols using formal methods with an emphasis on automation. The key contributions are threefold. Firstly, a definition of election verifiability for electronic voting protocols is presented; secondly, a definition of user-controlled anonymity for Direct Anonymous Attestation is delivered; and, finally, a procedure to automatically evaluate observational equivalence is introduced.

This work enables security properties of cryptographic protocols to be studied. In particular, we evaluate security in electronic voting protocols and Direct Anonymous Attestation schemes; discovering, and fixing, a vulnerability in the RSA-based Direct Anonymous Attestation protocol. Ultimately, this thesis will help avoid the current situation whereby numerous cryptographic protocols are deployed and found to be insecure.

In loving memory of Andrew Cavan Ellis, who sadly died on 11 November 2009, aged 43.

For David Hercules Cash Carroll, Megan Naomi Eleanor Lee, and Alyssia Anne Smyth. May the future be good to you.

Acknowledgements

First, and foremost, I would like to thank my supervisor Mark Ryan. Mark has dedicated a considerable amount of time and energy to the supervision of my research. He has helped me explore ideas, to critically evaluate my work, and to express my results clearly. He has imparted the fundamental skills of our discipline and, more importantly, he has taught me how to further my skills independently. His efforts will be cherished, and memories of my Ph.D. will be held in the highest regard. I look forward to our future collaboration and continued friendship.

I have had the opportunity to work with a number of collaborators and I have learnt a great deal from them all. Liqun Chen taught me the intricacies of cryptography; in particular, Direct Anonymous Attestation. Stéphanie Delaune guided my study of the applied pi calculus. Mounira Kourjieh's boundless energy drove our research. Steve Kremer explained the subtleties of the applied pi calculus and insisted upon mathematical rigour. Bruno Blanchet helped me understand the theoretical underpinnings of ProVerif, and introduced numerous new ideas. Myrto Arapinis insisted that proofs could not begin with QED and patiently taught their proper construction; in addition, her Sushi is divine. Petr Klus helped implement theoretical results as software. The efforts of my collaborators have developed my research skills, for which I am truly grateful.

Birmingham's *Formal Verification and Security Group* has been an excellent source of feedback and I am particularly grateful to: Myrto Arapinis, Sergiu Bursuc, Tom Chothia, Dan Ghica, Eike Ritter, Mark Ryan and Guilin Wang. I have also benefited from the wisdom of Martín Escardó and Eike Ritter who provided academic support as members

of my *research monitoring group*. Mark Lee has provided an endless source of advice on topics as diverse as academic life and culinary masterpieces; and the occasional unsolicited bout on badgers, morris dancing and other such fields. Academic life at Birmingham has been wonderful and many individuals have influenced my stay: thank you Birmingham.

My experience gained through research visits has been hugely beneficial and the hospitality offered by each institution was superb. I am particularly grateful to my hosts: Liqun Chen at HP Labs; Steve Kremer at LSV, CNRS & ENS de Cachan; and Bruno Blanchet at ENS, CNRS & INRIA.

I will receive all the credit for this thesis – I also accept liability for any criticism – but, in addition to those already mentioned, some credit should be attributed to those whom studied early drafts. I am truly grateful to: Myrto Arapinis, Bruno Blanchet, Gavin Brown, Liqun Chen, Michael Clarkson, Véronique Cortier, Cas Cremers, Morten Dahl, Catherine Harris, Lucie Jíchová, Hugo Jonker, Vivek Nallur, James Nestoruk, Olufunmilola Onolaja, Mark Ryan, Winona Ryder, Matt Smart, Martin Smyth, Sue Smyth and Guilin Wang. In particular, Lucie Jíchová is my favourite *stickler*, Mark Ryan is a thorough technical critic, and my mum, Sue Smyth, read every last word.

Finally, I would like to thank my parents – Martin and Sue – for supporting me in everything I have chosen to do; my siblings – Matthew, Joe and Megan – for providing constant entertainment; and Beth Wells for a great friendship and numerous hours spent in *The Barrels*.

This research was primarily funded by the Engineering and Physical Sciences Research Council (EPSRC), under the WINES initiative, as part of the *UbiVal: Fundamental Approaches to Validation of Ubiquitous Computing Applications and Infrastructures* project (EP/D076625/1 & EP/D076625/2). Additional funding was provided by the *Direction Générale pour l'Armement* (DGA) to support a five month internship at CNRS, Département d'Informatique, École Normale Supérieure, Paris, France.

Contents

List of Figures

Part I

Preliminaries

1 Introduction

The security objectives of numerous deployed systems have not been satisfied. As a consequence, high-profile security failures have been observed, for example: an exploit in the London Underground's Oyster card system allows free travel [GHG08, GGM+08, GRVS09]; vulnerabilities in electronic voting systems have resulted in their decommission [UK07, Bow07, Min08, Bun09]; and issues with electronic passports permit modification, cloning and invasions of privacy [JMW05, Jue06, KJBK09, CS10a]. These failures can be attributed to the difficulties of designing secure cryptographic protocols.

Cryptographers aim to deliver protocols which preserve security requirements in the presence of an arbitrary adversary. However, such an adversary should not be expected to follow any prescribed rules, nor adhere to any particular behavioural patterns. Moreover, the ingenuity of the adversary should be considered boundless. As an analogy, consider, for example, the challenge faced by the architects of the *'inescapable'* Alcatraz. The prison is situated 1.5 miles from San Francisco, California, and isolated from the mainland by the strong currents of San Francisco Bay. In addition to the treacherous waters and the physical security features, inmates were counted twelve times a day and the ratio of inmates to armed guards was three to one. Despite this, an elaborate escape plan was hatched by Clarence Anglin, John Anglin, Frank Morris and Allen West which involved the fabrication of: life-like dummies, fashioned from soap, toilet paper and hair (the dummies were used to avoid detection during evening head counts); tunnelling equipment, built from the motor of a stolen vacuum cleaner; and a raft, constructed from the rain coats

of fellow inmates. On June 11, 1962 the two Anglin brothers and Morris escaped the prison. Their whereabouts are currently unknown. Given the water temperature and tidal direction, the official FBI investigation presumes the three men drowned; however, the only thing known for certain, is the security of Alcatraz was violated. Designing a resilient prison to contain such devious inmates, whom are determined to escape, is difficult. In some sense, cryptographers face a more challenging problem: a prison may be considered inescapable if no inmate has successfully broken out; by comparison, we will only consider a protocol to be secure, if security requirements cannot be violated by *any* adversary.

This thesis will aid the secure design of cryptographic protocols and facilitate the evaluation of existing schemes, with a particular focus on electronic voting and anonymous attestation. These application domains are particularly salient, due to the discovery of vulnerabilities in the RSA-based Direct Anonymous Attestation scheme (Chapter 4) and the world-wide loss of public confidence in electronic voting systems following numerous failures [Sal88, Rub02, Mer02, CEC+08].

The remainder of this chapter is structured as follows. Cryptographic protocols will be introduced (Section 1.1) and their security properties will be discussed (Section 1.2). Computational and symbolic approaches for protocol verification will be considered (Section 1.3) and symbolic analysis techniques will be investigated in more depth (Section 1.4). A detailed description of the contribution made by this thesis will be presented in the context of existing work (Section 1.5), a summary of publications will be provided (Section 1.6) and, finally, an overview of the thesis structure will be summarised (Section 1.7).

1.1 Cryptographic protocols

Cryptographic protocols are small distributed algorithms that aim to provide some security-related objective over a public communication network, such as the Internet. Since the communication medium is public, an adversary may interfere with messages;

this introduces the possibility of interference by a powerful adversary and has made the design of secure protocols notoriously difficult. The canonical example is the Needham-Shroeder public key protocol [NS78], which is intended to provide mutual authentication of two principals. The protocol was scrutinised by experts for nearly two decades before Lowe [Low96] discovered a man-in-the-middle attack. This dictates the necessity for protocol verification and, moreover, highlights the need for automated support to overcome the inherent human weaknesses present in the manual verification process.

1.2 Security properties

Cryptographic protocols are required to satisfy a plethora of security requirements. These requirements include classical properties such as secrecy and authentication, and emerging properties including, but not limited to, anonymity, non-repudiation and verifiability. These security requirements can generally be classified as *indistinguishability* or *reachability* properties. Reachability properties are more simplistic, and they are typically used to express requirements of a protocol's derivable states. For example, secrecy can be expressed as the inability of deriving a particular value from all possible protocol executions. The notion of indistinguishability allows us to reason about more complex properties. Intuitively, two protocols are said to be indistinguishability if an observer has no way of telling them apart. Anonymity can be naturally formulated as an indistinguishability property, for example, actions within a group of agents are anonymous if an action performed by an arbitrary agent is indistinguishable from an action performed by another agent.

1.3 Protocol verification

The process of verifying cryptographic protocols has led to the development of two research communities, namely: provable security and formal methods. The techniques of

each field are almost completely disjoint [AR00, PSW00, AR02, War03, War05]: formal methods are reliant on simple abstract modelling techniques, whereas provable security uses complicated computational proofs. With the exception of *perfect security* [Sha49], which is generally unobtainable, neither community can make any absolute mathematical statement about the security of a protocol, and the best we can do is prove the relative difficulty of violating security properties. In particular, we can consider a provable security *reduction*, which shows that the difficulty of breaking the protocol is *at least as difficult* as solving a well-known mathematical problem that is widely believed to be intractable. However, as cryptography has evolved, more complex protocols have been developed and provable security methodologies have, unfortunately, provided limited success in their analysis [Moo88, Mea03, KM06, KM07]. Indeed, Chen, Morrissey & Smart [CMS08a, pp157] attribute flaws in the security proof of RSA-based Direct Anonymous Attestation protocol [BCC04] to the highly complex nature of Direct Anonymous Attestation schemes and the inherent difficulties of composing correct provable security proofs. By comparison, formal method approaches have shown to be particularly suited to the analysis of complex cryptographic protocols, and will be adopted here.

1.3.1 Formal methods

Formal methods, as defined by Meadows [Mea03, §2], combine a language which can be used to model a cryptographic protocol and its security properties, together with an efficient procedure to determine whether a model does indeed satisfy those properties. Gritzalis, Spinellis & Georgiadis [GSG99] classify three types of protocol analysis: *inference-construction*, *attack-construction* and *proof-construction*. Inference-construction techniques are based upon modal logics; they reason about the evolution of knowledge and beliefs within a system to show that certain conditions are satisfied. Attack-construction methodologies attempt to discover vulnerabilities using algebraic properties of a protocol's algorithms. The majority of these attack-construction techniques conduct some form

of state space exploration; that is, a model of the system is defined and then analysed in an attempt to discover a path to an insecure state. Proof-construction techniques model the computations performed in a protocol and define security properties as theorem, automated theorem checkers are then used to verify these theorems and thus prove properties of the model. Gritzalis, Spinellis & Georgiadis [GSG99] liken proof-construction to attack-construction, suggesting that proof-construction techniques replace state space exploration *"with theorems about these searches."*

Meadows argues that inference-construction techniques are generally weaker than attack-construction methods, because they operate at a higher level of abstraction [Mea00, Mea01]; hence, interest in inference-construction has waned, as attack-construction methodologies have improved. Paulson [Pau98] claims that attack-construction and proof-construction techniques are complementary: attack-construction techniques are typically easy to use and can provide an assurance that a model satisfies security criteria; by comparison, proof-construction methodologies are more complicated, but allow a more thorough analysis. A primary objective of this thesis is to introduce tools suited to the verification of security properties during protocol development; accordingly, we focus on attack-construction techniques to allow rapid prototyping of cryptographic schemes.

Dolev-Yao adversary. Attack-construction methods typically assume that cryptographic protocols should achieve their objectives in the presence of an adversary that has full control of the network (sometimes called the Dolev-Yao attacker [DY83]) and cryptography is usually assumed to be perfect. Since the adversary has complete control of the network, messages may be read, modified, deleted, or injected. The adversary is also able to manipulate data contained within those messages, under the restriction of perfect cryptography, that is, the attacker is only able to perform cryptographic operations when in possession of the required keys. It follows, for example, that an adversary may compute the ith element of a tuple or, given the necessary keys, decrypt ciphertexts. The relationships between cryptographic primitives are captured by a set of deduction

rules. For example, an equation modelling symmetric decryption can be expressed as $\mathsf{dec}(k, \mathsf{enc}(k, m)) = m$. This equation captures the notion that symmetric encryption is perfect: given a ciphertext $\mathsf{enc}(k, m)$, the plaintext m can only be recovered using the secret key k. If the encryption scheme has some other characteristic (for example, if the scheme is homomorphic), then it is important that these properties are also captured to avoid missing attacks. For expressibility, we should consider frameworks which permit a large class of cryptographic primitives to be modelled.

1.4 Analysis using formal methods

Verification of cryptographic protocols is difficult due to several sources of unboundedness, including: messages of arbitrary length and the possibility of an unbounded number of sessions (protocol executions). In this section, we will consider methodologies that allow the formulation of precise security statements about cryptographic protocols and the conditions under which these statements hold. (For further discussion see Blanchet [Bla08, §1.4], Delaune [Del11, §1] and Meadows [Mea03, Mea04].)

1.4.1 Reachability properties

Reachability properties are expressed as assertions defined over protocol execution traces and the possibility of arbitrarily many sessions makes analysis particular problematic. One solution is to explore a finite part of the state space and prove partial results using model checkers, for example, FDR [Ros95], Murφ [MMS97], and Brutus [CJM00]. Alternatively, Rusinowitch & Turuani [RT01, RT03] avoid this source of undecidability by restricting honest participants to a bounded number of sessions (the number of messages sent by the adversary, and the length of those messages, remains unbound) and present an NP decision procedure for secrecy properties in the Dolev-Yao model with symmetric and asymmetric cryptography. This result has been enriched to consider more complicated properties of cryptographic primitives, for example, Chevalier *et al.* [CKRT03a, CKRT05]

and Comon-Lundh & Shmatikov [CLS03] present a decision procedure for secrecy in the presence of an adversary that can exploit properties of the XOR operator and Chevalier *et al.* [CKRT03b] consider an adversary that can exploit Diffie-Hellman exponentiation, the complexity of these procedures are shown to be in NP.

Techniques exploring finite state spaces or considering a bounded number of sessions are useful for discovering attacks, but cannot be used to assert the security of a protocol; in particular, there may exist an attack outside the considered search space (for example, a revision of the Garay & Mackenzie contract signing protocol [GM99] was shown to be secure for $n \leq 4$ participants [CKS04, CKS06]; but, Mukhamedov & Ryan [MR08] presented an attack for $n \geq 5$ participants). Henceforth, this limitation will be avoided by considering only techniques that consider all states and an unbounded number of sessions; unfortunately, proving the security of a protocol in this context is undecidable [DLMS99, DLMS04]. Accordingly, we will consider sound but incomplete techniques and as a consequence we will accept methodologies that may not terminate or may report false attacks.

Lowe [Low96, Low99] demonstrated that limitations of finite state exploration can be overcome by introducing manual proofs that show such results are sufficient; in particular, Lowe proved the security of the Needham-Shroeder-Lowe public key protocol in this context by manually proving that if the revised scheme was vulnerable to an attack, then the attack would be discovered in the finite model. Ideally, we would like to avoid such manual proofs. Meadows [Mea94, Mea96b, EMM05, EMM06] considers an unbounded number of sessions by applying sound user-guided pruning to derive a tractable model from an infinite state space, these results have been implemented in the NRL Protocol Analyzer. (See Meadows [Mea96a] for a comparison between the NRL Protocol Analyzer and Lowe's approach using FDR.) Thayer, Herzog & Guttman [THG98, THG99] introduce *strand spaces* which allow more compact formalisations of state and Song, Berezin & Perrig [Son99, SBP01] propose a verification procedure which has been implemented as Athena, this tool also reduces the state space by pruning. Both the NRL Protocol

Analyzer and Athena may fail to terminate for an unbounded number of sessions and, based upon Athena, Cremers [CM05, Cre06, Cre08b, Cre08a] proposes the Scyther tool which guarantees termination, but may only analyse a bounded number of sessions when a result cannot be obtained in the unbounded case. Meier, Cremers & Basin [MCB10] extend Scyther to generate correctness proofs which can be automatically checked using theorem provers, this exploits the strengths of proof-construction techniques to provide stronger results.

Bolignano [Bol97] proposes an abstract model which over-approximates the adversary's power. This has proven to be useful for approaches using tree automata [Mon99, Mon03, GK00] and, more generally, techniques based upon resolution of Horn clauses [Wei99]. (Tree automata can be encoded as Horn clauses [FSVY91].) Boichut, Héam & Kouchnarenko [BHKO04, BHK05, BHK06, BHK09] extend the tree automata reasoning technique proposed by Genet & Klay [GK00] and implement their results in the TA4SP verification tool. Blanchet [Bla01, Bla02, Bla09, Bla11] uses resolution of Horn clauses to reason with reachability properties in processes of the applied pi calculus [AF01] and these results have been implemented in ProVerif. These over-approximation techniques have proven to be successful, despite the possibility of reporting false attacks and non-termination.

1.4.2 Indistinguishability properties

Indistinguishability properties are captured using equivalence [MPW92a, MPW92b, AG97, AF01]. Equivalence has been studied independently of cryptography in the pi calculus [San93, Vic94, Dam95, San96, Dam97] and automated tools have been proposed, for example, Another Bisimulation Checker [Bri05] and the Mobility Workbench [VM94]. In the context of cryptographic protocols, Abadi & Gordon [AG97] introduced the spi calculus to study equivalence. The spi calculus is restricted to a basic set of cryptographic primitives and this framework has been extended to the applied pi calculus by Abadi &

Fournet [AF01] to overcome this limitation.

Definitions of equivalence usually quantify over all possible adversaries and this makes proofs difficult. It is therefore useful to introduce alternative definitions that coincide and prove results in the alternative setting; *labelled bisimilarity* [AG98a, AG98b, BN02, BN05] has been particular useful for this purpose in the spi calculus. In the applied pi calculus, Abadi & Fournet [AF01] claimed that their definitions of equivalence and labelled bisimilarity coincide; however, a formal proof of this claim has not been published, and additional assumptions are known to be necessary [AF06, BJPV09, ARR10]. Accordingly, the use of labelled bisimilarity for proofs of equivalence in the applied pi calculus should be avoided. We will now review decision procedures for equivalence.

Durante, Sisto & Valenzano [DSV00, DSV03] present an decision procedure for equivalence which considers a bounded number of sessions in the spi calculus. In the applied pi calculus, Abadi & Cortier [AC04a, AC05a, AC06] introduce a decision procedure for *static equivalence* (that is, in the presence of a passive adversary with a bounded number of sessions) in the context of *subterm convergent equational theories*; this result has been exploited by Baudet, Cortier & Delaune [BCD09, BCD10] and Ciobâcă, Delaune & Kremer [CDK09b] to provide automated tools (namely, YAPA and KISS) for the analysis of static equivalence. Baudet [Bau05] extends the work of Abadi & Cortier to consider an active adversary for a bounded number of sessions. As discussed in the context of reachability properties, such techniques are useful for finding attacks but cannot guarantee the absence of attacks and, henceforth, we consider methodologies that analyse all states. Unfortunately, proving equivalence for an unbounded number of sessions is an undecidable problem [Han03, AC04a, AC06] and we will therefore consider sound but incomplete decision procedures.

Blanchet, Abadi & Fournet [Bla04, BAF08] focus on equivalences between pairs of processes which share the same structure and differ only in the choice of terms. They introduce the notion of *uniformity*, a sufficient condition for observational equivalence, and automated reasoning is supported by ProVerif. However, the notion of uniformity is

often too strict for the equivalences we wish to consider and the technique may report false attacks.

The potentially infinite number of execution traces due to messages sent by the adversary makes reasoning about equivalence difficult. In a variant of the spi calculus, Borgström, Briais & Nestmann [BBN04] avoid this problem by treating each message from the adversary as a variable (thereby ensuring that a message input is mapped to a single transition) and similar results are obtained by Delaune, Kremer & Ryan [DKR07, DKR10a] for the applied pi calculus. Both Borgström, Briais & Nestmann and Delaune, Kremer & Ryan present decision procedures for fragments of the original calculi, for example, Delaune, Kremer & Ryan do not consider disequalities (that is, they do not consider else-branches in conditional statements). In a different direction, Cortier & Delaune [CD09] have shown that equivalence coincides with trace equivalence for *determinate processes* (based upon [Bau05], this yields a decision procedure for a bounded number of sessions).

1.4.3 Automated verification tools

The inability of experts to identify the flaw in the Needham-Shroeder public key protocol highlights inherent weaknesses in the manual verification process and necessitates automated analysis tools. We will now summarise the capabilities of the state-of-the-art tools introduced in the previous section, with the notable omission of the NRL Protocol Analyzer which is not publicly available.

AVISPA [ABB+05] AVISPA provides a common interface for a number of analysis tools including TA4SL [BHKO04, BHK05, BHK06, BHK09]. AVISPA/TA4SL is capable of evaluating reachability properties for an unbounded number of sessions. (AVISPA also provides support for CL-Atse [Tur06], OFMC [BMV03, BMV05], and SATMC [AC04b, Com05, AC05b]; but, these tools are limited to a finite number of sessions.)

ProVerif [BS11] ProVerif is capable of evaluating reachability properties [Bla01, AB02, Bla02, AB05a, Bla09, Bla11] and observational equivalence [Bla04, BAF05, BAF08], in the context of an unbounded number of sessions. The tool is capable of producing attack traces [AB05c]: when a property cannot be proved, an execution trace which falsifies the desired property is constructed. Support is provided for user-defined equations and the resolution algorithm has been proven to terminate for tagged protocols [BP03, BP05]

Scyther [Cre07] Scyther [CM05, Cre06, Cre08b, Cre08a] is capable of evaluating reachability properties for an unbounded number of sessions; moreover, if a result cannot be obtained (due to the undecidable nature of the problem), then security of a bounded number of sessions is evaluated. It follows that termination is guaranteed. The tool is also capable of generating correctness proofs suitable for theorem checkers [MCB10] and producing graphical attack traces.

The AVISPA tool is particularly well-suited to the analysis of security properties for a bounded number of sessions (using OFMC, CL-Atse, and SATMC), but TA4SL provides only limited support in the unbounded case. For example, over-approximation coupled with the absence of attack traces is particularly problematic in AVISPA/T4SL, because it is difficult to distinguish between false attacks caused by over-approximation and real attacks [CLN09]. ProVerif provides support for equivalence, in addition to reachability, although due to over-approximation it may report false attacks. ProVerif also supports user-defined equations and the following (non-exhaustive list of) cryptographic primitives can be modelled: symmetric and asymmetric cryptography; digital signatures; hash functions; bit-commitment; and signature proofs of knowledge. Scyther does not over-approximate and is therefore able to avoid reporting false attacks; moreover, unlike ProVerif, Scyther is guaranteed to terminate and produce a result (albeit, possibly only for a bounded number of sessions). However, Scyther does not support user-defined equational theories nor can control flow be modelled and these factors limit the class of protocols that can be

modelled. Table 1.1 summarises the capabilities of these tools. (For further comparison between tools see [CLN09, PBP+10, LTV10, DSHJ10].)

	AVISPA/TA4SL	ProVerif	Scyther
Reachability	✓	✓	✓
Equivalence	✗	✓	✗
User-defined theory	✗	✓	✗
Attack traces	✗	✓	✓
No false attacks	✗	✗	✓
Termination	✗	✗	✓

Table 1.1: A comparison of automated verification tools

Applied pi calculus and ProVerif. The applied pi calculus provides a framework for analysing reachability and equivalence properties with automated support provided by ProVerif. The support for user-defined equations allows a large class of cryptographic protocols to be modelled and our discussion demonstrates ProVerif is a leading tool. Moreover, the applied pi calculus and ProVerif have been successfully used to model cryptographic protocols from a variety of application domains and some applications are listed below. On this basis, the applied pi calculus and ProVerif will be adopted by this thesis.

- Abadi & Blanchet [AB05b] use correspondence assertions to verify certified email [AGHP02].
- Abadi, Blanchet & Fournet [ABF07] analyse the JFK (Just Fast Keying) [ABB+04] protocol – which was one of the candidates to replace IKE as the key exchange protocol in IPSec – using correspondence assertions and equivalence properties in a combination of manual and automated proofs.
- Blanchet & Chaudhuri [BC08] study the integrity of the Plutus file system [KRS+03] on untrusted storage using correspondence assertions, resulting in the discovery and subsequent fixing of weaknesses in the initial system.

- Bhargavan *et al.* [BFGT06, BFG06, BFGS08, BFGT08] use ProVerif to analyse cryptographic protocol implementations written in F#; in particular, the Transport Layer Security (TLS) protocol has been studied in this manner [BCFZ08].

- Chen & Ryan [CR09] have evaluated authentication protocols found in trusted computing, and have discovered vulnerabilities.

- Delaune, Kremer & Ryan [KR05, DKR06, DKR09, DKR10b] and Backes, Hriţcu & Maffei [BHM08] formalise and analyse privacy properties for electronic voting using equivalence.

The application of formal method techniques are limited by: 1) the absence of rigorously defined security properties [Oka98, Bel99, Aba00, Mea04]; and 2) the lack of efficient procedures to evaluate security properties [BM92, Rus93, GSG99]. This thesis will advance the applied pi calculus and ProVerif in this context.

1.5 Contribution and literature review

The objective of this thesis is to aid the secure design of cryptographic protocols and facilitate the evaluation of existing schemes. This is achieved in the context of the applied pi calculus by defining the security properties of protocols (Part II) and developing efficient procedures for the automated analysis of observational equivalence (Part III). Accordingly, this thesis will help avoid the current situation whereby numerous protocols are deployed and subsequently found to be insecure; ultimately helping to prevent attacks which cause financial loss, violations of personal privacy and threats towards democracy. The applicability of the methodologies and techniques introduced by this thesis are demonstrated throughout by analysing cryptographic protocols. A detailed description of the key chapters will now be presented.

Chapter 3

Electronic voting systems lack the transparency offered by their paper-based counterparts. For example, in paper-based systems, the whole process, from ballot casting to tallying, can be observed; by contrast, it is not possible to observe electronic operations performed on data. Some electronic voting systems attempt to avoid this limitation by assuming the hardware and software running the election is trusted. In practice, this level of trust is very difficult to achieve and thus, systems based upon such assumptions have failed [Bun09, Min08, Bow07, UK07]. A better approach is to provide *verifiable* voting systems (for example, [JCJ02, JCJ05, CRS05, Adi06, Dag07, Adi08]), that allow voters and election observers to check that votes have been recorded, tallied and declared correctly. Moreover, these checks can be performed in a manner independent of the election system's hardware and software. In this chapter, a definition of election verifiability is presented.

In related work, Juels, Catalano & Jakobsson [JCJ02, JCJ05, JCJ10] present a definition of universal verifiability in the provable security model. Their definition assumes voting protocols produce non-interactive zero-knowledge proofs of knowledge (also called signature proofs of knowledge) demonstrating the correctness of tallying. Here we consider a definition in the formal model. Universal verifiability was also studied by Chevallier-Mames *et al.* [CMFP$^+$06, CMFP$^+$10]. They show an incompatibility result: protocols cannot satisfy verifiability and vote privacy in an unconditional way (that is, without reliance on computational assumptions). But, as witnessed by [JCJ02, JCJ05], weaker versions of these properties can hold simultaneously; furthermore, our case studies demonstrate that privacy and verifiability can coexist. Baskar, Ramanujam & Suresh [BRS07] formalise individual verifiability, and Talbi *et al.* [TMT$^+$08] consider individual and universal verifiability. Both of these definitions are tightly coupled with the protocol by Fujioka, Okamoto & Ohta [FOO92] and cannot easily be generalised. Moreover, Talbi *et al.* characterise individual execution traces as verifiable or not, and it is unclear how their definition can be used to assert that a protocol satisfies verifiability (that is, every

execution of the protocol is verifiable); in particular, the authors only show that a single execution of the protocol by Fujioka, Okamoto & Ohta satisfies verifiability, and do not make any claims about the verifiability of the protocol.

In complimentary work, Backes, Hriţcu & Maffei [BHM08] formalise correctness properties (including inalterability and eligibility) for honest protocol executions; if an electronic voting scheme satisfies their definition and the participants execute the protocol as intended, then no one can change a voter's vote and only registered voters can vote and at most once. However, their definition provides no assurances when participants deviate from the protocol (that is, when participants do not perform honest executions). It follows that a scheme can be correct, but not provide the intended security properties; this may occur, for example, because a dishonest administrator chooses to run a malicious variant of the protocol. In this respect, a verifiable electronic voting protocol is more desirable because voters and observers can check that the correctness properties hold, even in the presence of dishonest administrators. In addition to verifiability and correctness properties, electronic voting protocols are expected to satisfy privacy properties (including ballot secrecy, receipt freeness, and coercion resistance) and fairness (including ballot independence and no early results). Table 1.2 summarises these properties and highlights several definitions of security properties that have emerged in the literature.

We present a generic definition of election verifiability in terms of boolean tests. These tests are required to satisfy several conditions for all possible execution traces. This facilitates the evaluation of verifiability in electronic voting protocols. The definition also allows identification of the hardware and software components that must be trusted for the purpose of election verifiability, thereby facilitating the comparison of electronic voting protocols on the basis of trust assumptions. The framework is compatible with a large class of electronic voting schemes, including those based upon blind signatures, homomorphic encryption and mixnets. Moreover, the definition is used to analyse verifiability in three electronic voting protocols, two of which have been implemented and deployed. In particular, the Helios 2.0 protocol is evaluated; this scheme is particularly significant due

Verifiability	*Individual verifiability:* a voter can check that her own ballot is published on the election's bulletin board.	[JCJ05, BRS07, TMT+08]
	Universal verifiability: anyone can check that all the votes in the election outcome correspond to ballots published on the election's bulletin board.	[JCJ05, CMFP+06, TMT+08]
	Eligibility verifiability: anyone can check that each ballot published on the bulletin board was cast by a registered voter and at most one ballot is tallied per voter.	[JCJ05]
Correctness	*Declared result:* the election outcome is the correct sum of the votes cast.	[BHM08]
	Eligibility: only registered voters can vote and at most once.	[JCJ05, KR05, TMT+08, BHM08]
	Inalterability: no one can change a voter's vote.	[JCJ05, BHM08]
Privacy	*Ballot secrecy:* A voter's vote is not revealed to anyone.	[KR05, DKR06, CMFP+06, BRS07, BHM08, DKR09, CDK09a, JMP09]
	Receipt freeness: A voter cannot gain information which can be used to prove, to a coercer, how she voted.	[DKR06, MN06, JV06, CMFP+06, BRS07, BHM08, DKR09, KT09, JMP09]
	Coercion resistance: A voter cannot collaborate, with a coercer, to gain information which can be used to prove how she voted.	[JCJ05, DKR06, MN06, BHM08, DKR09, KT09, KTV10, UMQ10]
Fairness	*Ballot independence:* voters cannot cast related votes.	[Gen95, Gen00]
	No early results: partial election results, which could alter voter behaviour (e.g., by influencing a voter's decision or prompting a voter to pull-out), are unavailable.	[KR05, BRS07, TMT+08]

Table 1.2: Summary of electronic voting properties and security definitions

to its real-world deployment: the International Association of Cryptologic Research used Helios to elect its board members [BVQ10], following a successful trial in a non-binding poll [HBH10]; the Catholic University of Louvain adopted the system to elect the university president [AMPQ09]; and Princeton University used Helios to elect the student vice president [Pri10]. This demonstrates the suitability of the framework for analysing real-world election systems.

Chapter 4

Trusted computing has introduced a hardware device, called a Trusted Platform Module, to allow systems to provide cryptographic guarantees about their state. The hardware chip contains a unique key to root trust and, as a consequence, cryptographic operations may reveal a platform's identity. This raises privacy concerns. Brickell, Camenisch & Chen [BCC04] overcome this privacy issue with the introduction of Direct Anonymous Attestation (DAA): a remote authentication scheme for trusted platforms which provides user-controlled anonymity and traceability. In this chapter, a definition of user-controlled anonymity is presented and used to analyse the RSA-based DAA scheme [BCC04], discovering a vulnerability.

In related work, Brickell, Camenisch & Chen [BCC04] and Brickell, Chen & Li [BCL08b, BCL09] present definitions in the provable security model; the relationship between these models is unknown [CMS08a, pp158]. Here we consider a definition in the formal model, based upon informal descriptions of user-controlled anonymity [BCL08b, BCL09]. By comparison, Backes, Maffei & Unruh [BMU08] formalised an earlier notion of user-controlled anonymity (informally described in [BCC04]) for their model of the RSA-based DAA protocol. This formalisation can probably be generalised, but it pre-dates the definition of user-controlled anonymity by Brickell, Chen & Li [BCL08b, BCL09] and considers a conceptually weaker adversary. Rudolph [Rud07] also proposes an attack against RSA-based DAA which violates privacy; however, this vulnerability has been discounted by Leung, Chen & Mitchell [LCM08] as infeasible for large-scale violations of privacy and,

moreover, can be detected in the presence of a single honest verifier.

We present a generic definition of user-controlled anonymity as an equivalence property suited to automated reasoning using ProVerif. The definition is used to analyse anonymity in the RSA-based DAA protocol [BCC04]. This scheme is particularly significant, as the TPM specification version 1.2 [TCG07] (which has been defined as an ISO/IEC international standard [Int09]) mandates support for RSA-based DAA and, moreover, estimates suggest the TPM has been embedded in over 300 million computers [Tru09] (although, some experts claim only 5% of these TPMs have been turned on [Mar08, §6]). The analysis discovers a vulnerability which can be exploited by a passive adversary and, under weaker assumptions, by corrupt administrators. This demonstrates the suitability of the framework for analysing DAA schemes which have been deployed. Finally, a fix is identified and the revised scheme is shown to satisfy user-controlled anonymity.

Chapter 5

Formal method techniques require efficient procedures for evaluating security properties. Moreover, automated reasoning is highly desirable to avoid errors associated with hand-written proofs. In this chapter, a new procedure for automatically verifying observational equivalence will be introduced, with particular emphasis on proving privacy properties.

As previously discussed, uniformity is too strong for the equivalences we wish to consider and this chapter will introduce a weaker condition (see Section 1.4.2 for a summary of related work). In addition, the applied pi calculus is extended to allow modelling of barrier synchronisation. The study of equivalences in the context of synchronisation is particularly interesting, because certain security properties can only be realised if participants synchronise their actions in a specific manner. For example, privacy preserving protocols (including electronic voting, vehicular ad-hoc networks, and anonymity networks) make such assumptions. These results are implemented in the tool *ProSwapper*, an extension of ProVerif, and the applicability of the tool is demonstrated by analysing vote privacy in electronic voting protocols and privacy in vehicular ad-hoc networks.

1.6 List of publications

This thesis is partly based upon the following publications:

[RS11]: Mark D. Ryan and Ben Smyth. Applied pi calculus. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, chapter 6. IOS Press, 2011.

[KRS10]: Steve Kremer, Mark D. Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *ESORICS'10: 15th European Symposium on Research in Computer Security*, volume 6345 of *LNCS*, pages 389–404. Springer, 2010.

[SRKK10]: Ben Smyth, Mark D. Ryan, Steve Kremer, and Mounira Kourjieh. Towards automatic analysis of election verifiability properties. In *ARSPA-WITS'10: Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*, volume 6186 of *LNCS*, pages 165–182. Springer, 2010.

[DRS08]: Stéphanie Delaune, Mark D. Ryan, and Ben Smyth. Automatic verification of privacy properties in the applied pi-calculus. In *IFIPTM'08: 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security*, volume 263 of *International Federation for Information Processing (IFIP)*, pages 263–278. Springer, 2008.

[SRC07]: Ben Smyth, Mark D. Ryan, and Liqun Chen. Direct Anonymous Attestation (DAA): Ensuring privacy with corrupt administrators. In *ESAS'07: 4th European Workshop on Security and Privacy in Ad hoc and Sensor Networks*, volume 4572 of *LNCS*, pages 218–231. Springer, 2007.

1.7 Overview and structure

An overview of this thesis is outlined below.

Part I. Presents introductory and background material.

Chapter 1. Introduces this thesis and describes the research problem.

Chapter 2. Recalls the applied pi calculus which forms the foundation of this thesis.

Part II. Formalises symbolic security definitions for electronic voting and anonymous attestation.

Chapter 3. Presents a definition of election verifiability for electronic voting protocols. This definition is used to analyse election verifiability in the FOO, Helios and JCJ-Civitas electronic voting schemes.

Chapter 4. Delivers a definition of user-controlled anonymity for Direct Anonymous Attestation protocols. This definition is used to discover vulnerabilities in the RSA-based Direct Anonymous Attestation scheme.

Part III. Develops procedures for evaluation of security properties, with an emphasis on automation.

Chapter 5. Defines an automated reasoning procedure for observational equivalences with a focus on protocols in which participants synchronise their actions. This technique is used to analyse vote privacy in the FOO electronic voting scheme and privacy in the CMIX vehicular ad hoc networking protocol.

Part IV. Presents an evaluation.

Chapter 6. Considers further work and presents a conclusion.

2

Background: Applied pi calculus[†]

The applied pi calculus [AF01, RS11] is a language for describing and analysing cryptographic protocols. It provides an intuitive process syntax to describe the actions of the participants in a protocol, emphasising their communication. The syntax is coupled with a formal semantics to allow reasoning about protocols. The language is based upon the pi calculus with the addition of a rich term algebra to enable modelling of the cryptographic operations used by protocols. In this respect, the applied pi calculus also has similarities with the spi calculus [AG97]. The key difference concerns the way in which cryptographic primitives are handled. The spi calculus has a fixed set of primitives built in (namely, symmetric and public key encryption), while the applied pi calculus allows a wide variety of more complex primitives (including, for example, non-deterministic encryption, digital signatures, and proofs of knowledge) to be defined by means of an equational theory. The applied pi calculus permits formal modelling of properties including: reachability, correspondence and observational equivalence. In the context of cryptographic protocols, these properties are particularly useful, since they allow the analysis of traditional security goals such as secrecy and authentication. Moreover, emerging properties such as privacy, traceability and verifiability can also be considered.

[†]This chapter is a compressed variant of [RS11].

Figure 2.1 Syntax for terms

$L, M, N, T, U, V ::=$	terms
$a, b, c, \ldots, k, \ldots, m, n, \ldots, s$	name
x, y, z	variable
$f(M_1, \ldots, M_l)$	function application

2.1 Syntax and informal semantics

The calculus assumes an infinite set of names, an infinite set of variables, and a signature Σ consisting of a finite set of function symbols, each with an associated arity. A function symbol with arity 0 is a constant. Terms are built by applying function symbols to names, variables and other terms; as shown in Figure 2.1, where f ranges over the functions of Σ and l is the arity of f. We use metavariables u, w to range over both names and variables. Tuples $u_1, \ldots, u_l$ and $M_1, \ldots, M_l$ are occasionally abbreviated $\tilde{u}$ and $\tilde{M}$. We write $\{^M/_x\}$ for the *substitution* that replaces the variable x with the term M. Similarly, we write $\{^m/_n\}$ for the substitution that replaces the name n with the name m. Arbitrarily large substitutions can be written as $\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}\}$ or $\{^{\tilde{M}}/_{\tilde{x}}\}$. The letters σ and τ range over substitutions. We write $N\sigma$ for the result of applying σ to the variables (or names) of term N. A term is ground when it does not contain variables.

We assume a type system for terms generated by a set of base types $\mathcal{S}$, which includes the universal type Data. In addition, if ω is a type, then $\mathsf{Channel}\langle\omega\rangle$ is a type too. Formally, the set of types generated by the base types $\mathcal{S}$ is the smallest set Ω satisfying: 1) $\mathcal{S} \subseteq \Omega$, and 2) if $\omega \in \Omega$ then $\mathsf{Channel}\langle\omega\rangle \in \Omega$. Names and variables can have any type. By convention we use a, b, c for channel names, k, s as names of base type and m, n for names of any type. A channel of type $\mathsf{Channel}\langle\omega\rangle$ may communicate messages of type ω. For simplicity, function symbols can only be applied to, and return, terms of base type. We always assume that terms are well typed and that substitutions preserve types.

The syntax for *processes* is presented in Figure 2.2. The null process 0 does nothing; $P \mid Q$ is the parallel composition of processes P and Q; replication $!P$ is the infinite

Figure 2.2 Syntax for processes

$P, Q, R ::=$	processes (or plain processes)
0	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu\, n.P$	name restriction
if $M = N$ then P else Q	conditional
$u(x).P$	message input
$\overline{u}\langle M\rangle.P$	message output

composition $P \mid P \mid \ldots$; and name restriction $\nu\, n.P$ binds n inside P. The conditional if $M = N$ then P else Q is standard, but we stress $M = N$ represents equality (modulo an equational theory) rather than strict syntactic identity. For convenience, we abbreviate conditionals as: if $M = N$ then P, when Q is the null process. Finally, communication is captured by message input and message output. The process $u(x).P$ awaits a message from channel u and then behaves as P, with the received message bound to the variable x. The process $\overline{u}\langle M\rangle.P$ is ready to send M on channel u and then run P. In both of these cases, we may omit P when it is 0. We write $P\{^M/_x\}$ for P with all free occurrences of x replaced by M and, in such a substitution, we insist that no name n occurring in M becomes bound by a restriction $\nu\, n$ occurring in P; sometimes we write let $x = M$ in P in place of $P\{^M/_x\}$.

Bracketing must be used to avoid ambiguities in the way processes are written down. For example, the process $!\, P \mid Q$ might be interpreted as $(!P) \mid Q$ or as $!\,(P \mid Q)$. To avoid too much bracketing, we adopt conventions about the precedence of process operators. Unary operators $!$, $\nu\, n$, $u(x)$, and $\overline{u}\langle M\rangle$ bind more closely than binary operators, and the binary if-then-else operator binds more closely than the binary operator $|$.

The expression $P \mid Q \mid R$ is also ambiguous, since it could mean either $(P \mid Q) \mid R$ or $P \mid (Q \mid R)$. However, we will later see that these processes are semantically identical, so we tolerate the ambiguity in the syntax. Another possible ambiguity arises because of the convention of omitting "else 0" in the if-then-else construct; consequently, it is

Figure 2.3 Syntax for extended processes

$A, B, C ::=$	extended processes
P	plain process
$A \mid B$	parallel composition
$\nu\, n.A$	name restriction
$\nu\, x.A$	variable restriction
$\{^M/_x\}$	active substitution

not clear which "if" the "else" applies to in the expression: if $M = N$ then if $K = L$ then Q else R. In absence of brackets indicating the contrary, we adopt the convention that the else branch belongs to the closest if and hence the statement should be interpreted as if $M = N$ then (if $K = L$ then Q else R).

Processes are extended with *active substitutions* to capture the knowledge exposed to the adversarial environment (Figure 2.3). The active substitution $\{^M/_x\}$ represents a process that has previously output M. The value M is now available to the environment by reference to the 'handle' x. The active substitution $\{^M/_x\}$ can replace the variable x for the term M in every process it comes into contact with. This behaviour can be controlled by restriction, and the process $\nu\, x.(\{^M/_x\} \mid P)$ corresponds exactly to: let $x = M$ in P. This allows access to terms which the environment cannot construct. Arbitrarily large active substitutions can be obtained by parallel composition, and we occasionally abbreviate $\{^{M_1}/_{x_1}\} \mid \ldots \mid \{^{M_l}/_{x_l}\}$ as $\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}\}$ or $\{^{\tilde{M}}/_{\tilde{x}}\}$. We also use σ and τ to range over active substitutions and we write $N\sigma$ for the result of applying σ to the variables of N. Extended processes must have at most one active substitution for each variable, and there is exactly one when the variable is under restriction. Finally, we write $\nu\, \tilde{u}$ for the (possibly empty) series of pairwise-distinct binders $\nu\, u_1.\nu\, u_2.\cdots.\nu\, u_l$.

The type system for terms is extended to processes. It enforces: M, N are of the same type in the conditional expression if $M = N$ then P else Q; message input $u(x)$ is defined only where u is of type $\mathsf{Channel}\langle\omega\rangle$ and x is of type ω; similarly, message output $\overline{u}\langle M\rangle$ requires u of type $\mathsf{Channel}\langle\omega\rangle$ and M of type ω. Substitutions are always assumed to be cycle-free. In addition, active substitutions $\{^M/_x\}$ are such that the term M and variable

x have the same base type; this assumption was not explicitly stated in [AF01] but its necessity has been confirmed [AF06] and is essential for the validity of [AF01, Theorem 1] as shown in [BJPV09]. Finally, we assume extended processes are well-typed.

The *scope* of names and variables are delimited by binders $u(x)$ and $\nu\, u$. The set of bound names $\mathrm{bn}(A)$ contains every name n which is under restriction $\nu\, n$ inside A. The set of bound variables $\mathrm{bv}(A)$ consists of all those variables x occurring in A that are bound by a restriction $\nu\, x$ or input $u(x)$. We also define the set of free names and the set of free variables. The set of free names in A, denoted $\mathrm{fn}(A)$, consists of those names n occurring in A not in the scope of the binder $\nu\, n$. The set of free variables $\mathrm{fv}(A)$ contains the variables x occurring in A which are not in the scope of a restriction $\nu\, x$ or input $u(x)$. We occasionally write $\mathrm{fn}(M)$, and $\mathrm{fv}(M)$, for the set of names, and respectively variables, which appear in term M. An extended process is *closed* when every variable x is either bound or defined by an active substitution. Bound names and bound variables are subject to α-renaming.

A *frame*, denoted φ or ψ, is an extended process built from the null process 0 and active substitutions $\{^{M}/_{x}\}$, which are composed by parallel composition and restriction. The *domain* $\mathrm{dom}(\varphi)$ of a frame φ is the set of variables that φ exports; that is, the set of variables x for which φ contains an active substitution $\{^{M}/_{x}\}$, where x is not under restriction. Every extended process A can be mapped to a frame $\varphi(A)$ by replacing every plain process in A with 0. The frame $\varphi(A)$ represents the static knowledge that is output by a process to its environment. The domain $\mathrm{dom}(A)$ of A is the domain of $\varphi(A)$.

2.2 Operational semantics

The signature Σ is equipped with an equational theory E, that is, a set of equations of the form $M = N$, where the terms M, N are defined over the signature Σ (sometimes written $M, N \in T_{\Sigma}$). This allows us to capture relationships between primitives defined in Σ. We define equality modulo the equational theory, written $=_E$, as the smallest

equivalence relation on terms that contains E and is closed under application of function symbols, substitution of terms for variables and bijective renaming of names. We write $M =_E N$ when the equation $M = N$ is in the theory E, and keep the signature implicit. When E is clear from the context, we may abbreviate $M =_E N$ as $M = N$. The negation of $M =_E N$ is denoted $M \neq_E N$ (and similarly abbreviated $M \neq N$). For further discussion on equational theories see Abadi & Fournet [AF01, §3] and Abadi & Cortier [AC04a, AC05a, AC06].

Contexts may be used to represent the adversarial environment in which a process is run; that environment provides the data that the process inputs, and consumes the data that it outputs. We define *context* $C[\_]$ to be an extended process with a hole. We obtain $C[A]$ as the result of filling $C[\_]$'s hole with the extended process A. An *evaluation context* is a context whose hole is not in the scope of a replication, a conditional, an input, or an output. A context $C[\_]$ closes A when $C[A]$ is closed.

2.2.1 Structural equivalence

Informally, two processes are structurally equivalent if they model the same thing, but the grammar permits different encodings. For example, to describe a pair of processes A, B running in parallel, the grammar forces us to put one on the left and one on the right; that is, we have to write either $A \mid B$, or $B \mid A$. These two processes are said to be structurally equivalent. Formally, *structural equivalence* ($\equiv$) is the smallest equivalence relation on extended processes that is closed by α-conversion of both bound names and bound variables, closed under application of evaluation contexts, and which satisfies the rules in Figure 2.4. The rules for parallel composition, replication and restriction are self-explanatory. Alias enables the introduction of an arbitrary active substitution with restricted scope. Subst describes the application of an active substitution to a process that it comes into contact with. The final rule, Rewrite, allows terms that are equal modulo the equational theory to be swapped as desired.

Figure 2.4 Semantics for processes

PAR-0	A	$\equiv$	$A \mid 0$
PAR-A	$A \mid (B \mid C)$	$\equiv$	$(A \mid B) \mid C$
PAR-C	$A \mid B$	$\equiv$	$B \mid A$
REPL	$!P$	$\equiv$	$P \mid !P$
NEW-0	$\nu\, n.0$	$\equiv$	0
NEW-C	$\nu\, u.\nu\, w.A$	$\equiv$	$\nu\, w.\nu\, u.A$
NEW-PAR	$A \mid \nu\, u.B$	$\equiv$	$\nu\, u.(A \mid B)$ where $u \notin \mathrm{fv}(A) \cup \mathrm{fn}(A)$
ALIAS	$\nu\, x.\{^M/_x\}$	$\equiv$	0
SUBST	$\{^M/_x\} \mid A$	$\equiv$	$\{^M/_x\} \mid A\{^M/_x\}$
REWRITE	$\{^M/_x\}$	$\equiv$	$\{^N/_x\}$ where $M =_E N$

COMM	$\overline{c}\langle x\rangle.P \mid c(x).Q \to P \mid Q$
THEN	if $N = N$ then P else $Q \to P$
ELSE	if $L = M$ then P else $Q \to Q$ for ground terms L, M such that $L \neq_E M$

Structural equivalence allows every closed extended process A to be rewritten as a substitution and a closed plain process with some restricted names: $A \equiv \nu\, \tilde{n}.(\{^{\tilde{M}}/_{\tilde{x}}\} \mid P)$, where $\mathrm{fv}(\tilde{M}) = \mathrm{fv}(P) = \emptyset$ and $\tilde{n} \subseteq \mathrm{fn}(\tilde{M})$. It follows immediately that every closed frame φ can be rewritten as a substitution with some restricted names: $\varphi \equiv \nu\, \tilde{n}.\{^{\tilde{M}}/_{\tilde{x}}\}$, where $\mathrm{fv}(\tilde{M}) = \emptyset$ and $\tilde{n} \subseteq \mathrm{fn}(\tilde{M})$. We note that the domain of φ is $\tilde{x}$.

2.2.2 Internal reduction

A process can be executed without contact with its environment, either because if-statements are evaluated and the then- or else-branch is taken, or because internal subprocesses communicate with each other. The execution of a process with respect to control flow and communication is captured by *internal reduction*. Formally, internal reduction ($\to$) is the smallest relation on extended processes closed under structural equivalence and application of evaluation contexts, satisfying the rules of Figure 2.4. We write

$\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$. Communication (Comm) is defined on variables, making it look rather restricted. However, this entails no loss of generality because Alias and Subst can be used to allow communication of an arbitrary term M instead of a variable x. Conditionals (Then and Else) are dependent on the equational theory. Applications of Then may require the use of the structural equivalence rule Rewrite to derive "$N = N$" from "$M = N$" where $M =_E N$. Else may require that active substitutions in the context be applied using Alias and Subst to ensure L, M are ground.

2.2.3 Labelled reduction

The semantics in Section 2.2.2 allow us to reason about protocols with an adversary represented by a context. In order to prove that security properties hold for all adversaries, quantification over all contexts is typically required, which can be difficult in practice. The labelled operational semantics we now present aims to eliminate universal quantification of the context.

The labelled semantics defines a ternary relation written $A \xrightarrow{\alpha} B$, where α is a *label* of the form $c(M)$, $\overline{c}\langle u\rangle$, or $\nu\, u.\overline{c}\langle u\rangle$ such that u is either a channel name or a variable of base type. The transition $A \xrightarrow{c(M)} B$ means that the process A performs an input of the term M from the environment on the channel c, and the resulting process is B. The situation for output is a bit more complicated, since there are several cases. If the value being output is a free variable x or a free channel name d, then the label $\overline{c}\langle x\rangle$, respectively $\overline{c}\langle d\rangle$, is used. If the value is a restricted channel name d, then the label $\nu\, d.\overline{c}\langle d\rangle$ is used. Finally, if the value being output is a term M, then the label $\nu\, x.\overline{c}\langle x\rangle$ is used, after replacing the occurrence of the term M by x and wrapping the process in $\nu x.(\{^M/_x\} \mid \_)$. The semantics are extended to include the rules that appear in Figure 2.5.

Figure 2.5 Semantics for labelled transitions

IN	$c(x).P \xrightarrow{c(M)} P\{^M/_x\}$
OUT-ATOM	$\overline{c}\langle u\rangle.P \xrightarrow{\overline{c}\langle u\rangle} P$
OPEN-ATOM	$\dfrac{A \xrightarrow{\overline{c}\langle u\rangle} A' \qquad u \neq c}{\nu\, u.A \xrightarrow{\nu\, u.\overline{c}\langle u\rangle} A'}$
SCOPE	$\dfrac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu\, u.A \xrightarrow{\alpha} \nu\, u.A'}$
PAR	$\dfrac{A \xrightarrow{\alpha} A' \qquad \mathrm{bv}(\alpha) \cap \mathrm{fv}(B) = \mathrm{bn}(\alpha) \cap \mathrm{fn}(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$
STRUCT	$\dfrac{A \equiv B \qquad B \xrightarrow{\alpha} B' \qquad B' \equiv A'}{A \xrightarrow{\alpha} A'}$

2.3 Observational equivalence

The notion of indistinguishability is a powerful concept which allows us to reason about complex properties that cannot be expressed as trace-based assertions. Intuitively, two processes are said to be equivalent if an observer has no way to tell them apart. The processes may be handling different data, and internally performing quite different computations, but they look the same to an external observer. This notion allows us to define strong notions of secrecy and privacy.

Roughly speaking, processes A and B are said to be observationally equivalent if: they can output on the same channel for all contexts they are placed inside of, and for every step made by A, there exists a step that B can make, such that the resulting pair of processes are observationally equivalent (and vice-versa). We write $A \Downarrow c$ when A can evolve to a process that can send a message on channel c; that is, when $A \to^* C[\overline{c}\langle M\rangle.P]$ for some term M, process P and evaluation context $C[\_]$ that does not bind c.

Definition 2.1 (Observational equivalence)**.** *Observational equivalence $\approx$ is the largest symmetric relation $\mathcal{R}$ between closed extended processes with the same domain such that $A\ \mathcal{R}\ B$ implies:*

1. *if $A \Downarrow c$, then $B \Downarrow c$;*

2. *if* $A \rightarrow^* A'$, *then* $B \rightarrow^* B'$ *and* $A' \mathcal{R} B'$ *for some* B';

3. $C[A] \mathcal{R} C[B]$ *for all closing evaluation contexts* $C[\_]$.

2.4 Assumptions and notation

In this thesis, all signatures are tacitly assumed to include the constant $\varnothing$, unary functions fst, snd, and the binary function pair. In addition, equational theories are assumed to include:

$$\mathsf{fst}(\mathsf{pair}(x, y)) = x \qquad \mathsf{snd}(\mathsf{pair}(x, y)) = y$$

For convenience, $\mathsf{pair}(M_1, \mathsf{pair}(\ldots, \mathsf{pair}(M_n, \varnothing)))$ is occasionally abbreviated as $(\!|M_1, \ldots, M_n|\!)$ and $\mathsf{fst}(\mathsf{snd}^{i-1}(M))$ is denoted $\pi_i(M)$. We also introduce some convenient notation for sequence of names or variables. Given an infinite sequence of names or variables $\mathsf{u} = u_1, u_2, u_3, u_4, \ldots$, the following self-explanatory abbreviations are used:

$$\begin{aligned} \mathsf{head}(\mathsf{u}) &= u_1 \\ \mathsf{tail}(\mathsf{u}) &= u_2, u_3, u_4, \ldots \\ \mathsf{odd}(\mathsf{u}) &= u_1, u_3, \ldots \\ \mathsf{even}(\mathsf{u}) &= u_2, u_4, \ldots \end{aligned}$$

Part II

Formalisation of security properties

The failure of cryptographic protocols has been directly attributed to the absence of rigorously defined security properties. This part introduces symbolic definitions for security in electronic voting and anonymous attestation.

3

Election verifiability in electronic voting[†]

Overview. *A definition of election verifiability is presented in terms of boolean tests which can be performed on the data produced by an election. The definition allows the evaluation of election verifiability in electronic voting protocols. It also allows the identification of hardware and software components that must be trusted for the purpose of verifiability, thereby facilitating the comparison of electronic voting protocols on the basis of trust assumptions. Our definition of election verifiability is compatible with a large class of electronic voting schemes – including those based upon blind signatures, homomorphic encryption and mixnets – as will be demonstrated by analysing the FOO, Helios 2.0 and JCJ-Civitas electronic voting protocols.*

Electronic voting systems lack the transparency provided by their paper counterparts. For example, paper-based elections often allow observation of the whole process (that is, from ballot casting to tallying) and rely upon robustness characteristics of the physical world (such as the impossibility of altering the markings on a paper ballot sealed inside a locked ballot box). By comparison, it is not possible to observe the electronic operations performed on bitstrings. As a consequence, computer systems may alter votes in a way that cannot be detected. Some electronic voting systems attempt to eliminate the necessity for transparency under the hypothesis that the hardware and software running the election can be trusted. Unsurprisingly, this level of trust is very difficult to achieve and thus

[†]This chapter is an extension of [KRS10, SRKK10].

deployed systems based upon such assumptions have failed [Bun09, Min08, Bow07, UK07].

The concept of *election verifiability* (also known as *end-to-end verifiability*) has emerged in the academic literature to address this problem (for example, [JCJ02, CRS05, Adi06, Dag07, Adi08]). The notion should allow voters and election observers to verify – independently of the hardware and software running the election – that votes have been recorded, tallied and declared correctly. Two aspects of verifiability are generally distinguished.

- *Individual verifiability:* a voter can check that her own ballot is published on the election's bulletin board.
- *Universal verifiability:* anyone can check that all the votes in the election outcome correspond to ballots published on the election's bulletin board.

In this thesis, another aspect is also identified.

- *Eligibility verifiability:* anyone can check that each ballot published on the bulletin board was cast by a registered voter and at most one ballot is tallied per voter.

Eligibility verifiability is explicitly distinguished as a distinct property (although it occasionally appears as part of universal verifiability in the literature).

Chapter contribution

A definition of election verifiability which captures individual, universal and eligibility verifiability is presented. Formally, the definition captures verifiability as a triple of boolean tests Φ^{IV}, Φ^{UV}, and Φ^{EV} which are required to satisfy several conditions when parametrised with the data produced by an election protocol, for all possible protocol executions. The test Φ^{IV} is intended to be checked by individual voters who instantiate the test with their private information (for example, their vote, and data derived during the execution of the protocol) and public information relating to the election (for example, the contents of the bulletin board). The tests Φ^{UV} and Φ^{EV} can be checked by any external observer and hence only rely on public information.

The consideration of eligibility verifiability is particularly interesting, as it provides an assurance that the election outcome corresponds to votes legitimately cast. This property has been largely neglected by existing electronic voting protocols, despite its suitability as a mechanism to detect ballot stuffing.

Our definition of election verifiability dictates that only those parts of the voting system that need to be trusted to achieve verifiability should be modelled; all the remaining parts of the system are controlled by the adversarial environment. This is a further interesting aspect because it allows the clear identification of trust assumptions needed for verifiability, and, therefore, permits comparison of electronic voting protocols. In complementary work, Pieters [Pie10] compares voting systems on the basis of the proof techniques used to achieve verifiability.

Tests Φ^{IV}, Φ^{UV}, and Φ^{EV} are assumed to be verified in a trusted environment (if a test is checked by malicious software that always evaluates the test to hold, it is useless). However, the verification of these tests can be repeated on different machines, using software provided by various stakeholders, thereby increasing confidence. Alternatively, this assumption can be eliminated by adopting human-verifiable tests as described by Adida [Adi06, Chapter 5].

The application of our election verifiability definition is demonstrated by analysing verifiability in three case studies: the protocol by Fujioka, Okamoto & Ohta [FOO92] (commonly referred to as the FOO protocol); the Helios 2.0 protocol [AMPQ09] which was effectively used in the election of board memebers for the International Association of Cryptologic Research, the presidential election at the Catholic University of Louvain, and the student vice president at Princeton University; and the protocol by Juels, Catalano & Jakobsson [JCJ02, JCJ05, JCJ10] which has been implemented by Clarkson, Chong & Myers [CCM08, CCM07] as Civitas (we occassionally refer to this protocol as JCJ-Civitas). This demonstrates the suitability of the definition for a large class of protocols, including schemes based upon mixnets, homomorphic encryption and blind signatures. The protocols by Fujioka, Okamoto & Ohta and Helios 2.0 will be shown not to satisfy

eligibility verifiability and are vulnerable to ballot stuffing by dishonest administrators. As these protocols do not proclaim to satisfy eligibility verifiability, this is not claimed to be an attack, but simply a clarification of precisely which aspects of verifiability are satisfied.

Structure of this chapter. Section 3.1 formalises electronic voting protocols in the applied pi calculus. Section 3.2 specifies some notational conventions and defines the individual and universal verifiability aspects of our definition (Section 3.2.1). The analysis of FOO is presented in Section 3.2.2, and Helios is studied in Section 3.2.3. Our definition is extended to eligibility in Section 3.3, and JCJ-Civitas is analysed in Section 3.4. Finally, a summary is presented in Section 3.5.

3.1 Formalising electronic voting protocols

The framework should permit explicit specification of the trusted parts of an election protocol. Ideally, only the communication channel between voters and voting terminals should be trusted. In particular, the voter should not need to trust the election hardware or software. However, achieving absolute verifiability in this context is difficult and the trustworthiness of some parts of the protocol are often assumed. Such trust assumptions are motivated by the fact that certain components of a protocol can be audited, or can be executed in a distributed manner amongst several different election officials. For instance, in Helios 2.0 [AMPQ09] the ballot construction can be audited using a cast-or-audit mechanism (see Benaloh [Ben06, Ben07] for further details on ballot auditing).

Formally, the trusted parts of the voting protocol can be captured using a voting process specification (Definition 3.1). This specification makes trust assumptions explicit; however, whether such assumptions are reasonable depends on the context of an election.

Definition 3.1 (Voting process specification)**.** *A* voting process specification *is a tuple* $\langle V, A\rangle$*, where* V *is a plain process without replication and* A *is a closed evaluation context*

such that $\mathrm{fv}(V) = \{v\}$ *and* $\mathrm{rv}(V) = \emptyset$.

Given a voting process specification $\langle V, A\rangle$, integer $n \in \mathbb{N}$, and names $s_1, \ldots, s_n$, we can build the voting process

$$\mathsf{VP}_n(s_1, \ldots, s_n) = A[V_1 \mid \cdots \mid V_n]$$

where $V_i = V\{^{s_i}/_{v}\}$. Intuitively, $\mathsf{VP}_n(s_1, \ldots, s_n)$ models the protocol with n voters casting votes for candidates $s_1, \ldots, s_n$. Note that the votes $s_1, \ldots, s_n$ are not required to be distinct (several voters may cast votes for the same candidate).

Example 3.1 (Raising hands protocol)**.** *Consider the* raising hands *protocol in which every voter outputs her signed vote. Cryptographic primitives are captured by the signature* $\Sigma = \{\mathsf{true}, \mathsf{getmsg}, \mathsf{pk}, \mathsf{checksign}, \mathsf{sign}\}$*, where* true *is a constant,* getmsg *and* pk *are unary functions, and functions* $\mathsf{checksign}$ *and* sign *are binary. The signature is associated with the equations:*

$$\begin{aligned} \mathsf{checksign}(\mathsf{pk}(x_{\mathsf{sk}}), \mathsf{sign}(x_{\mathsf{sk}}, x_{\mathsf{msg}})) &= \mathsf{true} \\ \mathsf{getmsg}(\mathsf{sign}(x_{\mathsf{sk}}, x_{\mathsf{msg}})) &= x_{\mathsf{msg}} \end{aligned}$$

A trusted administrator, modelled by the context A_{ex}*, is assumed to distribute keying material to voters and publish each voter's public key.*

$$A_{\mathsf{ex}}[\_] \mathrel{\widehat{=}} \nu d.\nu sk_A.(!\nu sk_V.\overline{d}\langle sk_V\rangle.\overline{c}\langle\mathsf{sign}(sk_A, \mathsf{pk}(sk_V))\rangle \mid \{^{\mathsf{pk}(sk_A)}/_{x_{pk}}\} \mid \_)$$

The channel d *is under name restriction to ensure voters' keys are distributed privately, and the active substitution* $\{^{\mathsf{pk}(sk_A)}/_{x_{pk}}\}$ *models the fact that the administrator's public key is known (for example, published on the election bulletin board). The voter, who receives her private key and then outputs her signed vote paired with her public key, is modelled by the process:*

$$V_{\mathsf{ex}} \mathrel{\widehat{=}} d(x_{sk_V}).\overline{c}\langle(\!|\mathsf{pk}(x_{sk_V}), \mathsf{sign}(x_{sk_V}, v)|\!)\rangle$$

This protocol satisfies election verifiability, as will be shown later in this chapter.

For the purposes of individual verifiability, the voter may rely upon some data derived during the protocol execution. To take such data into consideration, we distinguish a subset of variables which we call *record variables* r and extend plain processes with the *record message* $\mathsf{rec}(r, M).P$ construct. The record message construct permits a voter to privately record some information which she may later use to verify the election. This behaviour is captured by extending internal reduction to include $\mathsf{rec}(r, M).P \rightarrow P \mid \{^{M}/_{r}\}$. Record variables are assumed to be unique in each process; that is, a record variable may appear at most once. The type system ensures that the term M and record variable r have the same type in $\mathsf{rec}(r, M)$; it also enforces that record variables may only be used in the first argument of the rec construct or in the domain of an extended process. Definition 3.2 allows processes to record restricted names and message inputs.

Definition 3.2. *Let* rv *be an infinite sequence of distinct record variables. Given a process* P *which does not contain replication, we define* $\mathsf{R}(P) = \mathsf{R}'(\mathsf{rv}, P)$*, where:*

$$
\begin{array}{lcl}
\mathsf{R}'(\mathsf{r},\, 0) & \widehat{=} & 0 \\
\mathsf{R}'(\mathsf{r},\, P \mid Q) & \widehat{=} & \mathsf{R}'(\mathit{odd}(\mathsf{r}), P) \mid \mathsf{R}'(\mathit{even}(\mathsf{r}), Q) \\
\mathsf{R}'(\mathsf{r},\, \nu\, n.P) & \widehat{=} & \nu\, n.\mathsf{rec}(\mathit{head}(\mathsf{r}), n).\mathsf{R}'(\mathit{tail}(\mathsf{r}), P) \\
\mathsf{R}'(\mathsf{r},\, u(x).P) & \widehat{=} & u(x).\mathsf{rec}(\mathit{head}(\mathsf{r}), x).\mathsf{R}'(\mathit{tail}(\mathsf{r}), P) \\
\mathsf{R}'(\mathsf{r},\, \overline{u}\langle M\rangle.P) & \widehat{=} & \overline{u}\langle M\rangle.\mathsf{R}'(\mathsf{r}, P) \\
\mathsf{R}'(\mathsf{r},\, \text{if } M = N \text{ then } P \text{ else } Q) & \widehat{=} & \text{if } M = N \text{ then } \mathsf{R}'(\mathit{odd}(\mathsf{r}), P) \text{ else } \mathsf{R}'(\mathit{even}(\mathsf{r}), Q)
\end{array}
$$

Given a tuple of record variables $\tilde{r}$, the tuple of record variables obtained by indexing each record variable in $\tilde{r}$ with i is denoted $\tilde{r}_i$. The set of record variables in a process and term are denoted $\mathrm{rv}(A)$ and $\mathrm{rv}(M)$. A voting process can now be constructed such that the voter V records the values constructed and input during execution.

Definition 3.3 (Augmented voting process)**.** *Given a voting process specification* $\langle V, A\rangle$*, integer* $n \in \mathbb{N}$ *and names* $s_1, \ldots, s_n$*, the* augmented voting process

$$\mathsf{VP}_n^+(s_1, \ldots, s_n) = A[V_1^+ \mid \cdots \mid V_n^+]$$

where $V_i^+ = \mathsf{R}(V)\{^{s_i}/_v\}\{^{r_i}/_r \mid r \in \mathrm{rv}(\mathsf{R}(V))\}$.

The augmented voting process $\mathsf{VP}_n^+(s_1, \ldots, s_n)$ models the voting protocol for n voters casting votes $s_1, \ldots, s_n$, who privately record the data that may be needed for verification using record variables $\tilde{r}_i$.

Example 3.2. *Given integer* $n \in \mathbb{N}$ *and names* $s_1, \ldots, s_n$*, the augmented voting process associated with our raising hands process specification* $\langle V_{\mathsf{ex}}, A_{\mathsf{ex}}\rangle$ *is defined as follows*

$$A_{\mathsf{ex}}[d(x_{sk_V}).\mathsf{rec}(r_1, x_{sk_V}).\overline{c}\langle(\!|\mathsf{pk}(x_{sk_V}), \mathsf{sign}(x_{sk_V}, s_1)|\!)\rangle \mid \ldots \mid d(x_{sk_V}).\mathsf{rec}(r_n, x_{sk_V}).\overline{c}\langle(\!|\mathsf{pk}(x_{sk_V}), \mathsf{sign}(x_{sk_V}, s_n)|\!)\rangle]$$

3.2 Election verifiability

Election verifiability is captured using three tests Φ^{IV}, Φ^{UV}, and Φ^{EV}. Formally, a test is built from conjunctions and disjunctions of *atomic tests* of the form $(M = N)$, where M, N are terms. Tests may contain variables, and will need to hold on frames arising from arbitrary protocol executions. We now recall the purpose of each test and assume some naming conventions about variables.

Individual verifiability: The test Φ^{IV} allows a voter to identify her ballot on the bulletin board. The test has:

- a variable v referring to a voter's vote.
- a variable w referring to a voter's public credential.

- some variables $x, x', \bar{x}, \ldots$ expected to refer to global public values pertaining to the election (for example, public keys belonging to election administrators).
- a variable y expected to refer to the voter's ballot on the bulletin board.
- some record variables $r_1, \ldots, r_k$ referring to the voter's private data.

Universal verifiability: The test Φ^{UV} allows an observer to check that votes in the election outcome correspond to ballots on the bulletin board.

- a variable $\hat{v}$ referring to a tuple representing the election outcome.
- some variables $x, x', \bar{x}, \ldots$ as above.
- a variable $\hat{y}$ expected to refer to a tuple containing all of the voters' ballots on the bulletin board.
- some variables $z, z', \bar{z}, \ldots$ expected to refer to outputs generated by the protocol for the purposes of universal and eligibility verification.

Eligibility verifiability: The test Φ^{EV} allows an observer to check that each ballot on the bulletin board was cast by a registered voter and at most one ballot is tallied per voter. The test has:

- a variable $\hat{w}$ referring to a tuple containing public credentials of eligible votes.
- a variable $\hat{y}$, variables $x, x', \bar{x}, \ldots$ and variables $z, z', \bar{z}, \ldots$ as above.

The remainder of this section will focus on the individual and universal aspects of our definition; eligibility verifiability will be discussed in Section 3.3.

3.2.1 Security definition: Individual and universal verifiability

The test Φ^{IV} is parametrised with a single bulletin board entry (which an individual voter needs to identify in some way, possibly by testing all of them); by comparison, the tests

Φ^{UV} and Φ^{EV} can be applied to several bulletin board entries. The tests Φ^{UV} and Φ^{EV} are therefore parametrised with an integer $m \in \mathbb{N}$ denoting the m bulletin board entries considered; accordingly, we write Φ^{UV}_m and Φ^{EV}_m to denote such tests.

The tests suitable for the purposes of election verifiability have to satisfy certain conditions: if the tests succeed, then the data output by the election is indeed valid (*soundness*), and there is a behaviour of the election administrators which produces election data satisfying the tests (*effectiveness*). Formally, these requirements are captured by Definition 3.4. We write $\hat{T} \simeq \hat{T}'$ to denote that the terms $\hat{T}$ and $\hat{T}'$ are a permutation of each other modulo the equational theory; that is, $\hat{T} = (\!|T_1, \ldots T_n|\!)$, $\hat{T}' = (\!|T'_1, \ldots T'_n|\!)$ such that there exists a permutation χ on $\{1, \ldots, n\}$ and for all $1 \leq i \leq n$ we have $\pi_i(T) =_E \pi_{\chi(i)}(T')$ for some terms $T_1, T'_1, \ldots, T_n, T'_n$ and integer $n \in \mathbb{N}$.

Definition 3.4 (Individual and universal verifiability)**.** *A voting specification* $\langle V, A\rangle$ satisfies individual and universal verifiability *if there exists a test* Φ^{IV}*, where for all* $m \in \mathbb{N}$ *there exists a test* Φ^{UV}_m*, such that* $\mathrm{fn}(\Phi^{IV}) = \mathrm{fn}(\Phi^{UV}_m) = \mathrm{rv}(\Phi^{UV}_m) = \emptyset$ *and* $\mathrm{rv}(\Phi^{IV}) \subseteq \mathrm{rv}(\mathsf{R}(V))$*, and for all names* $s_1, \ldots, s_n$ *the conditions below hold. Let* $\tilde{r} = \mathrm{rv}(\Phi^{IV})$ *and* $\Phi^{IV}_i = \Phi^{IV}\{^{s_i}/_{v}, ^{\tilde{r}_i}/_{\tilde{r}}\}$.

Soundness. *For all contexts* C*, such that* $C[\mathsf{VP}^+_n(s_1, \ldots, s_n)]\ (\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*\ B$ *and* $\varphi(B) \equiv \nu\tilde{n}.\sigma$ *for some process* B*, substitution* σ *and names* $\tilde{n}$*, we have:*

$$\forall i, j.\quad \Phi^{IV}_i\sigma \wedge \Phi^{IV}_j\sigma \Rightarrow i = j \tag{3.1}$$

$$\Phi^{UV}_m\sigma \wedge \Phi^{UV}_m\{^{\hat{v}'}/_{\hat{v}}\}\sigma \Rightarrow \hat{v}\sigma \simeq \hat{v}'\sigma \tag{3.2}$$

$$\bigwedge_{1\leq i\leq n} \Phi^{IV}_i\{^{\pi_i(\hat{y})}/_{y}\}\sigma \wedge \Phi^{UV}_m\sigma \wedge n = m \Rightarrow (\!|s_1, \ldots, s_n|\!) \simeq \hat{v}\sigma \tag{3.3}$$

Effectiveness. *There exists a context C, process B, substitution σ and names $\tilde{n}$, such that* $C[\mathsf{VP}_n^+(s_1,\ldots,s_n)]\ (\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*\ B$, $\varphi(B) \equiv \nu\tilde{n}.\sigma$ *and*

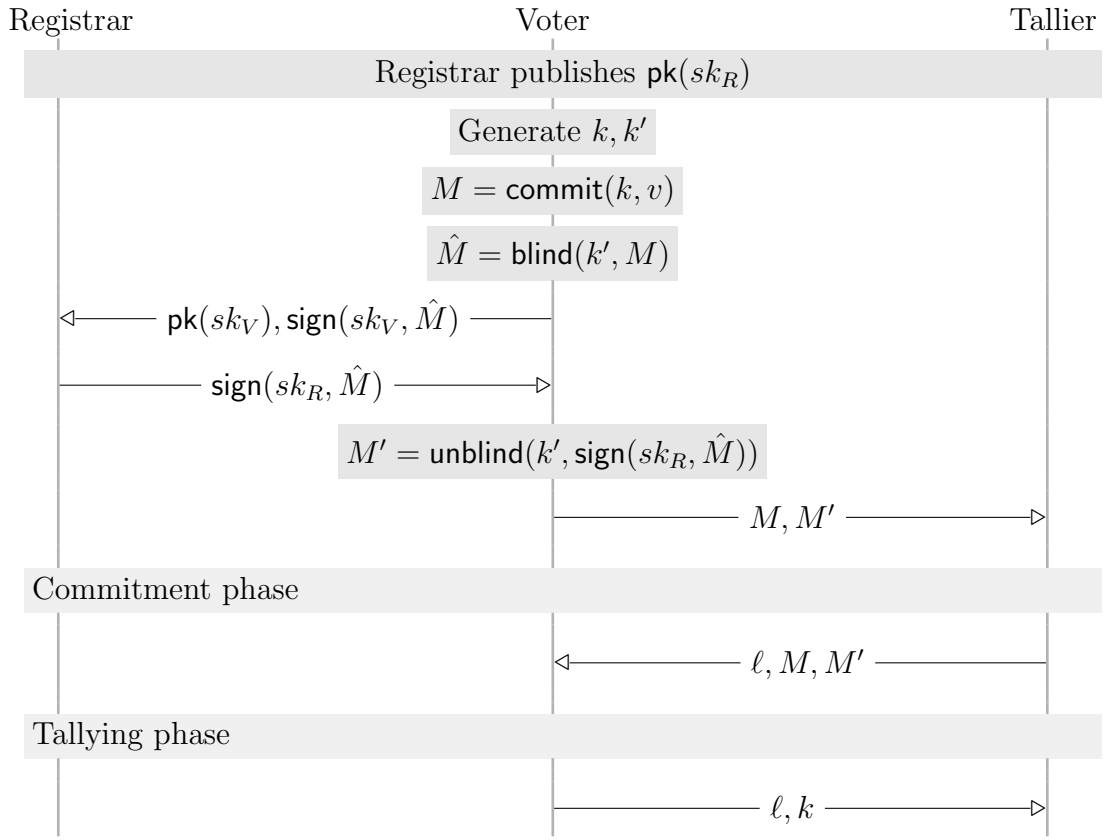
$$\bigwedge_{1\leq i\leq n} \Phi_i^{IV}\{\pi_i(\hat{y})/_y\}\sigma \wedge \Phi_n^{UV}\sigma \tag{3.4}$$

An individual voter should verify that the test Φ^{IV} holds when instantiated with her vote s_i, the information $\tilde{r}_i\sigma$ recorded during the execution of the protocol and some bulletin board entry $y\sigma$. Condition (3.1) ensures that the test Φ^{IV} will hold for at most one bulletin board entry. (Note that Φ_i^{IV} and Φ_j^{IV} are evaluated with the same ballot $y\sigma$ provided by $C[\_]$.) The fact that her ballot is counted will be ensured by Φ_m^{UV}, which should also be tested by the voter. The voter or an observer will instantiate the test Φ_m^{UV} with the bulletin board entries $\hat{y}\sigma$ and the election outcome $\hat{v}\sigma$. Condition (3.2) ensures that Φ_m^{UV} holds only for a single outcome. Condition (3.3) ensures that if the bulletin board contains the ballots of voters who voted $s_1,\ldots,s_n$, then Φ_m^{UV} holds only if the declared outcome is a permutation of these votes, where Φ_m^{UV} is instantiated with precisely the n ballots cast by those voters (that is, there are no additional ballots in $\hat{y}\sigma$). The necessity for the precondition $n = m$ should make it clear that individual and universal verifiability cannot detect ballot stuffing. Finally, Condition (3.4) ensures that there exists an execution where the tests hold. In particular, this allows us to verify whether the protocol can satisfy the tests when executed as expected. This also forbids tests which always evaluate to false and would make Conditions (3.1)-(3.3) vacuously hold.

3.2.2 Case study: FOO

The FOO protocol, by Fujioka, Okamoto & Ohta [FOO92], is a seminal work based upon blind signatures.

Figure 3.1 FOO electronic voting protocol



Protocol description

An election is created by naming a registrar and a tallier. The protocol is divided into four phases: *setup*, *preparation*, *commitment* and *tallying*. The steps of the protocol (Figure 3.1) will now be detailed, starting with the setup phase.

1. The registrar creates a signing key pair $sk_R, \mathsf{pk}(sk_R)$ and publishes the public part $\mathsf{pk}(sk_R)$. Similarly, each voter is assumed to have a key pair $sk_V, \mathsf{pk}(sk_V)$.

The preparation phase then proceeds as follows.

2. The voter computes the commitment to her vote $M = \mathsf{commit}(k, v)$ and sends the signed blind commitment $\mathsf{sign}(sk_V, \mathsf{blind}(k', M))$, paired with her public key, to the registrar.

3. The registrar checks that the signature belongs to an eligible voter and returns the blind commitment signed by the registrar $\mathsf{sign}(sk_R, \mathsf{blind}(k', M))$.

4. The voter verifies the registrar's signature and unblinds the message to recover $M' = \mathsf{sign}(sk_R, M)$, that is, her commitment signed by the registrar.

After some fixed deadline, the protocol enters the commitment phase.

5. The voter posts her ballot M, M' to the bulletin board.

Similarly, the last phase begins after some fixed deadline.

6. The tallier verifies all the bulletin board entries and appends an identifier ℓ to each valid entry.

7. The voter checks the bulletin board for her entry, the triple ℓ, M, M', and appends the commitment factor k.

8. Finally, using k, the tallier opens all of the ballots and announces the election outcome.

The distinction between phases is essential to uphold the protocol's security properties. In particular, voters must synchronise before the commitment phase to ensure privacy (observe that without synchronisation, traffic analysis may allow the voter's signature to be linked with the commitment to her vote, which can then be linked to her vote) and before the tallying phase to avoid publishing partial results, that is, to ensure the *fairness* property of electronic voting.

Equational theory

Based upon [KR05, DKR06, DKR09, DKR10b], blind signatures and bit commitment are modelled by the following equations

$$\begin{aligned}
\mathsf{unblind}(x_{\mathsf{rand}}, \mathsf{sign}(x_{\mathsf{sk}}, \mathsf{blind}(x_{\mathsf{rand}}, x_{\mathsf{msg}}))) &= \mathsf{sign}(x_{\mathsf{sk}}, x_{\mathsf{msg}}) \\
\mathsf{unblind}(x_{\mathsf{rand}}, \mathsf{blind}(x_{\mathsf{rand}}, x_{\mathsf{plain}})) &= x_{\mathsf{plain}} \\
\mathsf{open}(x_{\mathsf{rand}}, \mathsf{commit}(x_{\mathsf{rand}}, x_{\mathsf{plain}})) &= x_{\mathsf{plain}} \\
\mathsf{checksign}(\mathsf{pk}(x_{\mathsf{sk}}), \mathsf{sign}(x_{\mathsf{sk}}, x_{\mathsf{msg}})) &= \mathsf{true} \\
\mathsf{getmsg}(\mathsf{sign}(x_{\mathsf{sk}}, x_{\mathsf{msg}})) &= x_{\mathsf{msg}}
\end{aligned}$$

where true is a constant.

Model in applied pi

For verifiability, the voter must be able to generate a fresh nonce k (which should subsequently be used by the untrusted voting terminal to compute the commitment to her vote); all other operations – such as computing the commitment, blinding and signing – can be performed by an untrusted voting terminal. Accordingly, the trusted components are modelled as follows.

Definition 3.5. *The voting process specification* $\langle V_{\mathsf{foo}}, A_{\mathsf{foo}} \rangle$ *is defined as* $V_{\mathsf{foo}} \mathrel{\hat{=}} \nu k.\overline{c}\langle v \rangle.\overline{c}\langle k \rangle$ *and* $A_{\mathsf{foo}}[\_] \mathrel{\hat{=}} \_$.

We do not assert that the voting terminal uses blind signatures (although this is crucial for privacy properties, it does not contribute to verifiability); accordingly, blinding operations do not appear in our specification and, moreover, no trust assumptions are made about whether blinding is used. Similarly, the voter's signature on the blinded committed vote and the confidentiality of signing keys are not required for individual and universal verifiability; they are, however, essential for eligibility.

Analysis: Individual and universal verifiability

Given integer $m \in \mathbb{N}$ let tests Φ^{IV} and Φ^{UV}_m be defined as follows:

$$\begin{aligned}\Phi^{IV} &\hat{=} y =_E (\!|r, \mathsf{commit}(r, v)|\!)\\ \Phi^{UV}_m &\hat{=} \hat{v} =_E (\!|\mathsf{open}(\pi_1(\pi_1(\hat{y})), \pi_2(\pi_1(\hat{y}))), \ldots, \mathsf{open}(\pi_1(\pi_m(\hat{y})), \pi_2(\pi_m(\hat{y})))|\!)\end{aligned}$$

Intuitively, a bulletin board entry should be a pair formed from the voter's nonce and the associated commitment to her vote. The identifier ℓ and the registrar's signature are not required for the purpose of verifiability, so these details are omitted.

Theorem 3.1. $\langle V_{\mathsf{foo}}, A_{\mathsf{foo}}\rangle$ *satisfies individual and universal verifiability.*

Proof. Suppose $m \in \mathbb{N}$ and tests Φ^{IV}, Φ^{UV}_m are given above. We will now show that the conditions of Definition 3.4 are satisfied.

(3.1) For all names $s_1, \ldots, s_n$ suppose C is a context, B is a process, σ is a substitution, $\tilde{n}$ is a tuple of names and i, j are integers such that $C[\mathsf{VP}^+_{\mathsf{n}}(s_1, \ldots, s_n)]\ (\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*\ B$, $\varphi(B) \equiv \nu\tilde{n}.\sigma$, $\Phi^{IV}\{{}^{s_i}/_{v}, {}^{r_i}/_{r}\}\sigma$ and $\Phi^{IV}\{{}^{s_j}/_{v}, {}^{r_j}/_{r}\}\sigma$, where $r = \mathrm{rv}(\Phi^{IV})$. It follows that $\pi_1(y)\sigma =_E r_i\sigma =_E r_j\sigma$. Since the record variables r_i and r_j are handles for fresh nonces generated by name restriction in processes $V_{\mathsf{foo},i}$ and $V_{\mathsf{foo},j}$, it follows that $i = j$.

(3.2) We prove a stronger result: namely, the condition holds for all substitutions σ. Suppose σ is an arbitrary substitution such that $\Phi^{UV}_m\sigma$ and $\Phi^{UV}_m\{{}^{\hat{v}'}/_{\hat{v}}\}\sigma$ hold. We have

$$\hat{v}\sigma =_E (\!|\mathsf{open}(\pi_1(\pi_1(\hat{y})), \pi_2(\pi_1(\hat{y}))), \ldots, \mathsf{open}(\pi_1(\pi_m(\hat{y})), \pi_2(\pi_m(\hat{y})))|\!)\sigma =_E \hat{v}'\sigma$$

and immediately derive $\hat{v}\sigma \simeq \hat{v}'\sigma$.

(3.3) Again, we will show that the condition holds for all substitutions σ. Suppose σ is an arbitrary substitution such that $\bigwedge_{1\le i\le n} \Phi^{IV}_i\{{}^{\pi_i(\hat{y})}/_{y}\}\sigma$ and $\Phi^{UV}_m\sigma$ hold, where $n = m$. By $\bigwedge_{1\le i\le n} \Phi^{IV}_i\{{}^{\pi_i(\hat{y})}/_{y}\}\sigma$ it must be the case for all $1 \le i \le n$ that

$$\pi_i(\hat{y})\sigma =_E (\!|r_i, \mathsf{commit}(r_i, s_i)|\!)\sigma$$

and hence $\mathsf{open}(\pi_1(\pi_i(\hat{y})), \pi_2(\pi_i(\hat{y})))\sigma =_E s_i$. Moreover, since $n = m$ and $\Phi_m^{UV}\sigma$ we have

$$\begin{aligned}\hat{v}\sigma &=_E (\!|\mathsf{open}(\pi_1(\pi_1(\hat{y})), \pi_2(\pi_1(\hat{y}))), \ldots, \mathsf{open}(\pi_1(\pi_m(\hat{y})), \pi_2(\pi_m(\hat{y})))|\!)\sigma \\ &=_E (\!|s_1, \ldots, s_n|\!)\end{aligned}$$

The result $(\!|s_1, \ldots, s_n|\!) \simeq \hat{v}\sigma$ follows.

(3.4) Suppose $s_1, \ldots, s_n$ are names and consider the following context:

$$\begin{aligned}C_{\mathsf{foo}} = \; & c(x_{\mathsf{vote},1}).c(x_{\mathsf{rand},1}).\text{let } y_1 = (\!|x_{\mathsf{rand},1}, \mathsf{commit}(x_{\mathsf{rand},1}, x_{\mathsf{vote},1})|\!) \text{ in} \\ & \vdots \\ & c(x_{\mathsf{vote},n}).c(x_{\mathsf{rand},n}).\text{let } y_n = (\!|x_{\mathsf{rand},n}, \mathsf{commit}(x_{\mathsf{rand},n}, x_{\mathsf{vote},n})|\!) \text{ in} \\ & \overline{c}\langle(\!|y_1, \ldots, y_n|\!)\rangle. \\ & \overline{c}\langle(\!|\mathsf{open}(\pi_1(y_1), \pi_2(y_1)), \ldots, \mathsf{open}(\pi_1(y_n), \pi_2(y_n))|\!)\rangle \mid \_\end{aligned}$$

We have $C[\mathsf{VP}_n^+(s_1, \ldots, s_n)]\ (\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*\ B$ such that

$$\begin{aligned}B = \nu\, k_1, \ldots, k_n.(\overline{c}\langle(\!|(\!|k_1, \mathsf{commit}(k_1, s_1)|\!), \ldots, (\!|k_n, \mathsf{commit}(k_n, s_n)|\!)|\!)\rangle. \\ \overline{c}\langle(\!|s_1, \ldots, s_n|\!)\rangle \mid \{k_1/r_1, \ldots, k_n/r_n\})\end{aligned}$$

Moreover, $B \xrightarrow{\nu\, \hat{y}.\overline{c}\langle\hat{y}\rangle} \xrightarrow{\nu\, \hat{v}.\overline{c}\langle\hat{v}\rangle} \nu\, k_1, \ldots, k_n.\sigma$ such that

$$\sigma = \{(\!|(\!|k_1, \mathsf{commit}(k_1, s_1)|\!), \ldots, (\!|k_n, \mathsf{commit}(k_n, s_n)|\!)|\!)/\hat{y}, (\!|s_1, \ldots, s_n|\!)/\hat{v}, k_1/r_1, \ldots, k_n/r_n\}$$

It can trivially be seen that $\bigwedge_{1 \leq i \leq n} \Phi_i^{IV}\{\pi_i(\hat{y})/y\}\sigma \wedge \Phi_n^{UV}\sigma$. □

3.2.3 Case study: Helios 2.0

Helios 2.0 [AMPQ09] is an open-source web-based election system, based upon homomorphic tallying of encrypted votes [ElG85, CP93, CDS94, CGS97, Sch09]. It allows the secret election key to be distributed amongst several trustees, and supports distributed

decryption of the election result. It also allows independent verification by voters and observers. Helios 2.0 has been successfully used by the International Association of Cryptologic Research to elect its board members [BVQ10], following a successful trial in a non-binding poll [HBH10]. Helios has also been used by the Catholic University of Louvain, in an election that had 25,000 eligible voters, to elect the University president, and by Princeton University to elect the student vice president [Pri10].

Protocol description

An election is created by naming an election officer, selecting a set of trustees and running a protocol that provides each trustee with a share of the secret part of a public key pair. The election officer publishes the public part of the trustees' key and the candidate list $\tilde{t} = (t_1, \ldots, t_l)$ on the bulletin board. The steps that participants take during a run of Helios are as follows. (See [CS10b, CS11] for a formal cryptographic description.)

1. To cast a vote, the voter runs a browser script that inputs her vote $s \in \tilde{t}$ and creates a ballot consisting of her vote encrypted by the trustees' public key and a proof that the ballot represents an allowed vote (this is needed because the ballots are never decrypted individually; in particular, it prevents multiple votes being encoded as a single ballot).

2. The voter can audit the ballot to check if it really represents a vote for her chosen candidate; if she decides to do this, then the script provides her with the random data used in the ballot creation. She can then independently reconstruct her ballot and verify that it was indeed well-formed, but the ballot is now invalid. (Invalidating audited ballots provides some protection against vote selling.)

3. When the voter has decided to cast her ballot, she submits it to the election officer. The election officer authenticates the voter and checks that she is eligible to vote. The election officer also verifies the proof, and publishes the ballot on the bulletin board.

4. Individual voters can check that their ballots appear on the bulletin board and, by verifying the proof, observers are assured that ballots represent permitted votes.

5. After some predefined deadline, the election officer homomorphically combines the ballots and publishes the encrypted tally on the bulletin board. Anyone can check that tallying is performed correctly.

6. Each of the trustees publishes a partial decryption of the encrypted tally, together with a proof of correct construction. Anyone can verify these proofs.

7. The election officer decrypts the tally and publishes the result. Anyone can check this decryption.

Equational theory

We use a signature in which $\mathsf{penc}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, x_{\mathsf{plain}})$ denotes the encryption of plaintext x_{plain} using random x_{rand} and key x_{pk}, and $x_{\mathsf{ciph}} * y_{\mathsf{ciph}}$ denotes the homomorphic combination of ciphertexts x_{ciph} and y_{ciph} (the corresponding operation on plaintexts is written $+$ and on randoms $\circ$). The term $\mathsf{ballotPf}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, s, x_{\mathsf{ballot}})$ represents a proof that the ballot x_{ballot} is a ciphertext on some plaintext name s, random x_{rand} and key x_{pk}; $\mathsf{partial}(x_{\mathsf{sk}}, x_{\mathsf{ciph}})$ is a partial decryption of x_{ciph} when x_{ciph} is a ciphertext encrypted using the public key $\mathsf{pk}(x_{\mathsf{sk}})$; and $\mathsf{partialPf}(x_{\mathsf{sk}}, x_{\mathsf{ciph}}, x_{\mathsf{partial}})$ is a proof that x_{partial} is a partial decryption of x_{ciph} when x_{ciph} is a ciphertext encrypted using public key $\mathsf{pk}(x_{\mathsf{sk}})$. We use the equational theory that asserts that $+, *, \circ$ are commutative and associative, and includes the equations:

$$\mathsf{dec}(x_{\mathsf{sk}}, \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{plain}})) = x_{\mathsf{plain}}$$

$$\mathsf{dec}(\mathsf{partial}(x_{\mathsf{sk}}, ciph), ciph) = x_{\mathsf{plain}}$$
$$\text{where } ciph = \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{plain}})$$

$$\mathsf{penc}(x_{\mathsf{pk}}, y_{\mathsf{rand}}, y_{\mathsf{plain}}) * \mathsf{penc}(x_{\mathsf{pk}}, z_{\mathsf{rand}}, z_{\mathsf{plain}}) = \mathsf{penc}(x_{\mathsf{pk}}, y_{\mathsf{rand}} \circ z_{\mathsf{rand}}, y_{\mathsf{plain}} + z_{\mathsf{plain}})$$

$$\mathsf{checkBallotPf}(x_{\mathsf{pk}}, ballot, \mathsf{ballotPf}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, s, ballot)) = \mathsf{true}$$

where $ballot = \mathsf{penc}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, s)$

$\mathsf{checkPartialPf}(\mathsf{pk}(x_{\mathsf{sk}}), ciph, partial, \mathsf{partialPf}(x_{\mathsf{sk}}, ciph, partial)) = \mathsf{true}$

where $ciph = \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{plain}})$ and $partial = \mathsf{partial}(x_{\mathsf{sk}}, ciph)$

Note that in the equation for $\mathsf{checkBallotPf}$, s is a name and not a variable. As the equational theory is closed under bijective renaming of names, this equation holds for any name, but fails if one replaces the name by a term, for example, $s + s$. We assume that all names are possible votes but give the possibility to check that a voter does not include a term $s + s$ which would add a vote to the outcome.

Model in applied pi

The browser script is not verifiable; accordingly, the voter most trust:

- Ballot construction; that is, the script generates a pair consisting of the voter's encrypted vote and a proof that the ballot represents an allowed vote.

Although the voter cannot be assured that the script behaves correctly, trust is motivated because the voter can audit ballot construction. Accordingly, the browser script is modelled as part of the trusted context A_{helios} in the voting process specification $\langle V_{\mathsf{helios}}, A_{\mathsf{helios}} \rangle$. The voter V_{helios} receives a channel name y on a private channel; this is a technical aspect of our formalisation which allows the voter to privately communicate with her browser script. She sends her vote on this channel to A_{helios}, which creates the ballot for her. The voter is sent the constructed ballot x_{ballot} and forwards it to the bulletin board. We assume that the voter's inputs y and x_{ballot} are stored in record variables r_y and r_{ballot}. A_{helios} represents the parts of the system that are required to be trusted; it publishes the election key, and includes the ballot creation script B which receives a voter's vote, generates a random m and returns the ballot (that is, the encrypted vote and a proof) to the voter.

Definition 3.6. *The voting process specification* $\langle V_{\mathsf{helios}}, A_{\mathsf{helios}}\rangle$ *is defined where*

$$
\begin{aligned}
V_{\mathsf{helios}} &\hat{=} d(y).\overline{y}\langle v\rangle.y(x_{\mathsf{ballot}}).\overline{c}\langle x_{\mathsf{ballot}}\rangle\\
A_{\mathsf{helios}}[\_] &\hat{=} \nu d.\big(\,(!\nu d'.\overline{d}\langle d'\rangle.B) \mid \{\mathsf{pk}(sk)/_{x_{\mathsf{pk}}}\} \mid \_\,\big)\\
B &\hat{=} d'(x_{\mathsf{vote}}).\nu m.\overline{d'}\langle(\!|\mathsf{penc}(x_{\mathsf{pk}}, m, x_{\mathsf{vote}}), \mathsf{ballotPf}(x_{\mathsf{pk}}, m, x_{\mathsf{vote}}, \mathsf{penc}(x_{\mathsf{pk}}, m, x_{\mathsf{vote}}))|\!)\rangle
\end{aligned}
$$

At the end of the election the bulletin board is represented by a frame. The frame is expected to define the trustees' public key as x_{pk} and the ballots as $\hat{y}$. It also contains the homomorphic tally z_{tally} of the encrypted ballots, and the partial decryption z_{partial}, together a proof of correctness $z_{\mathsf{partialPf}}$, obtained from the trustees. When the protocol is honestly executed by n voters, the resulting frame should have a substitution σ such that for all $1 \leq i \leq n$ we have

$$
\begin{aligned}
x_{pk}\sigma &= \mathsf{pk}(sk)\\
\pi_i(\hat{y})\sigma &= (\!|\mathsf{penc}(\mathsf{pk}(sk), m_i, s_i), \mathsf{ballotPf}(\mathsf{pk}(sk), m_i, s_i, \mathsf{penc}(\mathsf{pk}(sk), m_i, s_i))|\!)\\
z_{\mathsf{partial}}\sigma &= \mathsf{partial}(sk, z_{\mathsf{tally}})\sigma\\
z_{\mathsf{partialPf}}\sigma &= \mathsf{partialPf}(sk, z_{\mathsf{tally}}, z_{\mathsf{partial}})\sigma\\
z_{\mathsf{tally}}\sigma &= \pi_1(\pi_1(\hat{y}\sigma)) * \cdots * \pi_1(\pi_n(\hat{y}\sigma))
\end{aligned}
$$

Analysis: Individual and universal verifiability

Given $m \in \mathbb{N}$, the tests Φ^{IV} and Φ^{UV}_m, defined below, are introduced for verifiability purposes.

$$
\begin{aligned}
\Phi^{IV} &\mathrel{\widehat{=}} y =_E r_{ballot}\\
\Phi^{UV}_m &\mathrel{\widehat{=}} z_{\mathsf{tally}} =_E \pi_1(\pi_1(\hat{y})) * \cdots * \pi_1(\pi_m(\hat{y}))\\
&\quad \wedge \textstyle\bigwedge_{i=1}^{m}(\mathsf{checkBallotPf}(x_{\mathsf{pk}}, \pi_1(\pi_i(\hat{y})), \pi_2(\pi_i(\hat{y}))) =_E \mathsf{true})\\
&\quad \wedge \mathsf{checkPartialPf}(x_{\mathsf{pk}}, z_{\mathsf{tally}}, z_{\mathsf{partial}}, z_{\mathsf{partialPf}}) =_E \mathsf{true}\\
&\quad \wedge \pi_1(\hat{v}) + \cdots + \pi_m(\hat{v}) =_E \mathsf{dec}(z_{\mathsf{partial}}, z_{\mathsf{tally}})\\
&\quad \wedge \mathsf{snd}^m(\hat{v}) =_E \varnothing
\end{aligned}
$$

The test Φ^{IV} checks that the voter's ballot is recorded on the bulletin board and Φ^{UV}_m checks that the tally is correctly computed.

Theorem 3.2. $\langle V_{\mathsf{helios}}, A_{\mathsf{helios}}\rangle$ *satisfies individual and universal verifiability.*

Proof. Suppose $m \in \mathbb{N}$ and tests Φ^{IV}, Φ^{UV}_m are given above. We show for all names $s_1, \ldots, s_n$ that Conditions (3.1)–(3.4) of Definition 3.4 hold.

(3.1) Suppose C, B, σ, $\tilde{n}$, i and j are such that $C[\mathsf{VP}^+_{\mathsf{n}}(s_1, \ldots, s_n)](\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^* B$, $\varphi(B) \equiv \nu\tilde{n}.\sigma$, $\Phi^{IV}_i\sigma$ and $\Phi^{IV}_j\sigma$. It follows that $\pi_i(\hat{y})\sigma =_E r_{ballot,i}\sigma$ and similarly $\pi_j(\hat{y})\sigma =_E r_{ballot,j}\sigma$. But since record variables $r_{ballot,i}\sigma$ and $r_{ballot,j}\sigma$ contain distinct fresh nonces m_i and m_j created by name restriction in the ballot creation script B, it follows that $i = j$.

(3.2) We prove a stronger result: namely, the condition holds for all substitutions σ. Suppose σ is an arbitrary substitution such that $\Phi^{UV}_m\sigma$ and $\Phi^{UV}_m\{^{\hat{v}'}/_{\hat{v}}\}\sigma$ hold. Since $\bigwedge_{i=1}^m \mathsf{checkBallotPf}(x_{\mathsf{pk}}, \pi_1(\pi_i(\hat{y})), \pi_2(\pi_i(\hat{y}))\sigma =_E \mathsf{true}$, by inspection of the equational theory it must be the case that

$$\pi_1(\pi_i(\hat{y}))\sigma =_E \mathsf{penc}(M_{\mathsf{pk},i}, M_{\mathsf{rand},i}, s_i)$$

for some ground terms $M_{\mathsf{pk},i}$, $M_{\mathsf{rand},i}$ and name s_i, where $1 \leq i \leq m$. It follows that $z_{\mathsf{tally}}\sigma =_E \mathsf{penc}(M_{\mathsf{pk},1}, M_{\mathsf{rand},1}, s_1) * \cdots * \mathsf{penc}(M_{\mathsf{pk},m}, M_{\mathsf{rand},m}, s_m)$. Moreover, we have $\mathsf{checkPartialPf}(x_{\mathsf{pk}}, z_{\mathsf{tally}}, z_{\mathsf{partial}}, z_{\mathsf{partialPf}})\sigma =_E \mathsf{true}$, hence

$$z_{\mathsf{tally}}\sigma =_E \mathsf{penc}(\mathsf{pk}(M_{\mathsf{sk}}), M_{\mathsf{rand},1} \circ \cdots \circ M_{\mathsf{rand},m}, s_1 + \cdots + s_m)$$

where $\mathsf{pk}(M_{\mathsf{sk}}) =_E M_{\mathsf{pk},1} =_E \cdots =_E M_{\mathsf{pk},m}$ for some term M_{sk}. It also follows that

$$z_{\mathsf{partial}}\sigma =_E \mathsf{partial}(M_{\mathsf{sk}}, z_{\mathsf{tally}})\sigma$$

We have $\mathsf{dec}(z_{\mathsf{partial}}, z_{\mathsf{tally}})\sigma =_E \pi_1(\hat{v}) + \cdots + \pi_m(\hat{v})\sigma =_E \pi_1(\hat{v}') + \cdots + \pi_m(\hat{v}')\sigma =_E s_1 + \cdots + s_m$

and since $\mathsf{snd}^m(\hat{v})\sigma =_E \mathsf{snd}^m(\hat{v}')\sigma =_E \varnothing$, it must be the case that

$$\hat{v}\sigma =_E (\!|\pi_1(\hat{v}), \ldots, \pi_m(\hat{v})|\!)\sigma \simeq (\!|s_1, \ldots, s_m|\!) \simeq (\!|\pi_1(\hat{v}'), \ldots, \pi_m(\hat{v}')|\!)\sigma =_E \hat{v}'\sigma$$

It follows immediately that $\hat{v}\sigma \simeq \hat{v}'\sigma$.

(3.3) Again, we will show that the condition holds for all substitutions. Suppose σ is an arbitrary substitution such that $\bigwedge_{1\leq i\leq n} \Phi_i^{IV}\{y_i/y\}\sigma$ and $\Phi_m^{UV}\sigma$ hold, where $n = m$. By $\bigwedge_{1\leq i\leq n} \Phi_i^{IV}\{\pi_i(\hat{y})/y\}\sigma$ it must be the case for all $1 \leq i \leq n$ that

$$\pi_i(\hat{y}) =_E (\!|\mathsf{penc}(\mathsf{pk}(sk), m_i, s_i), \mathsf{ballotPf}(\mathsf{pk}(sk), m_i, s_i, \mathsf{penc}(\mathsf{pk}(sk), m_i, s_i))|\!)$$

Since $n = m$ and $\Phi_m^{UV}\sigma$ holds, we have

$$z_{\mathsf{tally}}\sigma =_E \pi_1(\pi_1(\hat{y})) * \cdots * \pi_1(\pi_m(\hat{y})) =_E \mathsf{penc}(\mathsf{pk}(sk), m_1 \circ \cdots \circ m_n, s_1 + \cdots + s_n)$$

Moreover, $\mathsf{checkPartialPf}(x_{\mathsf{pk}}, z_{\mathsf{tally}}, z_{\mathsf{partial}}, z_{\mathsf{partialPf}})\sigma =_E \mathsf{true}$ and hence

$$z_{\mathsf{partial}}\sigma =_E \mathsf{partial}(\mathsf{pk}(sk), z_{\mathsf{tally}})\sigma$$

It follows that $\mathsf{dec}(z_{\mathsf{partial}}, z_{\mathsf{tally}})\sigma =_E s_1 + \cdots + s_m$. By $\Phi_m^{UV}\sigma$ we also have $\mathsf{dec}(z_{\mathsf{partial}}, z_{\mathsf{tally}})\sigma =_E \pi_1(\hat{v}) + \cdots + \pi_m(\hat{v})\sigma$ and $\mathsf{snd}^m(\hat{v})\sigma =_E \varnothing$. As before, we conclude $\hat{v}\sigma =_E (\!|\pi_1(\hat{v}), \ldots, \pi_m(\hat{v})|\!)\sigma \simeq (\!|s_1, \ldots, s_m|\!)$, hence $(\!|s_1, \ldots, s_m|\!) \simeq \hat{v}\sigma$

(3.4) The context C must marshal the election data onto the frame such that for all $1 \leq i \leq n$ we have $x_{pk}\sigma$, $\pi_i(\hat{y})\sigma$, $z_{\mathsf{partial}}\sigma$, $z_{\mathsf{partialPf}}\sigma$ and $z_{\mathsf{tally}}\sigma$ as defined above. Moreover, it defines $\hat{v}\sigma = (\!|s_1, \ldots, s_n|\!)$. For brevity, we omit formally defining C. □

Observe that Condition 3.2 cannot be proved where $\Phi_m^{UV} \mathrel{\widehat{=}} \pi_1(\hat{v}) + \cdots + \pi_m(\hat{v}) =_E \mathsf{dec}(z_{\mathsf{partial}}, z_{\mathsf{tally}})$ because, in general, given terms $M_1, \ldots, M_k, N_1, \ldots, N_k, U, V$ such that $M_1 + \cdots + M_k =_E \mathsf{dec}(U, V) =_E M_1 + \cdots + N_k$ it is not the case that $(\!|M_1, \ldots, M_k|\!) \simeq$

$(\!|N_1, \ldots, N_k|\!)$.

3.3 Security definition: Election verifiability

To fully capture *election verifiability*, the tests Φ^{IV} and Φ^{UV} must be supplemented with a test Φ^{EV}. We suppose that the public credentials of registered voters appear on the bulletin board and Φ^{EV} allows an observer to check that only these individuals (that is, those in possession of credentials) cast ballots, and at most one ballot is tallied per voter.

Definition 3.7 (Election verifiability)**.** *A voting specification* $\langle V, A\rangle$ satisfies election verifiability *if there exists a test* Φ^{IV}*, where for all* $m \in \mathbb{N}$ *there exists tests* Φ^{UV}_m, Φ^{EV}_m*, such that* $\mathrm{fn}(\Phi^{IV}) = \mathrm{fn}(\Phi^{UV}_m) = \mathrm{fn}(\Phi^{EV}_m) = \mathrm{rv}(\Phi^{UV}_m) = \mathrm{rv}(\Phi^{EV}_m) = \emptyset$ *and* $\mathrm{rv}(\Phi^{IV}) \subseteq \mathrm{rv}(\mathsf{R}(V))$*, and for all names* $s_1, \ldots, s_n$ *the conditions below hold. Let* $\tilde{r} = \mathrm{rv}(\Phi^{IV})$*,* $\Phi^{IV}_i = \Phi^{IV}\{^{s_i}/_v, ^{\tilde{r}_i}/_{\tilde{r}}, ^{\pi_i(\hat{w})}/_w\}$ *and* $X = \mathrm{fv}(\Phi^{EV}_m)\backslash\mathrm{dom}(\mathsf{VP}^+_{\mathsf{n}}(s_1, \ldots, s_n))$.

Soundness. *For all contexts* C*, such that* $C[\mathsf{VP}^+_{\mathsf{n}}(s_1, \ldots, s_n)]\ (\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*\ B$ *and* $\varphi(B) \equiv \nu\tilde{n}.\sigma$ *for some process* B*, substitution* σ *and names* $\tilde{n}$*, we have Conditions (3.1)–(3.3) and Conditions (3.5) – (3.7) hold.*

$$\Phi^{EV}_m\sigma \wedge \Phi^{EV}_m\{^{x'}/_x \mid x \in X\backslash\hat{y}\}\sigma \Rightarrow \hat{w}\sigma \simeq \hat{w}'\sigma \tag{3.5}$$

$$\Phi^{EV}_m\sigma \wedge \Phi^{EV}_m\{^{x'}/_x \mid x \in X\backslash\hat{w}\}\sigma \Rightarrow \hat{y}\sigma \simeq \hat{y}'\sigma \tag{3.6}$$

$$\bigwedge_{1\le i\le n} \Phi^{IV}_i\{^{\pi_i(\hat{y})}/_y\}\sigma \wedge \Phi^{EV}_n\{^{\hat{w}'}/_{\hat{w}}\}\sigma \wedge \mathsf{snd}^n(\hat{w})\sigma =_E \varnothing \Rightarrow \hat{w}\sigma \simeq \hat{w}'\sigma \tag{3.7}$$

Effectiveness. *There exists a context* C*, process* B*, substitution* σ *and names* $\tilde{n}$ *such that* $C[\mathsf{VP}^+_{\mathsf{n}}(s_1, \ldots, s_n)]\ (\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*\ B$*,* $\varphi(B) \equiv \nu\tilde{n}.\sigma$ *and Condition (3.8) holds.*

$$\bigwedge_{1\le i\le n} \Phi^{IV}_i\{^{\pi_i(\hat{y})}/_y\}\sigma \wedge \Phi^{UV}_m\sigma \wedge \Phi^{EV}_m\sigma \tag{3.8}$$

An observer should verify that the test Φ_m^{EV} holds when instantiated with the public bulletin board, to ensure that ballots were cast by eligible voters. Given ballots $\hat{y}\sigma$ provided by the environment, Condition (3.5) ensures that Φ_m^{EV} succeeds for only one tuple of public voter credentials. Similarly, given credentials $\hat{w}\sigma$, Condition (3.6) ensures that only one tuple of ballots $\hat{y}\sigma$ are accepted by Φ_m^{EV} (observe that for such a strong requirement to hold, we expect the voting specification's frame to contain a public key to root trust). Condition (3.7) ensures that if the bulletin board contains ballots cast by voters with public credentials $\hat{w}\sigma$, then Φ_m^{EV} holds only on a permutation of these credentials. Finally, the effectiveness condition is similar to Condition (3.4) of the previous section.

Case studies: Raising hands, FOO and Helios 2.0. Neither FOO nor Helios use public voting credentials in a manner suitable for eligibility verifiability. In FOO, the administrator is responsible for ensuring eligibility, that is, checking the validity of the voter's ballots; whereas in Helios, there are no public voting credentials. It follows immediately that Condition (3.6), in particular, cannot be satisfied. By comparison, the raising hands protocol (Example 3.1) does define suitable public voting credentials and satisfies our definition.

Proposition 3.1. *The raising hands protocol* $\langle A_{\mathsf{ex}}, V_{\mathsf{ex}} \rangle$ *satisfies election verifiability.*

The proof of this result appears in Appendix A.1.

3.4 Case study: JCJ-Civitas

The protocol by Juels, Catalano & Jakobsson [JCJ02, JCJ05, JCJ10] is based upon mixnets and was implemented by Clarkson, Chong & Myers [CCM08, CCM07] as an open-source voting system called Civitas.

3.4.1 Protocol description

An election is created by naming a set of registrars and talliers. The protocol is divided into four phases: *setup*, *registration*, *voting* and *tallying*. We now detail the steps of the protocol, starting with the setup phase.

1. The registrars, respectively talliers, run a protocol which constructs a public key pair and distributes a share of the secret part amongst the registrars, respectively talliers. The public part $\mathsf{pk}(sk_R)$, respectively $\mathsf{pk}(sk_T)$, of the key is published. The registrars also construct a distributed signing key pair ssk_R, $\mathsf{pk}(ssk_R)$.

The registration phase then proceeds as follows.

2. The registrars generate and distribute voter credentials. These consist of a private part d and a public part $\mathsf{penc}(\mathsf{pk}(sk_R), m'', d)$, that is, the probabilistic encryption of d under the registrars' public key $\mathsf{pk}(sk_R)$. This is done in a distributed manner, so that no individual registrar learns the value of any private credential d.

3. The registrars publish the signed public voter credentials.

4. The registrars announce the candidate list $\tilde{t} = (t_1, \ldots, t_l)$.

The protocol then enters the voting phase.

5. Each voter selects her vote $s \in \tilde{t}$ and computes two ciphertexts $M = \mathsf{penc}(\mathsf{pk}(sk_T), m, s)$ and $M' = \mathsf{penc}(\mathsf{pk}(sk_R), m', d)$, where m, m' are nonces. M contains her vote and M' her credential. In addition, the voter constructs a non-interactive zero-knowledge proof of knowledge (or signature proof of knowledge) demonstrating the correct construction of her ciphertexts and the validity of her chosen candidate. (The proof provides protection against coercion resistance, by preventing forced abstention attacks via a *write in*, and binds the two ciphertexts for eligibility verifiability.) The voter derives her ballot as the triple, consisting of her ciphertexts and signature proof of knowledge, and posts it to the bulletin board.

After some predefined deadline the tallying phase commences.

6. The talliers read the n' ballots posted to the bulletin board by voters (that is, the triples consisting of the two ciphertexts and the signature proof of knowledge) and discards any entries for which the signature proof of knowledge does not hold.

7. Elimination of re-votes is performed on the ballots using pairwise *plaintext equality tests* (PET) on the ciphertexts containing private voter credentials. (A PET [JJ00] is a cryptographic predicate which allows a key holder to provide a proof that two ciphertexts contain the same plaintext.) Re-vote elimination is performed in a verifiable manner with respect to some publicly defined policy, for example, by the order of ballots on the bulletin board.

8. The talliers perform a verifiable re-encryption mix on the ciphertexts in the ballots (recall that the ballots include a vote ciphertext and a public credential ciphertext; the link between these ciphertexts is preserved by the mix.) The mix ensures that a voter cannot trace her vote, allowing the protocol to achieve coercion resistance.

9. The talliers perform a verifiable re-encryption mix on the list of public credentials published by the registrars. This mix anonymises public voter credentials, breaking any link with the voter for privacy purposes.

10. Ballots based upon invalid credentials are weeded using PETs between the mixed ciphertexts provided by voters and the mixed public credentials. (Using PETs the correctness of weeding is verifiable.)

11. Finally, the talliers perform a verifiable decryption and publish the result.

3.4.2 Equational theory

The protocol uses the homomorphic ElGamal encryption scheme. Accordingly, we adopt the signature and associated equational theory from the Helios case study. We model the

signature proof of knowledge demonstrating correct construction of the voter's ciphertexts, re-encryption and PETs by the equations

$$\begin{aligned}
&\mathsf{checkBallot}(\mathsf{ballotPf}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, x_{\mathsf{plain}}, x'_{\mathsf{pk}}, x'_{\mathsf{rand}}, x'_{\mathsf{plain}}),\\
&\qquad\qquad \mathsf{penc}(x_{\mathsf{pk}}, x_{\mathsf{rand}}, x_{\mathsf{plain}}), \mathsf{penc}(x'_{\mathsf{pk}}, x'_{\mathsf{rand}}, x'_{\mathsf{plain}})) = \mathsf{true}\\
&\mathsf{renc}(y_{\mathsf{rand}}, \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{plain}})) = \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}} \circ y_{\mathsf{rand}}, x_{\mathsf{plain}})\\
&\mathsf{pet}(\mathsf{petPf}(x_{\mathsf{sk}}, \mathit{ciph}, \mathit{ciph}'), \mathit{ciph}, \mathit{ciph}') = \mathsf{true}
\end{aligned}$$

where $\mathit{ciph} \mathrel{\hat{=}} \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{plain}})$ and $\mathit{ciph}' \mathrel{\hat{=}} \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x'_{\mathsf{rand}}, x_{\mathsf{plain}})$. In addition, we consider verifiable re-encryption mixnets, and introduce for each permutation χ on $\{1, \ldots, n\}$, where $n \in \mathbb{N}$, the equation:

$$\begin{aligned}
&\mathsf{checkMix}(\mathsf{mixPf}(x_{\mathsf{ciph},1}, \ldots, x_{\mathsf{ciph},n}, \mathit{ciph}_1, \ldots, \mathit{ciph}_n, z_{\mathsf{rand},1}, \ldots, z_{\mathsf{rand},n}),\\
&\qquad\qquad x_{\mathsf{ciph},1}, \ldots, x_{\mathsf{ciph},n}, \mathit{ciph}_1, \ldots, \mathit{ciph}_n) = \mathsf{true}
\end{aligned}$$

where $\mathit{ciph}_i \mathrel{\hat{=}} \mathsf{renc}(z_{\mathsf{rand},i}, x_{\mathsf{ciph},\chi(i)})$. We also define re-encryption of pairs of ciphertexts, and introduce for each permutation χ on $\{1, \ldots, n\}$,= where $n \in \mathbb{N}$, the equation:

$$\begin{aligned}
&\mathsf{checkMixPair}(\mathsf{mixPairPf}((\!|x_1, x'_1|\!), \ldots, (\!|x_n, x'_n|\!), (\!|c_1, c'_1|\!), \ldots, (\!|c_n, c'_n|\!),\\
&\qquad (\!|z_1, z'_1|\!), \ldots, (\!|z_n, z'_n|\!)), (\!|x_1, x'_1|\!), \ldots, (\!|x_n, x'_n|\!), (\!|c_1, c'_1|\!), \ldots, (\!|c_n, c'_n|\!)) = \mathsf{true}
\end{aligned}$$

where $c_i \mathrel{\hat{=}} \mathsf{renc}(z_i, x_{\chi(i)})$ and $c'_i \mathrel{\hat{=}} \mathsf{renc}(z'_i, x'_{\chi(i)})$.

3.4.3 Model in applied pi

We make the following trust assumptions for verifiability:

- The voter is able to construct her ballot; that is, she is able to generate nonces m, m', construct her ciphertexts and generate a signature proof of knowledge.
- The registrars construct distinct credentials d for each voter and construct the

voter's public credential correctly. (The latter assumption can be dropped if the registrars provides a designated verified proof that the public credential is correctly formed [JCJ02, JCJ05, JCJ10].) The registrars keep the private part of the signing key secret.

Although neither voters nor observers can verify that the registrars adhere to such expectations, they trust them because trust is distributed. The trusted components are modelled by the voting process specification $\langle A_{\mathsf{jcj}}, V_{\mathsf{jcj}} \rangle$ (Definition 3.8). The context A_{jcj} distributes private keys on a private channel, launches an unbounded number of registrar processes, and publishes the public keys of both the registrars and talliers. The registrar R constructs a fresh private credential d and sends the private credential along with the signed public part (that is, $\mathsf{sign}(ssk_R, \mathsf{penc}(x_{pk_R}, m'', d))$) to the voter; the registrar also publishes the signed public credential on the bulletin board. The voter V_{jcj} receives the private and public credentials from the registrar and constructs her ballot; that is, the pair of ciphertexts and a signature proof of knowledge demonstrating their correct construction.

Definition 3.8. *The voting process specification $A_{\mathsf{jcj}}, V_{\mathsf{jcj}}$ is defined where:*

$$
\begin{aligned}
A_{\mathsf{jcj}} &\hat{=} \nu\, a, ssk_R.(!R \mid \{\mathsf{pk}(sk_R)/_{x_{\mathsf{pk_R}}}, \mathsf{pk}(ssk_R)/_{x_{spk_R}}, \mathsf{pk}(sk_T)/_{x_{\mathsf{pk_T}}}\} \mid \_) \\
V_{\mathsf{jcj}} &\hat{=} \nu\, m, m'.a(x_{cred}).\text{let } ciph = \mathsf{penc}(x_{pk_T}, m, v) \text{ in} \\
&\quad \text{let } ciph' = \mathsf{penc}(x_{pk_R}, m', \pi_1(x_{cred})) \text{ in} \\
&\quad \text{let } spk = \mathsf{ballotPf}(x_{pk_T}, m, v, x_{pk_R}, m', \pi_1(x_{cred})) \text{ in} \\
&\quad \overline{c}\langle (\!| ciph, ciph', spk |\!) \rangle \\
R &\hat{=} \nu\, d, m''.\text{let } sig = \mathsf{sign}(ssk_R, \mathsf{penc}(x_{pk_R}, m'', d)) \text{ in } \overline{a}\langle (\!| d, sig |\!) \rangle.\overline{c}\langle sig \rangle
\end{aligned}
$$

At the end of the election, the bulletin board is represented by a frame. In our formalism, we expect the frame to contain the substitution σ which defines the voters' public credentials as $\hat{w}$, the registrars' public keys as x_{pk_R} and x_{spk_R}, and talliers' public key as x_{pk_T}. A tuple $\hat{y}$ containing triples representing each voter's ciphertexts and signature

proofs of knowledge. The mixed re-encryptions of the voter's ciphertexts $z_{\mathsf{bal},1}, \ldots, z_{\mathsf{bal},n}$, along with a proof $z_{\mathsf{mixPairPf}}$ that the mix was performed correct. For verifiable decryption, we assume $z_{\mathsf{partial},i}$ is defined as a partial decryption associated with the proof $z_{\mathsf{partialPf},i}$. For the purposes of eligibility verifiability, we also expect the mixed re-encryptions of the voter's public credentials $z_{\mathsf{cred},1}, \ldots, z_{\mathsf{cred},1}$, along with a proof of correctness z_{mixPf}. For convenience, a reordering $\bar{z}_{\mathsf{cred},1}, \ldots, \bar{z}_{\mathsf{cred},n}$ of these re-encryptions is also computed. Finally, we expect PET proofs $z_{\mathsf{petPf},1}, \ldots, z_{\mathsf{petPf},n}$ for the re-encryptions of the ciphertext constructed by the voter on her private credential (that is, the output of the verifiable mix in Step 8 of the protocol) and the re-encryptions of the voter's public credential constructed by the registrars (that is, the output of the mix in Step 9), such that the PET holds, that is, the pair of ciphertexts contain the same private credential. Accordingly, given n voters we expect σ to be such that for all $1 \leq i \leq n$ we have:

$$
\begin{aligned}
\pi_i(\hat{w})\sigma &= \mathsf{sign}(ssk_R, c_i'') \\
x_{pk_R}\sigma &= \mathsf{pk}(sk_R) \\
x_{spk_R}\sigma &= \mathsf{pk}(ssk_R) \\
x_{pk_T}\sigma &= \mathsf{pk}(sk_T) \\
\pi_i(\hat{y})\sigma &= (\!|c_i, c_i', \mathsf{ballotPf}(\mathsf{pk}(sk_T), m_i, s_i, \mathsf{pk}(sk_R), m_i', d_i)|\!) \\
z_{\mathsf{bal},i}\sigma &= (\!|\mathsf{renc}(\bar{m}_i, c_{\chi(i)}), \mathsf{renc}(\bar{m}_i', c_{\chi(i)}')|\!) \\
z_{\mathsf{mixPairPf}}\sigma &= \mathsf{pfMixPair}((\!|c_1, c_1'|\!), \ldots, (\!|c_n, c_n'|\!), (\!|\mathsf{renc}(\bar{m}_1, c_{\chi(1)}), \mathsf{renc}(\bar{m}_1', c_{\chi(1)}')|\!), \\
&\qquad \ldots, (\!|\mathsf{renc}(\bar{m}_n, c_{\chi(n)}), \mathsf{renc}(\bar{m}_n', c_{\chi(n)}')|\!), (\!|\bar{m}_1, \bar{m}_1'|\!), \ldots, (\!|\bar{m}_n, \bar{m}_n'|\!)) \\
z_{\mathsf{partial},i}\sigma &= \mathsf{partial}(sk_T, \mathsf{renc}(\bar{m}_i, c_{\chi(i)})) \\
z_{\mathsf{partialPf},i}\sigma &= \mathsf{partialPf}(sk_T, \mathsf{renc}(\bar{m}_i, c_{\chi(i)}), \mathsf{partial}(sk_T, \mathsf{renc}(\bar{m}_i, c_{\chi(i)}))) \\
z_{\mathsf{cred},i}\sigma &= \mathsf{renc}(\bar{m}_i'', c''_{\chi'(i)}) \\
\bar{z}_{\mathsf{cred},i}\sigma &= \mathsf{renc}(\bar{m}''_{\chi(\chi'^{-1}(i))}, c''_{\chi(i)}) \\
z_{\mathsf{mixPf}}\sigma &= \mathsf{pfMix}(c_1'', \ldots, c_n'', \mathsf{renc}(\bar{m}_1'', c''_{\chi'(1)}), \ldots, \mathsf{renc}(\bar{m}_n'', c''_{\chi'(n)}), \bar{m}_1'', \ldots, \bar{m}_n'') \\
z_{\mathsf{petPf},i}\sigma &= \mathsf{petPf}(sk_R, \mathsf{renc}(\bar{m}_i', c'_{\chi(i)}), \mathsf{renc}(\bar{m}''_{\chi(\chi'^{-1}(i))}, c''_{\chi(i)}))
\end{aligned}
$$

where $c_i \mathrel{\hat{=}} \mathsf{penc}(\mathsf{pk}(sk_T), m, s_i)$, $c'_i \mathrel{\hat{=}} \mathsf{penc}(\mathsf{pk}(sk_R), m', d_i)$, $c''_i \mathrel{\hat{=}} \mathsf{penc}(\mathsf{pk}(sk_R), m'', d_i)$ and χ, χ' are permutations on $\{1, \ldots, n\}$.

3.4.4 Analysis: Election verifiability

We assume record variables $\tilde{r} = (r_{cred}, r_m, r_{m'}) = \mathrm{rv}(\mathsf{R}(V))$ (corresponding to the variable x_{cred} and names m, m' in the process V). Accordingly, given $m \in \mathbb{N}$ we define:

$$
\begin{aligned}
\Phi^{IV} \mathrel{\hat{=}}\ & y =_E (\!|\mathsf{penc}(x_{\mathsf{pk_T}}, r_m, v), \mathsf{penc}(x_{\mathsf{pk_R}}, r_{m'}, \pi_1(r_{cred})),\\
& \qquad \mathsf{ballotPf}(x_{\mathsf{pk_T}}, r_m, v, x_{\mathsf{pk_R}}, r_{m'}, \pi_1(r_{cred}))|\!) \wedge w = \pi_2(r_{cred})\\
\Phi^{UV}_m \mathrel{\hat{=}}\ & \mathsf{checkMixPair}(z_{\mathsf{mixPairPf}}, (\!|\pi_1(\pi_1(\hat{y})), \pi_2(\pi_1(\hat{y}))|\!), \ldots, (\!|\pi_1(\pi_m(\hat{y})), \pi_2(\pi_m(\hat{y}))|\!),\\
& \qquad z_{\mathsf{bal},1}, \ldots, z_{\mathsf{bal},m}) =_E \mathsf{true}\\
& \wedge (\!|\mathsf{dec}(z_{\mathsf{partial},1}, \pi_1(z_{\mathsf{bal},1})), \ldots, \mathsf{dec}(z_{\mathsf{partial},m}, \pi_1(z_{\mathsf{bal},m}))|\!) =_E \hat{v}\\
& \wedge \textstyle\bigwedge_{i=1}^{m} \mathsf{checkPartialPf}(x_{\mathsf{pk_T}}, \pi_1(z_{\mathsf{bal},i}), z_{\mathsf{partial},i}, z_{\mathsf{partialPf},i}) =_E \mathsf{true}\\
\Phi^{EV}_m \mathrel{\hat{=}}\ & \textstyle\bigwedge_{i=1}^{m} \mathsf{checkBallot}(\pi_3(\pi_i(\hat{y})), \pi_1(\pi_i(\hat{y})), \pi_2(\pi_i(\hat{y}))) =_E \mathsf{true}\\
& \wedge \mathsf{checkMixPair}(z_{\mathsf{mixPairPf}}, (\!|\pi_1(\pi_1(\hat{y})), \pi_2(\pi_1(\hat{y}))|\!), \ldots, (\!|\pi_1(\pi_m(\hat{y})), \pi_2(\pi_m(\hat{y}))|\!),\\
& \qquad z_{\mathsf{bal},1}, \ldots, z_{\mathsf{bal},m}) =_E \mathsf{true}\\
& \wedge \textstyle\bigwedge_{i=1}^{m} \mathsf{pet}(z_{\mathsf{petPf},i}, \pi_2(z_{\mathsf{bal},i}), \bar{z}_{\mathsf{cred},i}) =_E \mathsf{true}\\
& \wedge (\!|z_{\mathsf{cred},1}, \ldots, z_{\mathsf{cred},m}|\!) \simeq (\!|\bar{z}_{\mathsf{cred},1}, \ldots, \bar{z}_{\mathsf{cred},m}|\!)\\
& \wedge \mathsf{checkMix}(z_{\mathsf{mixPf}}, \mathsf{getmsg}(\pi_1(\hat{w})), \ldots, \mathsf{getmsg}(\pi_m(\hat{w})), z_{\mathsf{cred},1}, \ldots, z_{\mathsf{cred},m}) =_E \mathsf{true}\\
& \wedge \textstyle\bigwedge_{i=1}^{m} \mathsf{checksign}(x_{spk_R}, \pi_i(\hat{w})) =_E \mathsf{true}\\
& \textstyle\bigwedge_{i=1}^{m} \pi_4(\hat{y}) =_E \varnothing \wedge \mathsf{snd}^m(\hat{w}) =_E \varnothing \wedge \mathsf{snd}^m(\hat{y}) =_E \varnothing
\end{aligned}
$$

The test Φ^{IV} checks that the voter's ballot and public credential are recorded on the bulletin board. The test Φ^{UV}_m checks that the tally is correctly computed; that is, the mix is checked, the validity of partial decryptions have been verified and the decrypted tally corresponds to the election outcome. Finally, the test Φ^{EV}_m checks that only eligible ballots are considered; that is, ballots are correctly formed, mixes have been handled in a suitable manner, PETs have been verified and only authentic public voter credentials are considered.

Theorem 3.3. *$\langle A_{\mathsf{jcj}}, V_{\mathsf{jcj}} \rangle$ satisfies election verifiability.*

The proof of Theorem 3.3 is presented in Appendix A.2.

3.5 Summary

This chapter presents a symbolic definition of election verifiability which allows the precise identification of voting system components that need to be trusted for the purposes of verifiability. The suitability of systems can then be evaluated and compared on the basis of trust assumptions. The consideration of eligibility verifiability is of particular interest, since it provides an essential mechanism for detecting ballot stuffing, although it is often neglected within the literature and only satisfied by a few protocols. Our definition of election verifiability has been successfully used to evaluate three electronic voting protocols, namely: FOO, which uses blind signatures; Helios 2.0, which is based upon homomorphic encryption; and JCJ-Civitas, which uses mixnets. For each of these protocols the trust assumptions required for election verifiability are discussed. Helios 2.0 and JCJ-Civitas protocols have been implemented and deployed, therefore demonstrating the suitability of the framework for analysing real-world election systems.

4

Anonymity in Direct Anonymous Attestation[†]

Overview. *A definition of user-controlled anonymity is introduced in the context of Direct Anonymous Attestation schemes. The definition is expressed as an equivalence property suited to automated reasoning using ProVerif. The practicality of the definition is demonstrated by analysing the RSA-based Direct Anonymous Attestation protocol by Brickell, Camenisch & Chen. The analysis discovers a vulnerability which can be exploited by a passive adversary and, under weaker assumptions, corrupt administrators. A security fix is identified and the revised scheme is shown to satisfy our definition of user-controlled anonymity.*

Trusted computing allows commodity computers to provide cryptographic assurances about their behaviour. At the core of the architecture is a hardware device called the Trusted Platform Module (TPM). The TPM uses shielded memory to store cryptographic keys, and other sensitive data, which can be used to achieve security objectives. In particular, the chip can measure and report its state, and authenticate. Cryptographic operations, by their nature, may reveal a platform's identity and as a consequence the TPM threatens privacy. Brickell, Camenisch & Chen [BCC04] have introduced the notion of Direct Anonymous Attestation (DAA) to overcome these privacy concerns. More precisely, DAA is a remote authentication mechanism for trusted platforms which provides user-

[†]This chapter is partly based upon [SRC07, DRS08]. A preliminary analysis of the RSA-based DAA protocol (Section 4.4) – which discovered an attack and presented a fix – originally appeared in [SRC07], and an initial analysis (Section 4.4.5) of the fixed scheme was considered in [DRS08].

controlled anonymity and user-controlled traceability. The concept is based upon group signatures with stronger anonymity guarantees; in particular, the identity of a signer can never be revealed, but signatures may be linked with the signer's consent, and signatures produced by compromised platforms can be identified. A DAA scheme considers a set of *hosts*, *issuers*, *TPMs*, and *verifiers*; the host and TPM together form a *trusted platform* or *signer*. DAA protocols proceed as follows. A host requests membership to a group provided by an issuer. The issuer authenticates the host as a trusted platform and grants an *attestation identity credential* (occasionally abbreviated *credential*). The host can now produce signatures using the credential, thereby permitting a verifier to authenticate the host as a group member and therefore a trusted platform.

Brickell, Chen & Li [BCL08b, BCL09] characterise the following properties for Direct Anonymous Attestation schemes:

- *User-controlled anonymity.*
  - *Privacy.* The identity of a signer cannot be revealed from a signature.
  - *Unlinkability.* Signatures cannot be linked without the signer's consent.
- *User-controlled traceability.*
  - *Unforgeability.* Signatures cannot be produced without a TPM.
  - *Basename linkability.* Signatures are linkable with the signer's consent.
- *Correctness.* Valid signatures can be verified and, where applicable, linked.

The contrasting nature of anonymity and traceability properties aims to provide a balance between the privacy demands of users and the accountability needs of administrators.

The first concrete Direct Anonymous Attestation scheme was introduced by Brickell, Camenisch & Chen [BCC04] and is based upon RSA. The RSA-based DAA protocol is reliant on the strong RSA and decisional Diffie-Hellman assumptions. However, some

users are uncomfortable with these assumptions. This motivated the work of Brickell, Chen & Li [BCL08a, BCL09] who provide the first ECC-based DAA protocol using symmetric pairing. This scheme is reliant on the LRSW [LRSW00] and decisional Bilinear Diffie-Hellman assumptions. Moreover, ECC-based schemes are more efficient and therefore better suited to devices with limited resources, such as the TPM. Chen, Morrissey & Smart [CMS08a, CMS08b] extend the scheme based upon symmetric pairing to an asymmetric setting to improve efficiency; however, Li discovered a vulnerability which violates user-controlled traceability and Chen & Li propose a fix [CL10]. Chen, Morrissey & Smart [CMS09] identified further traceability attacks against the asymmetric scheme and proposed a fix. In addition, Chen, Morrissey & Smart [CMS10] have found traceability attacks against the symmetric pairing based scheme [BCL08a, BCL09] and the original RSA-based scheme [BCC04]. These attacks allow a malicious host to extract the TPM's secret `tsk`, if the protocol is implemented in hardware without *stage control mechanisms*; the host can then violate user-controlled traceability by forging signatures. However, since the TPM provides stage control protection, there is no practical threat in the current setting; but, we should still consider these attacks interesting because it identifies settings in which DAA protocols cannot be deployed (for example, in other trusted computing settings which do not use the TPM). We remark that the analysis of user-controlled traceability of the RSA-based scheme by Backes, Maffei & Unruh [BMU08] could not identify this attack because they consider a setting where the host and TPM are both honest. An optimisation of the fixed asymmetric scheme has been proposed by Chen, Page & Smart [CPS10] and three further ECC-based DAA protocols have been defined: Chen & Feng [CF08], Brickell & Li [BL09a, BL09b] and Chen [Che10, Che11].

Chapter contribution

A definition of user-controlled anonymity is presented as an equivalence property which is suited to automated reasoning using ProVerif. Informally, the definition asserts that an adversary cannot distinguish between signatures produced by two distinct signers, even

when the adversary controls the issuer and has observed previous signatures produced by each signer.

The application of the definition is demonstrated by analysing user-controlled anonymity in the RSA-based DAA protocol [BCC04]. Support for the RSA-based scheme is mandated by the TPM specification version 1.2 [TCG07] which has been defined as an ISO/IEC international standard [Int09]. Moreover, estimates suggest that the TPM has been embedded in over 300 million computers [Tru09] (although, some experts claim only 15 million of these TPMs have been turned on [Mar08, §6]). This demonstrates the suitability of this framework for analysing schemes which have been deployed in the real-world.

The analysis of RSA-based DAA discovers a vulnerability in the protocol which allows an adversary to violate user-controlled anonymity. As a consequence, applications which use RSA-based DAA as a building block – for example, the peer-to-peer networking scheme by Balfe, Lakhani & Paterson [BLP05a, BLP05b] and the authentication scheme by Leung & Mitchell [LM07] – are also flawed. A fix is identified, and the revised RSA-based DAA protocol is shown to be secure.

Structure of this chapter. Section 4.1 recalls the variant of the applied pi calculus introduced by [Bla04, BAF08]. Section 4.2 defines Direct Anonymous Attestation protocols, and presents a formalisation in the applied pi calculus. Section 4.3 defines user-controlled anonymity as an observational equivalence property, and the definition is used in Section 4.4 to analyse the RSA-based DAA protocol. Finally, a summary is presented in Section 4.5.

4.1 Preliminaries: Calculus of ProVerif

The calculus presented here is a combination of the applied pi calculus (Chapter 2) with one of its dialects [Bla04, BAF08]. This variant is particularly useful due to the automated

support provided by ProVerif.

4.1.1 Syntax and semantics

The calculus assumes an infinite set of names, an infinite set of variables and a signature Σ consisting of a finite set of function symbols each with an associated arity. An explicit distinction is made between constructors (termed functions in Chapter 2) and destructors within the signature. We write f for a constructor, g for a destructor, and h for either a constructor or destructor. Terms are built by applying constructors to names, variables and other terms (Figure 4.1). As usual, the signature Σ is equipped with an equation theory E.

The semantics of a destructor g of arity l is given by a finite set $\mathrm{def}_\Sigma(g)$ of rewrite rules $g(M'_1, \ldots, M'_l) \to M'$, where $M'_1, \ldots, M'_l, M'$ are terms that contain only constructors and variables; moreover, the variables of M' are bound in $M'_1, \ldots, M'_l$, and variables are subject to renaming. The term $g(M_1, \ldots, M_l)$ is defined if and only if there exists a substitution σ and a rewrite rule $g(M'_1, \ldots, M'_l) \to M'$ in $\mathrm{def}_\Sigma(g)$ such that $M_i = M'_i\sigma$ for all $i \in \{1, \ldots, l\}$, and in this case $g(M_1, \ldots, M_l)$ is defined as $M'\sigma$.

The grammar for processes is presented in Figure 4.1. The process let $x = D$ in P else Q tries to evaluate D; if this succeeds, then x is bound to the result and P is executed, otherwise Q is executed. For convenience, the statement let $x = D$ in P else Q may be abbreviated as let $x = D$ in P when Q is the null process.

The syntax does not include the conditional if $M = N$ then P else Q, but this can be defined as let $x = \mathsf{eq}(M, N)$ in P else Q, where x is a fresh variable and eq is a binary destructor with the rewrite rule $\mathsf{eq}(x, x) \to x$. We always include this destructor in Σ. In addition, evaluation contexts forbid holes under the scope of a term evaluation. The rest of the syntax is standard (see Chapter 2 and [Bla04, BAF08]). The bracketing conventions defined in Section 2.1 will be adopted with an additional assumption that the expression $P \mid Q \mid R$ is bracketed as $P \mid (Q \mid R)$. Although $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$, this

convention simplifies our compiler design in Chapter 5.

The rules of Figure 4.2 axiomatise the reduction relation for processes ($\rightarrow_\Sigma$), thus defining the operational semantics of our calculus. Auxiliary rules define term evaluation ($\Downarrow_\Sigma$) and the structural congruence relation ($\equiv$). Both $\equiv$ and $\rightarrow_\Sigma$ are defined only on closed processes. We write $\rightarrow_\Sigma^*$ for the reflexive and transitive closure of $\rightarrow_\Sigma$, and $\rightarrow_\Sigma^*\equiv$ for its union with $\equiv$. When Σ is clear from the context, we abbreviate $\rightarrow_\Sigma$ and $\Downarrow_\Sigma$ to $\rightarrow$ and $\Downarrow$, respectively.

4.1.2 Biprocesses

The calculus provides a notation for modelling pairs of processes that have the same structure and differ only by the terms and term evaluations that they contain. We call such a pair of processes a *biprocess*. The grammar for the calculus is a simple extension of the grammar of Figure 4.1, with additional cases so that $\mathrm{diff}[M, M']$ is a term and $\mathrm{diff}[D, D']$ is a term evaluation. The semantics for biprocesses include the rules in Figure 4.2, except for (Red I/O), (Red Fun 1), and (Red Fun 2) which are revised in Figure 4.3. We also extend the definition of contexts to permit the use of diff, and sometimes refer to contexts without diff as plain contexts.

Given a biprocess P, we define two processes $\mathrm{fst}(P)$ and $\mathrm{snd}(P)$, as follows: $\mathrm{fst}(P)$ is obtained by replacing all occurrences of $\mathrm{diff}[M, M']$ with M and $\mathrm{diff}[D, D']$ with D in P; and similarly, $\mathrm{snd}(P)$ is obtained by replacing $\mathrm{diff}[M, M']$ with M' and $\mathrm{diff}[D, D']$ with D' in P. We define $\mathrm{fst}(D)$, $\mathrm{fst}(M)$, $\mathrm{snd}(D)$, and $\mathrm{snd}(M)$ similarly.

4.1.3 Observational equivalence

Intuitively, processes P and Q are said to be observationally equivalent if they can output on the same channels, no matter what context they are placed inside. Formally, we write $P \downarrow_M$ when P can send a message on M, that is, when $P \equiv C[\overline{M'}\langle N\rangle.R]$ for some evaluation context $C[\_]$ such that $\mathrm{fn}(C) \cap \mathrm{fn}(M) = \emptyset$ and $\Sigma \vdash M = M'$. The definition of

Figure 4.1 Syntax for terms and processes

$M, N ::=$	terms
$a, b, c, \ldots, k, \ldots, m, n, \ldots, s$	name
x, y, z	variable
$f(M_1, \ldots, M_l)$	constructor application
$D ::=$	term evaluations
M	term
eval $h(D_1, \ldots, D_l)$	function evaluation
$P, Q, R ::=$	processes
0	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu\, a.P$	name restriction
$M(x).P$	message input
$\overline{M}\langle N\rangle.P$	message output
let $x = D$ in P else Q	term evaluation

Figure 4.2 Semantics for terms and processes

$M \Downarrow M$

eval $h(D_1, \ldots, D_n) \Downarrow N\sigma$
if $h(N_1, \ldots, N_n) \to N \in \mathrm{def}_\Sigma(h)$,
and σ is such that for all i, $D_i \Downarrow M_i$ and $\Sigma \vdash M_i = N_i\sigma$

$P \mid 0 \equiv P$	$P \equiv P$
$P \mid Q \equiv Q \mid P$	$Q \equiv P \Rightarrow P \equiv Q$
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$
$\nu\, a.\nu\, b.P \equiv \nu\, b.\nu\, a.P$	$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$
$\nu\, a.(P \mid Q) \equiv P \mid \nu\, a.Q$ if $a \notin \mathrm{fn}(P)$	$P \equiv Q \Rightarrow \nu\, a.P \equiv \nu\, a.Q$

$\overline{N}\langle M\rangle.Q \mid N'(x).P \to Q \mid P\{^M/_x\}$ (Red I/O)
if $\Sigma \vdash N = N'$

let $x = D$ in P else $Q \to P\{^M/_x\}$ (Red Fun 1)
if $D \Downarrow M$

let $x = D$ in P else $Q \to Q$ (Red Fun 2)
if there is no M such that $D \Downarrow M$

$!P \to P \mid !P$ (Red Repl)

$P \to Q \Rightarrow P \mid R \to Q \mid R$ (Red Par)

$P \to Q \Rightarrow \nu\, a.P \to \nu\, a.Q$ (Red Res)

$P' \equiv P, P \to Q, Q \equiv Q' \Rightarrow P' \to Q'$ (Red $\equiv$)

Figure 4.3 Generalised semantics for biprocesses

$$\overline{N}\langle M\rangle.Q \mid N'(x).P \;\rightarrow\; Q \mid P\{^{M}/_{x}\} \qquad \text{(Red I/O)}$$
if $\Sigma \vdash \mathrm{fst}(N) = \mathrm{fst}(N')$ and $\Sigma \vdash \mathrm{snd}(N) = \mathrm{snd}(N')$

$$\text{let } x = D \text{ in } P \text{ else } Q \rightarrow P\{^{\mathrm{diff}[M_1, M_2]}/_{x}\} \qquad \text{(Red Fun 1)}$$
if $\mathrm{fst}(D) \Downarrow M_1$ and $\mathrm{snd}(D) \Downarrow M_2$

$$\text{let } x = D \text{ in } P \text{ else } Q \rightarrow Q \qquad \text{(Red Fun 2)}$$
if there is no M_1 such that $\mathrm{fst}(D) \Downarrow M_1$ and
there is no M_2 such that $\mathrm{snd}(D) \Downarrow M_2$

observational equivalence [Bla04, BAF08] can now be recalled as follows.

Definition 4.1 (Observational equivalence)**.** Observational equivalence $\sim$ *is the largest symmetric relation* $\mathcal{R}$ *between closed processes such that* $P\ \mathcal{R}\ Q$ *implies:*

1. *if* $P \downarrow_M$*, then* $Q \downarrow_M$*;*
2. *if* $P \rightarrow P'$*, then* $Q \rightarrow Q'$ *and* $P'\ \mathcal{R}\ Q'$ *for some* Q'*;*
3. $C[P]\ \mathcal{R}\ C[Q]$ *for all evaluation contexts* $C[\_]$*.*

We define observational equivalence as a property of biprocesses.

Definition 4.2. *The closed biprocess* P satisfies observational equivalence *if* $\mathrm{fst}(P) \sim \mathrm{snd}(P)$*.*

It follows from the semantics of biprocess that if $P \rightarrow Q$ for some biprocesses P, Q, then $\mathrm{fst}(P) \rightarrow \mathrm{fst}(Q)$ and $\mathrm{snd}(P) \rightarrow \mathrm{snd}(Q)$. However, reductions in $\mathrm{fst}(P)$ or $\mathrm{snd}(P)$ do not necessarily imply biprocess reductions in P; that is, there exist biprocesses P such that $\mathrm{fst}(P) \rightarrow \mathrm{fst}(Q)$, but there is no such reduction $P \rightarrow Q$, and symmetrically for $\mathrm{snd}(P)$. For example, consider the biprocess $P = \overline{\mathrm{diff}[a, c]}\langle n\rangle \mid a(x)$, we have $\mathrm{fst}(P) \rightarrow 0$, but there is no reduction $P \rightarrow 0$. Blanchet, Abadi & Fournet [Bla04, BAF08] have shown that a biprocess P satisfies observational equivalence when reductions in $\mathrm{fst}(P)$ or $\mathrm{snd}(P)$ imply reductions in P. This proof technique is formalised using the notion of *uniformity*.

Definition 4.3 (Uniform)**.** *A biprocess* P *is* uniform *if for all processes* Q_1 *such that* $\mathrm{fst}(P) \rightarrow Q_1$*, then* $P \rightarrow Q$ *for some biprocess* Q*, where* $\mathrm{fst}(Q) \equiv Q_1$*, and symmetrically for* $\mathrm{snd}(P) \rightarrow Q_2$*.*

Definition 4.4 (Strong uniformity)**.** *A closed biprocess P* satisfies strong uniformity *if for all plain evaluation contexts C and biprocesses Q such that $C[P] \rightarrow^* \equiv Q$, then Q is uniform.*

Theorem 4.1 (Strong uniformity implies equivalence [BAF08])**.** *Given a closed biprocess P, if P satisfies strong uniformity, then P satisfies observational equivalence.*

4.2 Formalising DAA protocols

The concept of Direct Anonymous Attestation was defined by Brickell, Camenisch & Chen [BCC04], and a historical account of its development is presented in [BCC05]. A Direct Anonymous Attestation scheme allows remote authentication of trusted platforms, and comprises of five algorithms, each of which will now be discussed.

Setup. The setup algorithm is primarily used by the issuer to construct a public key pair $sk_I, \mathsf{pk}(sk_I)$, and the public part $\mathsf{pk}(sk_I)$ is published. In addition, the setup algorithm may define implementation specific parameters.

Join. The join algorithm is run between a trusted platform and an issuer for the purpose of obtaining group membership. The algorithm assumes that the trusted platform and issuer have established a one-way authenticated channel, that is, the issuer is assured to be communicating with a host and TPM. The definition of DAA does not mandate a particular authentication mechanism, although the Trusted Computing Group recommend encrypting every message sent by the issuer under the TPM's endorsement key [TCG07]. (Although, as demonstrated by the RSA-based DAA scheme, lighter solutions are possible.) On successful completion of the join algorithm, the issuer grants the trusted platform with an attestation identity credential `cre` based upon a secret `tsk` known only by the TPM.

Sign. The sign algorithm is executed by a trusted platform to produce a signature σ, based upon an attestation identity credential `cre` and secret `tsk`, which asserts group membership and therefore trusted platform status. The algorithm takes as input a message m and a basename `bsn` (which is used to control linkability between signatures). If $\texttt{bsn} = \perp$, then signatures should be unlinkable; otherwise, signatures produced by the same signer and based upon the same basename can be linked.

Verify. The verification algorithm is used by a verifier to check the validity of a signature. The algorithm takes as input a set of secret keys $\texttt{ROGUE}_{\texttt{tsk}}$, which are known to have been successfully extracted from compromised TPMs (see Tarnovsky [Tar10] for further details on key extraction), allowing the identification of rogue platforms. The methodology used to build $\texttt{ROGUE}_{\texttt{tsk}}$ is not defined by DAA.

Link. The link algorithm is used by a verifier to check if two valid signatures σ, σ' are linked, that is, signed using the same basename `bsn` and secret `tsk`.

The inputs and outputs of these algorithms are explicitly summarised in Table 4.1.

Linkability in DAA protocols. The ability to link signatures, without revealing the identity of the signer, is a particularly interesting aspect of DAA schemes and the degrees of linkability are identified below, with reference to an application domain in which several verifiers offer multiple services and signers must consent to linkability.

1. *Single-service linkability.* A verifier offering a single service is able to link multiple transactions with the same signer.

2. *Cross-service linkability.* A verifier offering multiple services which share the same basename is able to link transactions with the same signer over multiple services.

3. *Cross-verifier linkability.* Multiple verifiers offering services which share the same basename are able to link transactions with the same signer across all services.

Algorithm	Input	Output
Setup	Security parameters.	A public key pair $sk_I, \mathsf{pk}(sk_I)$ and implementation specific parameters.
Join	Trusted platform inputs: the system parameters (including the issuer's public key and any implementation specific parameters defined by the setup algorithm), the TPM's internal secret `DAASeed` (this value is defined during manufacture [TCG07]), a counter value `cnt` selected by the host, and the TPM's endorsement key. Issuer's inputs: the system parameters.	Trusted platform outputs: a pair consisting of the attestation identity credential `cre` and secret `tsk`. Issuer's outputs: the public part of the TPM's endorsement key.
Sign	The system parameters, a verifier's basename `bsn`, a message m, an attestation identity credential `cre`, and a secret `tsk`.	A signature σ.
Verify	The system parameters, a verifier's basename `bsn`, a message m, a candidate signature σ for m, and a set of secret keys $\mathtt{ROGUE}_{\mathtt{tsk}}$.	1 (accept) or 0 (reject).
Link	The system parameters, and two candidate signatures σ and σ' for messages m and m'.	$\perp$ (invalid signature) if the verify algorithm outputs 0 for signature σ (respectively σ') using the system parameters, basename $\perp$, the message m (respectively m'), and the empty set of secret keys. Otherwise, the algorithm returns 1 if the signatures can be linked and 0 if the signatures cannot be linked.

Table 4.1: Summary of inputs and outputs for Direct Anonymous Attestation algorithms

The classifications of linkability all assume the existence of a single issuer. *Cross issuer linkability*, that is, linkability between signatures produced in different groups (under different issuers), is not expressly forbidden by Direct Anonymous Attestation; however, the schemes considered in this thesis do not provide this property because the secret `tsk` is partly derived from the issuer's public key. Moreover, the game-based security definition [BCL08b, BCL09] does not consider this case.

4.2.1 DAA process specification

This chapter considers user-controlled anonymity, which is dependent on a trusted platform's behaviour, that is, the join and sign algorithms. Formally, these algorithms are captured by a *Direct Anonymous Attestation process specification* (Definition 4.5). Since our focus is on automated reasoning, we will adopt the calculus of ProVerif (Section 4.1).

Definition 4.5 (Direct Anonymous Attestation process specification)**.** *A* Direct Anonymous Attestation process specification *is a tuple of processes* $\langle \mathsf{Join}, \mathsf{Sign} \rangle$.

The signer (or trusted platform) is able to execute arbitrarily many instances of the join, and sign, algorithms, to become a member of a group and subsequently produce signatures as a group member. This behaviour is captured by the Signer process modelled below. The join and sign algorithms are modelled by the processes Join and Sign, which are expected to behave like services; that is, they can be called by, and return results to, the Signer process. The communication between the Signer and Join/Sign processes is achieved using private communication over channels a_j, a'_j, a_s, a'_s. In essence, the private channel communication models the internal bus used by computer systems for communication between the host and TPM.

This chapter focuses on user-controlled anonymity and hence it is sufficient to assume the processes Join and Sign are initiated by input on channels a_j and a_s; and similarly, output results on channels a'_j and a'_s. Intuitively, it follows that some processes not satisfying these conditions will satisfy our definition of user-controlled anonymity. In fact,

the Direct Anonymous Attestation process specification $\langle 0, 0\rangle$ will satisfy our definition. We tolerate this limitations here, and in future work we will consider a complete definition of the DAA properties, including correctness and user-controlled traceability. The correctness property will exclude degenerate process specifications such as $\langle 0, 0\rangle$. Similar considerations are made in the literature, for example, in definitions of vote privacy for electronic voting [KR05, DKR06, DKR09, DKR10b] and privacy for vehicular ad-hoc networks [DDS10].

$$
\begin{aligned}
\mathsf{Signer} \;=\; & \nu\, a_j.\nu\, a'_j.\nu\, a_s.\nu\, a'_s\ .\ (\,(!\mathsf{Join}) \mid (!\mathsf{Sign}) \mid (\nu\, \mathtt{cnt}.\nu\, \mathtt{DAASeed}.\nu\, sk_M.\overline{c}\langle \mathsf{pk}(sk_M)\rangle. \\
& \quad !c(w_{\mathsf{params}}).\overline{a_j}\langle (\!| w_{\mathsf{params}}, \mathtt{DAASeed}, \mathtt{cnt}, sk_M |\!)\rangle.a'_j(x). \\
& \quad \text{let } x_{\mathsf{cre}} = \pi_1(x) \text{ in let } x_{\mathtt{tsk}} = \pi_2(x) \text{ in } (\\
& \qquad !c(y).\text{let } y_{\mathsf{bsn}} = \pi_1(y) \text{ in let } y_{\mathsf{msg}} = \pi_2(y) \text{ in} \\
& \qquad \overline{a_s}\langle (\!| w_{\mathsf{params}}, y_{\mathsf{bsn}}, y_{\mathsf{msg}}, x_{\mathsf{cre}}, x_{\mathtt{tsk}} |\!)\rangle.a'_s(z).\overline{c}\langle z\rangle \\
& \quad) \\
&))
\end{aligned}
$$

The process Signer instantiates arbitrarily many instances of the Join and Sign processes. The restricted channel names a_j, a'_j are introduced to ensure communication between the Signer and Join processes is private; similarly, names a_s, a'_s ensure private communication between the Signer and Sign processes. The bound name cnt is a counter value selected by the host. The bound name DAASeed represents the TPM's internal secret and sk_M represents the TPM's endorsement key (these values are defined during manufacture [TCG07]). The public part of the endorsement key is published by the Signer process. The remainder of the Signer process models a signer's ability to execute arbitrarily many instances of the join and sign algorithms. The Signer process must first input system parameters w_{params}, provided by the issuer. The Join process is assumed to act like a service and listens for input on channel a_j. It follows, that the Signer process can invoke the service by message output $\overline{a_j}\langle (\!| w_{\mathsf{params}}, \mathtt{DAASeed}, \mathtt{cnt}, w_{\mathsf{ek}} |\!)\rangle$, where $(\!| w_{\mathsf{params}}, \mathtt{DAASeed}, \mathtt{cnt}, w_{\mathsf{ek}} |\!)$ models the join

algorithm's parameters. The Join process is assumed to output results on channel a'_j, and this response can be received by the Signer process using message input $a'_j(x)$; the result is bound to the variable x, and is expected to consist of a pair $(\!|x_{\mathsf{cre}}, x_{\mathtt{tsk}}|\!)$ representing the attestation identity credential and TPM's secret. The interaction between the Sign and Signer processes is similar. The Signer process first inputs a variable y which is expected to be a pair representing the verifier's basename y_{bsn} and a message y_{msg}. The invocation of the sign algorithm by the signer is modelled by the message output $\overline{a_s}\langle(\!|w_{\mathsf{params}}, y_{\mathsf{bsn}}, y_{\mathsf{msg}}, x_{\mathsf{cre}}, x_{\mathtt{tsk}}|\!)\rangle$, where $(\!|w_{\mathsf{params}}, y_{\mathsf{bsn}}, y_{\mathsf{msg}}, x_{\mathsf{cre}}, x_{\mathtt{tsk}}|\!)$ represents the algorithm's parameters. The sign algorithm is expected to output a signature which can be sent to a verifier, in the Signer process this signature is received from the Sign process by message input $a'_s(z)$ and the variable z representing the signature is immediately output.

4.3 Security definition: User-controlled anonymity

Informally, the notion of user-controlled anonymity asserts that given two honest signers $\mathcal{A}$ and $\mathcal{B}$, an adversary cannot distinguish between a situation in which $\mathcal{A}$ signs a message, from another one in which $\mathcal{B}$ signs a message. Based upon Brickell, Chen & Li [BCL08b, BCL09] we present the following security definition.

Initial: The adversary constructs the public key pair $sk_I, \mathsf{pk}(sk_I)$ and publishes the public part $\mathsf{pk}(sk_I)$ for the signers. The adversary also publishes any additional parameters.

Phase 1: The adversary makes the following requests to signers $\mathcal{A}$ and $\mathcal{B}$:

- Join. The signer executes the join algorithm to create `cre` and `tsk`. The adversary, as the issuer, learns `cre` but typically not `tsk`.
- Sign. The adversary submits a basename `bsn` and a message m. The signer runs the sign algorithm and returns the signature to the adversary.

At the end of phase 1, both signers are required to have run the join algorithm at least once.

Phase 2 (Challenge): The adversary submits a message m and a basename $\mathtt{bsn}$ to the signers, with the restriction that the basename has not been previously used if $\mathtt{bsn} \neq \perp$. Each signer produces a signature on the message and returns it to the adversary.

Phase 3: The adversary continues to probe the signers with join and sign requests, but is explicitly forbidden to use the basename used in phase 2 if $\mathtt{bsn} \neq \perp$.

Result: The protocol satisfies user-controlled anonymity if the adversary cannot distinguish between the two signatures output during the challenge.

Formally, this definition can be modelled as an observational equivalence (Definition 4.6), which is suitable for automated reasoning, based upon the *augmented Direct Anonymous Attestation biprocess* DAA expressed in Figure 4.4. As observed by Rudolph [Rud07], and as specified by Brickell, Chen & Li [BCL08b, BCL09], user-controlled anonymity can only be expected if both signers use $\mathsf{pk}(sk_I)$ for the issuer (that is, the signers do not accept two distinct keys from the issuer). This is captured by the biprocess DAA by providing the same parameters w_{params} to both signers. The Challenge process is designed to capture the behaviour of the signers in phase 2. This is achieved by outputting an attestation identity credential x_{cre} and a secret $x_{\mathtt{tsk}}$, produced by the signers in phase 1, on the private channels b_A, b_B in Signer^+, and inputting these values in the Challenge process. The Challenge process then continues by producing a signature in the standard manner, but uses $\mathrm{diff}[x_{\mathsf{cre}}, y_{\mathsf{cre}}]$ and $\mathrm{diff}[x_{\mathtt{tsk}}, y_{\mathsf{tsk}}]$ to ensure that the signature is produced by $\mathcal{A}$ on the left hand side and $\mathcal{B}$ on the right hand side. Finally, the necessity for a distinct basename in phase 2 (when $\mathtt{bsn} \neq \perp$) is enforced by prefixing the basename used by Challenge with chl^- and, similarly, prefixing the basenames used by Signer^+ with chl^+. The definition of user-controlled anonymity follows naturally.

Figure 4.4 Biprocess modelling user-controlled anonymity in DAA

Given a Direct Anonymous Attestation process specification $\langle \mathsf{Join}, \mathsf{Sign} \rangle$, the *augmented Direct Anonymous Attestation biprocess* DAA is defined as

$$\nu\, b_A.\nu\, b_B.c(w_{\mathsf{params}})\ .\ (\mathsf{Challenge} \mid \mathsf{Signer}^{+}\{^{b_A}/_{w_b}\} \mid \mathsf{Signer}^{+}\{^{b_B}/_{w_b}\})$$

such that $b_A, b_B \notin (\mathrm{fn}(\mathsf{Sign}) \cup \mathrm{fv}(\mathsf{Sign}) \cup \mathrm{fn}(\mathsf{Join}) \cup \mathrm{fv}(\mathsf{Join}))$ and where

$$\begin{aligned}
\mathsf{Signer}^{+} \quad = \quad & \nu\, a_j.\nu\, a'_j.\nu\, a_s.\nu\, a'_s.(\ (!\mathsf{Join}) \mid (!\mathsf{Sign}) \mid (\nu\ \mathtt{cnt}.\nu\ \mathtt{DAASeed}.\nu\ sk_M.\overline{c}\langle \mathsf{pk}(sk_M)\rangle \\
& !\overline{a_j}\langle (\!| w_{\mathsf{params}}, \mathtt{DAASeed}, \mathtt{cnt}, sk_M |\!)\rangle.a'_j(x). \\
& \text{let } x_{\mathsf{cre}} = \pi_1(x) \text{ in let } x_{\mathtt{tsk}} = \pi_2(x) \text{ in } (\\
& \quad !c(y).\text{let } y_{\mathsf{bsn}} = \pi_1(y) \text{ in let } y_{\mathsf{msg}} = \pi_2(y) \text{ in} \\
& \quad \text{if } y_{\mathsf{bsn}} = \perp \text{ then} \\
& \quad\quad \overline{a_s}\langle (\!| w_{\mathsf{params}}, y_{\mathsf{bsn}}, y_{\mathsf{msg}}, x_{\mathsf{cre}}, x_{\mathtt{tsk}} |\!)\rangle.a'_s(z).\overline{c}\langle z\rangle \\
& \quad \text{else} \\
& \quad\quad \overline{a_s}\langle (\!| w_{\mathsf{params}}, (\!| chl^{+}, y_{\mathsf{bsn}} |\!), y_{\mathsf{msg}}, x_{\mathsf{cre}}, x_{\mathtt{tsk}} |\!)\rangle.a'_s(z).\overline{c}\langle z\rangle \\
&) \mid (\\
& \quad \overline{w_b}\langle (\!| x_{\mathsf{cre}}, x_{\mathtt{tsk}} |\!)\rangle \\
&) \\
&))
\end{aligned}$$

$$\begin{aligned}
\mathsf{Challenge} \quad = \quad & \nu\, a_s.\nu\, a'_s\ .\ (\ (\mathsf{Sign}) \mid (\\
& b_A(x).\text{let } x_{\mathsf{cre}} = \pi_1(x) \text{ in let } x_{\mathtt{tsk}} = \pi_2(x) \text{ in} \\
& b_B(y).\text{let } y_{\mathsf{cre}} = \pi_1(y) \text{ in let } y_{\mathsf{tsk}} = \pi_2(y) \text{ in} \\
& c(z).\text{let } z_{\mathsf{bsn}} = \pi_1(z) \text{ in let } z_{\mathsf{msg}} = \pi_2(z) \text{ in} \\
& \text{if } z_{\mathsf{bsn}} = \perp \text{ then} \\
& \quad \overline{a_s}\langle (\!| w_{\mathsf{params}}, z_{\mathsf{bsn}}, z_{\mathsf{msg}}, \mathrm{diff}[x_{\mathsf{cre}}, y_{\mathsf{cre}}], \mathrm{diff}[x_{\mathtt{tsk}}, y_{\mathsf{tsk}}] |\!)\rangle.a'_s(z).\overline{c}\langle z\rangle \\
& \text{else} \\
& \quad \overline{a_s}\langle (\!| w_{\mathsf{params}}, (\!| chl^{-}, z_{\mathsf{bsn}} |\!), z_{\mathsf{msg}}, \mathrm{diff}[x_{\mathsf{cre}}, y_{\mathsf{cre}}], \mathrm{diff}[x_{\mathtt{tsk}}, y_{\mathsf{tsk}}] |\!)\rangle.a'_s(z).\overline{c}\langle z\rangle \\
&))
\end{aligned}$$

for some constants chl^{+}, chl^{-}.

Definition 4.6 (User-controlled anonymity)**.** *Given a Direct Anonymous Attestation process specification* $\langle \mathsf{Join}, \mathsf{Sign} \rangle$*,* user-controlled anonymity *is satisfied if the augmented Direct Anonymous Attestation biprocess* DAA *satisfies observational equivalence.*

The definition has been used to analyse user-controlled anonymity in RSA-based DAA.

4.4 Case study: RSA-based DAA

The first concrete Direct Anonymous Attestation scheme was introduced by Brickell, Camenisch & Chen [BCC04] and is based on RSA.

4.4.1 Primitives and building blocks

We first recall the details of Camenisch-Lysyanskaya (CL) signatures [CL03, Lys02], which form the foundations of RSA-based DAA, and introduce some notational conventions.

Signature scheme. A CL signature is denoted $\mathsf{clsign}(x_{\mathsf{sk}}, x_{\mathsf{prime}}, x_{\mathsf{rand}}, x_{\mathsf{msg}})$, where x_{sk} is the secret key, x_{prime} is a random prime, x_{rand} is a nonce, and x_{msg} is a message. The prime and nonce components can be derived from a signature. Verification is standard given a signature, message, and public key, that is, $\mathsf{checkclsign}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{msg}}, \mathsf{clsign}(x_{\mathsf{sk}}, x_{\mathsf{prime}}, x_{\mathsf{rand}}, x_{\mathsf{msg}})) = \mathsf{accept}$.

Signature scheme for committed values. The scheme supports signatures on committed values. Given the public part of a signing key $\mathsf{pk}(x_{\mathsf{sk}})$, a message x_{csk}, and commitment factor x_{cf}, the committed value is $U = \mathsf{clcommit}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{cf}}, x_{\mathsf{csk}})$ and the associated signature is $\mathsf{clsign}(x_{\mathsf{sk}}, y_{\mathsf{prime}}, y_{\mathsf{rand}}, U)$ for some prime y_{prime} and random y_{rand}. This signature can be opened to recover $\sigma = \mathsf{clopen}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{cf}}, \mathsf{clsign}(x_{\mathsf{sk}}, y_{\mathsf{prime}}, y_{\mathsf{rand}}, U)) = \mathsf{clsign}(x_{\mathsf{sk}}, y_{\mathsf{prime}}, y_{\mathsf{rand}} \circ x_{\mathsf{cf}}, x_{\mathsf{csk}})$, that is, the signature on x_{csk}. (A proof should also be provided to demonstrate that σ does not contain a covert channel – such details will be omitted from the model presented here – see Appendix B for further details.)

Notation for primitives that prove knowledge. Various primitives which prove knowledge of, and relations among, discrete logarithms are used by CL signatures and RSA-based DAA. These primitives will be described using the notation introduced by Camenisch & Stadler [CS97a]. For instance,

$$PK\{(\alpha, \beta) : N = \mathsf{commit}(\alpha, Z) \wedge U = \mathsf{clcommit}(\mathsf{pk}(sk_I), \alpha, \beta)\}$$

denotes a *"zero-knowledge* P*roof of* K*nowledge of* α, β *such that* $N = \mathsf{commit}(\alpha, Z)$ *and* $U = \mathsf{clcommit}(\mathsf{pk}(sk_I), \alpha, \beta)$ *holds."* In the example, the Greek letters in parentheses are used for values about which knowledge is being proved and these values are kept secret by the prover. All other values, that is, those from the Latin alphabet, are known to the verifier. The Fiat-Shamir heuristic [FS87, PS96] allows an interactive zero-knowledge scheme to be converted into a signature scheme. A signature acquired in this way is termed a S*ignature* P*roof of* K*nowledge* and is denoted, for example, as $SPK\{(\alpha) : N = \mathsf{commit}(\alpha, Z)\}(m)$, where m is a message.

Proving knowledge of a signature. The signature scheme for committed values can be used to build an anonymous credential system. Given a signature $\sigma = \mathsf{clsign}(x_{\mathsf{sk}}, x_{\mathsf{prime}}, x_{\mathsf{rand}}, x_{\mathsf{csk}})$ and commitment factor x_{cf}, an anonymous credential $\hat{\sigma} = \mathsf{clcommit}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{cf}}, \sigma)$. The zero-knowledge proof of knowledge

$$PK\{(x_{\mathsf{csk}}, x_{\mathsf{cf}}) : \mathsf{checkclsign}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{csk}}, \mathsf{clopen}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{cf}}, \hat{\sigma})) = \mathsf{accept}\}$$

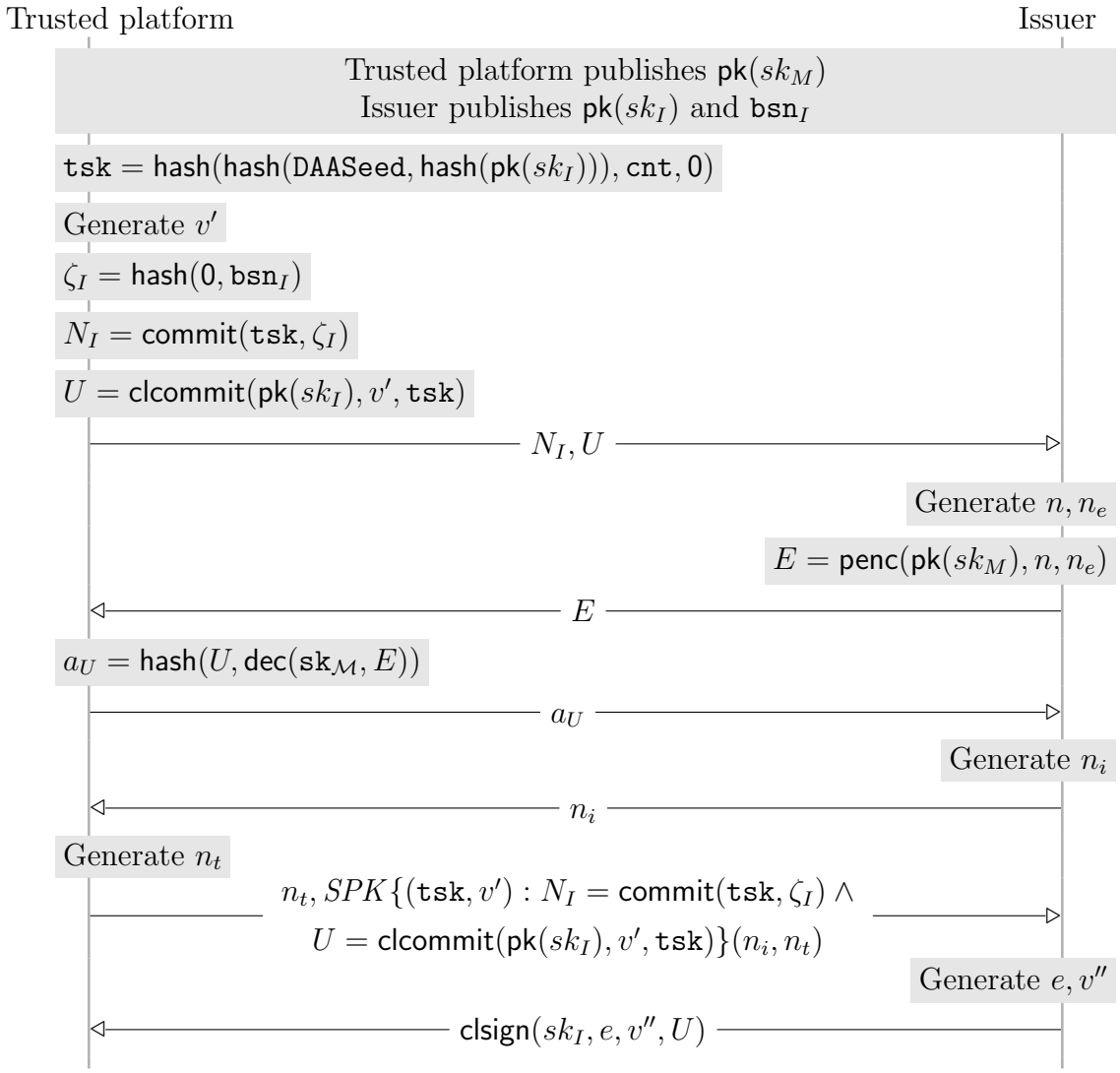
can then be used to demonstrate that the anonymous credential $\hat{\sigma}$ is indeed a commitment to a signature on the message x_{csk} using commitment factor x_{cf}.

The application of these primitives to construct the RSA-based DAA protocol will be considered in the next section.

4.4.2 Protocol description

For the purpose of studying user-controlled anonymity, it is sufficient to consider the join and sign algorithms. Given system parameters $\mathsf{pk}(sk_I)$, $\mathtt{bsn}_I$ (that is, the issuer's public key and basename), the TPM's secret `DAASeed`, a counter value `cnt`, and the TPM's endorsement key; the join algorithm (Figure 4.5) proceeds as follows:

Figure 4.5 RSA-based DAA join algorithm



1. The host computes $\zeta_I = \mathsf{hash}(0, \mathtt{bsn}_I)$ and inputs the value to the TPM. The TPM computes secret $\mathtt{tsk} = \mathsf{hash}(\mathsf{hash}(\mathtt{DAASeed}, \mathsf{hash}(\mathsf{pk}(sk_I))), \mathtt{cnt}, 0)$ and derives the commitment $N_I = \mathsf{commit}(\mathtt{tsk}, \zeta_I)$. The TPM also generates a blinding factor v',

which is used to compute the commitment $U = \mathsf{clcommit}(\mathsf{pk}(sk_I), v', \mathtt{tsk})$. The trusted platform sends U, N_I to the issuer.

2. The issuer generates a nonce n_e and sends the encrypted nonce to the TPM. The TPM decrypts the ciphertext to recover n_e, computes $a_U = \mathsf{hash}(U, n_e)$ and sends a_U to the issuer, therefore authenticating as a trusted platform. (Note that the protocol does not rely on the authentication technique recommended by the Trusted Computing Group.)

3. The trusted platform generates a signature proof of knowledge that the messages U, N_I are correctly formed and sends it to the issuer.

4. The issuer verifies the proof and generates a signature $\mathsf{clsign}(sk_I, e, v'', U)$. The signature is sent to the trusted platform.

5. The trusted platform verifies the signature and opens it to reveal the credential $\mathtt{cre} = \mathsf{clsign}(sk_I, e, v' \circ v'', \mathtt{tsk})$, that is, the TPM's secret $\mathtt{tsk}$ signed by the issuer.

The join algorithm outputs $\mathtt{cre}, \mathtt{tsk}$; which can be provided as input, along with the system parameters, a basename $\mathtt{bsn}$, and message m, to the sign algorithm. The sign algorithm proceeds as follows.

5. If $\mathtt{bsn} = \perp$, the host generates a nonce ζ; otherwise, the host computes $\zeta = \mathsf{hash}(0, bsn)$. The host provides the TPM with ζ. The TPM computes the commitment $N_V = \mathsf{commit}(\mathtt{tsk}, \zeta)$ and generates the anonymous credential $\widehat{\mathtt{cre}} = \mathsf{clcommit}(\mathsf{pk}(sk_I), w, \mathtt{cre})$ using a nonce w. The trusted platform then produces a signature proof of knowledge that $\widehat{\mathtt{cre}}$ is a commitment to a valid credential, and that N_V is correctly formed.

The sign algorithm outputs the signature proof of knowledge which is sent to the verifier. Intuitively, if a verifier is presented with such a proof, then the verifier is convinced that it is communicating with a trusted platform.

4.4.3 Equational theory

The signature is defined below

$$\Sigma = \{\mathsf{accept}, \bot, 0, 1, \mathsf{F_{join}}, \mathsf{F_{sign}}, \mathsf{clgetnonce}, \mathsf{clgetprime}, \mathsf{hash}, \mathsf{pk}, \mathsf{commit}, \circ, \mathsf{dec}, \mathsf{open}, \mathsf{checkclsign}, \mathsf{checkspk}, \mathsf{clcommit}, \mathsf{clopen}, \mathsf{penc}, \mathsf{spk}, \mathsf{clsign}\}$$

Functions accept, $\bot$, 0, 1, $\mathsf{F_{join}}$, $\mathsf{F_{sign}}$ are constant symbols; clgetnonce, clgetprime, hash, pk are unary functions; commit, $\circ$, dec, open are binary functions; checkclsign, checkspk, clcommit, clopen, penc, spk are ternary functions; and clsign is a function of arity four. Since hash is defined as a unary function, we occasionally write $\mathsf{hash}(x_{\mathsf{plain},1}, \ldots, x_{\mathsf{plain,n}})$ to denote $\mathsf{hash}((\!|x_{\mathsf{plain},1}, \ldots, x_{\mathsf{plain,n}}|\!))$. The equations associated with these functions are defined below.

$$\mathsf{dec}(x_{\mathsf{sk}}, \mathsf{penc}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{plain}})) = x_{\mathsf{plain}}$$

$$\mathsf{clgetprime}(\mathsf{clsign}(x_{\mathsf{sk}}, x_{\mathsf{prime}}, x_{\mathsf{rand}}, x_{\mathsf{msg}})) = x_{\mathsf{prime}}$$

$$\mathsf{clgetnonce}(\mathsf{clsign}(x_{\mathsf{sk}}, x_{\mathsf{prime}}, x_{\mathsf{rand}}, x_{\mathsf{msg}})) = x_{\mathsf{rand}}$$

$$\mathsf{checkclsign}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{msg}}, \mathsf{clsign}(x_{\mathsf{sk}}, x_{\mathsf{prime}}, x_{\mathsf{rand}}, x_{\mathsf{msg}})) = \mathsf{accept}$$

$$\mathsf{open}(x_{\mathsf{rand}}, \mathsf{commit}(x_{\mathsf{rand}}, x_{\mathsf{plain}})) = x_{\mathsf{plain}}$$

$$\mathsf{clopen}(x, x_{\mathsf{rand}}, \mathsf{clcommit}(x, x_{\mathsf{rand}}, x_{\mathsf{plain}})) = x_{\mathsf{plain}}$$

$$\mathsf{clopen}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, \mathsf{clsign}(x_{\mathsf{sk}}, y_{\mathsf{prime}}, y_{\mathsf{rand}}, \mathsf{clcommit}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{rand}}, x_{\mathsf{msg}}))) = \mathsf{clsign}(x_{\mathsf{sk}}, y_{\mathsf{prime}}, y_{\mathsf{rand}} \circ x_{\mathsf{rand}}, x_{\mathsf{msg}})$$

A signature proof of knowledge is encoded in the form $\mathsf{spk}(F, U, V)$, where F is a constant declaring the particular proof in use, U denotes the witness (or private component) of a signature of knowledge, and V defines the public parameters and message being signed. The function checkspk is used to verify a signature and we define the following equations.

$$\mathsf{checkspk}(\mathsf{F}_{\mathsf{join}}, V, \mathsf{spk}(\mathsf{F}_{\mathsf{join}}, (x_{\mathtt{tsk}}, x_{\mathsf{cf}}), V)) = \mathsf{accept}$$
$$\text{where } V = (x_{\zeta_{\mathsf{I}}}, x_{\mathsf{pk}}, \mathsf{commit}(x_{\mathtt{tsk}}, x_{\zeta_{\mathsf{I}}}), \mathsf{clcommit}(x_{\mathsf{pk}}, x_{\mathsf{cf}}, x_{\mathtt{tsk}}), x_{\mathsf{msg}})$$

$$\mathsf{checkspk}(\mathsf{F}_{\mathsf{sign}}, V, \mathsf{spk}(\mathsf{F}_{\mathsf{sign}}, (x_{\mathtt{tsk}}, x_{\mathsf{cf}}), V)) = \mathsf{accept}$$
$$\text{where } V = (x_{\zeta}, \mathsf{pk}(x_{\mathsf{sk}}), \mathsf{commit}(x_{\mathtt{tsk}}, x_{\zeta}),$$
$$\mathsf{clcommit}(\mathsf{pk}(x_{\mathsf{sk}}), x_{\mathsf{cf}}, \mathsf{clsign}(x_{\mathsf{sk}}, x_{\mathsf{prime}}, x_{\mathsf{rand}}, x_{\mathtt{tsk}})), x_{\mathsf{msg}})$$

The first equation is used to verify the signature proof of knowledge produced by the trusted platform during the join algorithm and the second is used by a trusted platform during the sign algorithm to assert group membership.

4.4.4 Model in applied pi

The RSA-based DAA process specification is presented in Definition 4.7. For convenience we abbreviate $c(x).\text{let } x_1 = \pi_1(x) \text{ in } \ldots \text{ let } x_n = \pi_n(x) \text{ in } P \text{ as } c(x_1, \ldots, x_n).P$. The join process $\mathsf{Join}_{\mathsf{RSA}}$ is instantiated by inputting the join algorithm's parameters: the RSA-based DAA system parameters w_{params}; the TPM's internal secret w_{DAASeed}; the counter value w_{cnt} chosen by the host; and the TPM's endorsement key w_{ek}. The system parameters w_{params} are expected to be a pair containing the issuer's public key w_{pk} and basename $w_{\mathsf{bsn_I}}$. The process constructs the terms N_I, U in accordance with the protocol's description (Section 4.4.2) and outputs the values to the issuer. The process then receives a ciphertext x, which it decrypts, and outputs the hash of the plaintext paired with U. A nonce y is then input and a signature proof of knowledge is produced. Finally, the process inputs a signature z on the commitment U and concludes by outputting the attestation identity credential `cre` and TPM's secret `tsk` on the private channel a'_j; that is, the $\mathsf{Join}_{\mathsf{RSA}}$ process returns the values `cre`, `tsk` to the Signer^{+} process. The sign process $\mathsf{Sign}_{\mathsf{RSA}}$ is instantiated by inputting the sign algorithm's parameters: the RSA-based DAA system parameters w_{params}; the verifier's basename w_{bsn}; the message w_{msg} to be signed; the attestation identity credential w_{cre}; and the TPM's secret w_{tsk}. The process recovers the issuer's public key w_{pk} from the system parameters, and inputs a

Definition 4.7. *The DAA process specification* $\langle \mathsf{Join}_{\mathsf{RSA}}, \mathsf{Sign}_{\mathsf{RSA}} \rangle$ *is defined where*

$$
\begin{aligned}
\mathsf{Join}_{\mathsf{RSA}} \;\hat{=}\;\; & a_j(w_{\mathsf{params}}, w_{\mathsf{DAASeed}}, w_{\mathsf{cnt}}, w_{\mathsf{ek}}) \,.\, \nu\, v' \,. \\
& \text{let } w_{\mathsf{pk}} = \pi_1(w_{\mathsf{params}}) \text{ in let } w_{\mathsf{bsn_I}} = \pi_2(w_{\mathsf{params}}) \text{ in} \\
& \text{let } \zeta_I = \mathsf{hash}(0, w_{\mathsf{bsn_I}}) \text{ in} \\
& \text{let } \boldsymbol{tsk} = \mathsf{hash}(\mathsf{hash}(w_{\mathsf{DAASeed}}, \mathsf{hash}(w_{\mathsf{pk}})), w_{\mathsf{cnt}}, 0) \text{ in} \\
& \text{let } N_I = \mathsf{commit}(\boldsymbol{tsk}, \zeta_I) \text{ in} \\
& \text{let } U = \mathsf{clcommit}(w_{\mathsf{pk}}, v', \boldsymbol{tsk}) \text{ in} \\
& \overline{c}\langle (\!| N_I, U |\!) \rangle \,.\, c(x) \,.\, \overline{c}\langle \mathsf{hash}(U, \mathsf{dec}(w_{\mathsf{ek}}, x)) \rangle \,.\, c(y) \,.\, \nu\, n_t \,. \\
& \overline{c}\langle (\!| n_t, \mathsf{spk}(\mathsf{F}_{\mathsf{join}}, (\!| \boldsymbol{tsk}, v' |\!), (\!| \zeta_I, w_{\mathsf{pk}}, N_I, U, (\!| n_t, y |\!) |\!)) |\!) \rangle \,.\, c(z) \\
& \text{let } \boldsymbol{cre} = \mathsf{clopen}(w_{\mathsf{pk}}, v', z) \text{ in} \\
& \text{if } \mathsf{checkclsign}(w_{\mathsf{pk}}, \boldsymbol{tsk}, \boldsymbol{cre}) = \mathsf{accept} \text{ then} \\
& \overline{a'_j}\langle (\!| \boldsymbol{cre}, \boldsymbol{tsk} |\!) \rangle
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Sign}_{\mathsf{RSA}} \;\hat{=}\;\; & a_s(w_{\mathsf{params}}, w_{\mathsf{bsn}}, w_{\mathsf{msg}}, w_{\mathsf{cre}}, w_{\mathsf{tsk}}) \,.\, \text{let } w_{\mathsf{pk}} = \pi_1(w_{\mathsf{params}}) \text{ in} \\
& c(x) \,.\, \nu\, n_t \,.\, \nu\, w \,. \\
& \text{if } w_{\mathsf{bsn}} = \perp \text{ then} \\
& \quad \nu\, \zeta \,. \\
& \quad \text{let } \widehat{\boldsymbol{cre}} = \mathsf{clcommit}(w_{\mathsf{pk}}, w, w_{\mathsf{cre}}) \text{ in} \\
& \quad \text{let } N_V = \mathsf{commit}(w_{\mathsf{tsk}}, \zeta) \text{ in} \\
& \quad \text{let } spk = \mathsf{spk}(\mathsf{F}_{\mathsf{sign}}, (\!| w_{\mathsf{tsk}}, w |\!), (\!| \zeta, w_{\mathsf{pk}}, N_V, \widehat{\boldsymbol{cre}}, (\!| n_t, x, w_{\mathsf{msg}} |\!) |\!)) \text{ in} \\
& \quad \overline{a'_s}\langle (\!| \zeta, w_{\mathsf{pk}}, N_V, \widehat{\boldsymbol{cre}}, n_t, spk |\!) \rangle \\
& \text{else} \\
& \quad \text{let } \zeta = \mathsf{hash}(0, w_{\mathsf{bsn}}) \text{ in} \\
& \quad \text{let } \widehat{\boldsymbol{cre}} = \mathsf{clcommit}(w_{\mathsf{pk}}, w, w_{\mathsf{cre}}) \text{ in} \\
& \quad \text{let } N_V = \mathsf{commit}(w_{\mathsf{tsk}}, \zeta) \text{ in} \\
& \quad \text{let } spk = \mathsf{spk}(\mathsf{F}_{\mathsf{sign}}, (\!| w_{\mathsf{tsk}}, w |\!), (\!| \zeta, w_{\mathsf{pk}}, N_V, \widehat{\boldsymbol{cre}}, (\!| n_t, x, w_{\mathsf{msg}} |\!) |\!)) \text{ in} \\
& \quad \overline{a'_s}\langle (\!| \zeta, w_{\mathsf{pk}}, N_V, \widehat{\boldsymbol{cre}}, n_t, spk |\!) \rangle
\end{aligned}
$$

nonce x from the verifier. The if-then-else branch models the signer's ability to produce either linkable or unlinkable signatures, based upon the parameter w_{bsn}; in particular, the if-branch produces an unlinkable signature, whereas the else-branch produces a linkable signature. The process concludes by outputting a signature on the private channel a'_s; that is, the $\mathsf{Sign}_{\mathsf{RSA}}$ process returns the signature to the Signer^+ process.

4.4.5 Analysis: Violating user-controlled anonymity

The Direct Anonymous Attestation process specification $\langle \mathsf{Join}_{\mathsf{RSA}}, \mathsf{Sign}_{\mathsf{RSA}} \rangle$ does not satisfy *user-controlled anonymity* (Theorem 4.2). This is due to a vulnerability which can be exploited by the following adversaries to violate privacy.

1. Passive adversary. A passive adversary can violate privacy under the assumptions: a) the identity of a trusted platform can be observed during the join algorithm[1]; b) there exists a basename which is shared between an issuer and a verifier; and c) a signer is willing to accept the same basename from an issuer and verifier. The attack proceeds as follows. We have $\mathtt{bsn}_I = \mathtt{bsn}$, and since the signer is willing to accept the same basename as input to both the sign and join algorithms we have $\zeta_I = \zeta$, hence $N_I = N_V$. Since the N_I, N_V values are unique for a particular signer and the adversary knows the identity of the trusted platform that produced N_I during the join algorithm; it follows that the signer's identity can be revealed.

2. Corrupt administrators. Corrupt administrators can violate privacy under the assumption that a signer is willing to accept the same basename from an issuer and verifier. An issuer and verifier can conspire to use the same basename (that is, $\mathtt{bsn}_I = \mathtt{bsn}$) and since the issuer knows the identity of the trusted platform that produced N_I, the identity of the signer can be revealed.

[1]The RSA-based DAA protocol [BCC04] does not explicitly specify how the issuer learns a trusted platform's public endorsement key during an execution of the join algorithm. However, it seems reasonable to assume that the public key would be sent as plaintext. By contrast, Cesena *et al.* [CLR+10, Ces10] define an extension of RSA-based DAA which uses TLS to hide the affiliation between groups and trusted platforms; this variant would thwart a passive adversary, but not corrupt administrators.

The linchpin of these attacks is the willingness of a signer to accept the same basename from an issuer and verifier. This can be justified as follows. Firstly, this mode of operation is not explicitly forbidden by the protocol definition [BCC04]. Secondly, this behaviour is expected when the issuer and verifier are the same entity, as demonstrated, for example, by Camenisch *et al.* [CL01, CH02] in the *idemix* system. Finally, the signer has insufficient resources to handle such a duty.

Theorem 4.2. *The RSA-based Direct Anonymous Attestation process specification* $\langle \mathsf{Join}_{\mathsf{RSA}}, \mathsf{Sign}_{\mathsf{RSA}} \rangle$ *does not satisfy user-controlled anonymity.*

Formally, Theorem 4.2 can be proved by presenting a context $C[\_]$ such that $\mathrm{fst}(C[\mathsf{DAA}_{\mathsf{RSA}}]) \rightarrow^* Q$ and Q can output on some channel M, but there is no reduction $\mathrm{snd}(C[\mathsf{DAA}_{\mathsf{RSA}}]) \rightarrow^* Q'$ such that Q' can output on M, where both reductions are of the same length and $\mathsf{DAA}_{\mathsf{RSA}}$ is the augmented Direct Anonymous Attestation biprocess derived from $\langle \mathsf{Join}_{\mathsf{RSA}}, \mathsf{Sign}_{\mathsf{RSA}} \rangle$. Such a context is presented in Appendix C.1. In addition to the formal result (Theorem 4.2), a real-world attack is demonstrated against the RSA-based Direct Anonymous Attestation protocol [BCC04] in Appendix B.3.

4.4.6 Solution: Restoring user-controlled anonymity

The protocol can be fixed by refining the definition of ζ; the revised RSA-based Direct Anonymous Attestation process specification $\langle \mathsf{Join}_{\mathsf{RSA}'}, \mathsf{Sign}_{\mathsf{RSA}'} \rangle$ is the same as the original, with ζ redefined as $\zeta = \mathsf{hash}(1, \mathtt{bsn})$. The attacks presented are no longer possible, regardless of whether $\mathtt{bsn}_I = \mathtt{bsn}$. Furthermore, the revised RSA-based Direct Anonymous Attestation process specification $\langle \mathsf{Join}_{\mathsf{RSA}'}, \mathsf{Sign}_{\mathsf{RSA}'} \rangle$ satisfies user-controlled anonymity; this can be automatically verified using ProVerif (see Appendix C.2). A fix for the concrete scheme [BCC04] is presented in Appendix B.4.

4.5 Summary

This chapter presents a definition of user-controlled anonymity for Direct Anonymous Attestation protocols. The definition is expressed as an equivalence property suitable for automated reasoning. The practicality of the approach is demonstrated by evaluating the RSA-based Direct Anonymous Attestation protocol. This scheme is particularly significant because support is mandated by the TPM specification version 1.2, which has been implemented and deployed in over 300 million computers. The analysis discovers a vulnerability which can be exploited by a passive adversary and, under weaker assumptions, by corrupt administrators. A security fix is identified and the revised protocol is shown to satisfy user-controlled anonymity. The fix only affects the host's part of the protocol and therefore no hardware changes to the TPM are required. Furthermore, this research appears to have influenced the design of subsequent Direct Anonymous Attestation schemes, for example, the conceptual design of the join algorithm has been revised in [BCL08a, BCL09].

Part III

Automated reasoning

The lack of formal security analysis prior to system deployment can be partially attributed to the absence of suitable tools. This part develops new procedures for the automated analysis of observational equivalence.

5

Observational equivalence and barriers[†]

Overview. *A procedure for the automated analysis of observational equivalence is delivered. In addition, the notion of barrier synchronisation is characterised for processes. The study of barrier synchronisation in relation to equivalence is particularly interesting because certain equivalence properties can only be realised under specific synchronisation assumptions. In particular, privacy preserving protocols – including electronic voting schemes, vehicular ad-hoc networking protocols, and anonymity networks – make such assumptions. The results have been implemented in the tool* ProSwapper*, an extension to ProVerif, and the applicability of this research is demonstrated by analysing vote privacy in electronic voting protocols and privacy in vehicular ad-hoc networks.*

Blanchet, Abadi & Fournet [Bla04, BAF08] focus on equivalences $P \sim Q$ in which processes P and Q share the same structure and differ only in the choice of terms. Their work introduces the notion of *strong uniformity* (see Section 4.1), a sufficient condition for observational equivalence, that has been implemented in ProVerif, an automatic analysis tool for protocols written in the applied pi calculus. Strong uniformity is a sufficient condition for observational equivalence, but it is not necessary. This precludes the automated analysis of certain equivalence properties, as a simple example will demonstrate.

[†]This chapter is partly based upon [DRS08] which introduces the fundamental notion of swapping at synchronisation points and considered a preliminary analysis of privacy in FOO. The software associated with this chapter is available online: `http://www.bensmyth.com/proswapper.php`.

Example 5.1. *The biprocess* $P = \overline{c}\langle \text{diff}[m, n]\rangle \mid \overline{c}\langle \text{diff}[n, m]\rangle$ *is observationally equivalent because* $\text{fst}(P) \equiv \text{snd}(P)$*. However, the plain evaluation context*

$$C[\_] = \_ \mid c(x).\text{let } y = \mathsf{eq}(x, m) \text{ in } 0 \text{ else } 0$$

is such that $C[P] \to Q$*, where* $Q = \overline{c}\langle \text{diff}[n, m]\rangle \mid \text{let } y = \mathsf{eq}(\text{diff}[m, n], m) \text{ in } 0 \text{ else } 0$ *and* $\text{fst}(Q) \to \overline{c}\langle n\rangle$*; but, there is no biprocess* R *such that* $Q \to R$ *and* $\text{fst}(R) \equiv \overline{c}\langle n\rangle$*, because* $\text{fst}(\mathsf{eq}(\text{diff}[m, n], m)) \Downarrow m$*, but there is no term* M_2 *such that* $\text{snd}(\mathsf{eq}(\text{diff}[m, n], m)) \Downarrow M_2$*. It follows immediately that* P *does not satisfy Definition 4.4, that is, it cannot be shown to satisfy observational equivalence using Theorem 4.1. Similarly, the biprocess* $(\text{if } a = \text{diff}[a, c] \text{ then } \overline{c}\langle c\rangle \text{ else } 0) \mid (\text{if } b = \text{diff}[c, b] \text{ then } \overline{c}\langle c\rangle \text{ else } 0)$ *satisfies observational equivalence but does not satisfy strong uniformity.*

Barrier synchronisation [Bro86, HFM88, AJ89, Lub90] is a concept to coordinate the actions performed by concurrently executing processes. More precisely, the mechanism ensures that a process will block, when a barrier is encountered, until all other processes executing in parallel reach this barrier. Capturing such synchronisation is an important building block for protocol analysis because certain security properties can only be realised if processes synchronise their actions in a specific manner. For example, privacy preserving protocols typically require the existence of at least two honest participants that synchronise prior to performing critical actions. This particular prerequisite can be observed in domains including: electronic voting [DKR09, BHM08], vehicular ad-hoc networks [DDS10], and anonymity networks [RR98, PK01, Cho06].

Resolving the difficulties apparent in Example 5.1 is relatively straightforward: the biprocess $P' = \overline{c}\langle \text{diff}[m, m]\rangle \mid \overline{c}\langle \text{diff}[n, n]\rangle$ can trivially be shown to satisfy observational equivalence using strong uniformity as a proof technique, and since $\text{fst}(P) \equiv \text{fst}(P')$ and $\text{snd}(P) \equiv \text{snd}(P')$, it follows that P satisfies observational equivalence because observational equivalence is closed under structural equivalence. However, this technique cannot be applied to more complicated examples in which barriers occur.

Example 5.2. *Consider the biprocess*

$$P = \overline{c}\langle m\rangle.1::\overline{c}\langle \mathrm{diff}[m,n]\rangle \mid 1::\overline{c}\langle \mathrm{diff}[n,m]\rangle$$

in which $1::$ *represents a barrier synchronisation. Intuitively, there is no structurally equivalent biprocess* P' *satisfying strong uniformity, but nevertheless, barrier synchronisation ensures indistinguishability between* $\mathrm{fst}(P)$ *and* $\mathrm{snd}(P)$*. (This example will be elaborated upon once we formalise barrier synchronisation.)*

This chapter introduces a strictly weaker notion of uniformity which can be used to automatically reason with observational equivalence between processes containing barriers.

Chapter contribution

The contribution of this chapter is threefold. Firstly, the process syntax (Section 4.1) is extended to capture barriers and it is shown how the semantic behaviour of barriers can be encoded in the standard ProVerif model. Although the formalisation of barriers is only an encoding, it considerably simplifies the modelling process and, moreover, forms the foundation for our automated analysis technique.

Secondly, a methodology for automatic analysis of equivalence properties is developed, based upon [Bla04, BAF08]. This approach avoids the limitations of [Bla04, BAF08] by *swapping* data at synchronisation points. For example, in Example 5.2 swapping data between the two sides of the parallel composition is permitted at synchronisation; in essence, it is therefore sufficient to prove the equivalence of the biprocess $P' = \overline{c}\langle m\rangle.1::.\overline{c}\langle \mathrm{diff}[m,m]\rangle \mid 1::\overline{c}\langle \mathrm{diff}[n,n]\rangle$, which is trivially true because $\mathrm{fst}(P') = \mathrm{snd}(P')$.

Finally, automated support is provided by defining a static compiler which encodes barrier synchronisation and swapping using private channel communication. The compiler is designed to compute an approximation of a process's observable behaviour, and hence, if ProVerif can show observational equivalence between a pair of compiled processes, then

the original pair of processes satisfies observational equivalence. The compiler has been implemented as a tool called *ProSwapper*. Furthermore, the analysis of vote privacy in electronic voting protocols and privacy in vehicular ad-hoc networks demonstrate the applicability of this work.

Chapter limitations. Proving soundness of the proof technique is ongoing work. Although some confidence may be provided by the tool's adoption within the research community (for example, [BHM08, DDS10]), this is no substitute for a thorough proof.

Structure of this chapter. Section 5.1 extends the process syntax of Section 4.1 to capture barrier synchronisation and provides mechanical reasoning techniques for such processes. Section 5.2 defines a methodology for automatic analysis of equivalence properties. In Section 5.3 the applicability of this work is demonstrated by presenting an automated analysis of privacy in the electronic voting protocol FOO and in Appendix E privacy in vehicular ad-hoc networks is considered; the technique's adoption by the research community will also be discussed. Finally, a summary is presented in Section 5.4.

5.1 Processes with barriers

A notion of synchronisation, called *stages*, has previously been studied in the applied pi calculus by Blanchet, Abadi & Fournet [BAF08, §8]; however, stages are insufficient for our purposes. In the spirit of [BAF08, §8], we proceed by allowing processes to be annotated with *barriers*. Barriers are written $t{::}$ and may appear before a process P, where $t \in \mathbb{N}$. Intuitively, $t{::}\,P$ blocks P until all processes running in parallel are ready to synchronise at barrier t. For example, the process $1{::}\,\overline{c}\langle k\rangle.2{::}\,\overline{c}\langle m\rangle \mid 2{::}\,\overline{c}\langle n\rangle$ can only output n after synchronisation at barrier $t = 2$. It follows that the process cannot output n without having previously output k; this is in direct contrast with [BAF08, §8], which permits the output of n without having previously observed an output of k. Formally, the number of barriers which must be reached is computed statically, in advance

of execution, and thus branching behaviour may cause blocking. For example, the process $c(x).\text{if } x = k \text{ then } 1::\overline{c}\langle m\rangle \text{ else } \overline{c}\langle n\rangle \mid 1::\overline{c}\langle s\rangle$ may evolve to $\overline{c}\langle n\rangle \mid 1::\overline{c}\langle s\rangle$ which can never output s because synchronisation at barrier $t = 1$ requires two processes to synchronise. Barriers are assumed to be ordered; that is, given $t::P$ the process P also blocks if there are any barriers t' such that $t' < t$. In addition, the occurrence of barriers under replication is explicitly forbidden; this restriction is made for technical convenience, although the loss of generality is minimal, since such a process would deadlock.

5.1.1 Mechanical analysis

Processes containing barriers are not supported by ProVerif. Accordingly, we define a compiler which takes processes with barriers as input and returns processes defined over the standard ProVerif syntax. Although our characterisation of barriers is just an encoding, we find it useful in practice and, moreover, the compiler forms the basis of the methodology for automatic analysis which will be introduced in Section 5.2.

The *barrier elimination function* (Definition 5.1) takes processes containing barriers and replaces each barrier with a message output followed by a message input[1]. By ensuring that the communication channels are fresh private names and through the introduction of a *synchronisation process*, the semantics of barriers can be captured in the standard ProVerif model.

[1]The use of two distinct channel names for the input and output channels is not strictly necessary; however, it aids automation. In particular, it helps avoid a ProVerif incompleteness issue that arises because there is no distinction between messages input and messages output on a particular channel.

Definition 5.1 (Barrier elimination function)**.** *The* barrier elimination function δ *maps an infinite sequence of channel names* a *and a process* P *to another process, the function is recursively defined as follows:*

$$
\begin{array}{lcl}
\delta(\mathsf{c},\ 0) & = & 0 \\
\delta(\mathsf{c},\ !Q) & = & !Q \\
\delta(\mathsf{c},\ \nu\, n.Q) & = & \nu\, n.\delta(\mathsf{c},\ Q) \\
\delta(\mathsf{c},\ M(x).Q) & = & M(x).\delta(\mathsf{c},\ Q) \\
\delta(\mathsf{c},\ \overline{M}\langle N\rangle.Q) & = & \overline{M}\langle N\rangle.\delta(\mathsf{c},\ Q) \\
\delta(\mathsf{c},\ t{::}\, Q) & = & \overline{a'}\langle a'\rangle.a''(y).\delta(\mathsf{tail}(\mathsf{tail}(\mathsf{c})),\ Q) \\
 & & \textit{where } a' = \mathsf{head}(\mathsf{c}),\ a'' = \mathsf{head}(\mathsf{tail}(\mathsf{c})) \textit{ and } y \notin \mathrm{fv}(Q) \\
\delta(\mathsf{c},\ \text{let } x = D \text{ in } Q \text{ else } R) & = & \text{let } x = D \text{ in } \delta(\mathsf{odd}(\mathsf{c}),\ Q) \text{ else } \delta(\mathsf{even}(\mathsf{c}),\ R) \\
\delta(\mathsf{c},\ Q \mid R) & = & \delta(\mathsf{odd}(\mathsf{c}),\ Q) \mid \delta(\mathsf{even}(\mathsf{c}),\ R)
\end{array}
$$

Intuitively, processes which synchronise at a particular barrier all output a message on a private channel and these messages are received by the synchronisation process; once all messages have been sent/received, the synchronisation process sends replies to each of the processes and hence processes may proceed. For example, in the case where there are n processes containing a barrier t, the synchronisation process will receive precisely n inputs and will then send exactly n responses; prior to the responses being sent, the processes are awaiting input on a private channel and hence block. The application of the barrier elimination function will be demonstrated in Example 5.3; first, we study how to control the synchronisation. To ensure that the correct pair of channel names are used for synchronisation we first define the function ω which captures the set of channel names introduced for synchronisation (Definition 5.2).

The functions ω, δ are logically similar and separation has only been introduced for readability. The function ω builds a set of triples (t, a', a''), where $t \in \mathbb{N}$ represents a barrier synchronisation and a' (respectively a'') are the output (respectively input) channels used by δ for synchronisation at barrier t.

Definition 5.2 (Synchronisation channels)**.** *Given a process P and an infinite sequence of channel names* a*, the set of* synchronisation channels *is* $\omega(\mathsf{a},\ P)$*, where:*

$$
\begin{array}{lcl}
\omega(\mathsf{c},\ 0) & = & \emptyset \\
\omega(\mathsf{c},\ !Q) & = & \emptyset \\
\omega(\mathsf{c},\ \nu\, n.Q) & = & \omega(\mathsf{c},\ Q) \\
\omega(\mathsf{c},\ M(x).Q) & = & \omega(\mathsf{c},\ Q) \\
\omega(\mathsf{c},\ \overline{M}\langle N\rangle.Q) & = & \omega(\mathsf{c},\ Q) \\
\omega(\mathsf{c},\ t::Q) & = & \{(t, a', a'')\} \cup \omega(\mathsf{tail}(\mathsf{tail}(\mathsf{c})),\ Q) \\
 & & \textit{where } a' = \mathsf{head}(\mathsf{c}) \textit{ and } a'' = \mathsf{head}(\mathsf{tail}(\mathsf{c})) \\
\omega(\mathsf{c},\ \text{let } x = D \text{ in } Q \text{ else } R) & = & \omega(\mathsf{odd}(\mathsf{c}),\ Q) \cup \omega(\mathsf{even}(\mathsf{c}),\ R) \\
\omega(\mathsf{c},\ Q \mid R) & = & \omega(\mathsf{odd}(\mathsf{c}),\ Q) \cup \omega(\mathsf{even}(\mathsf{c}),\ R)
\end{array}
$$

We introduce *communication* processes, defined by the grammar $R ::= \epsilon \mid u(x) \mid \overline{u}\langle M\rangle \mid R.R'$, such that $\epsilon.R = R$. We omit R in $u(x).R$ and $\overline{u}\langle M\rangle.R$ when it is ϵ. Given communication processes, synchronisation is controlled by a synchronisation process.

Definition 5.3 (Synchronisation process)**.** *Given a process P and an infinite sequence of channel names* a*, the* synchronisation process *is* $\chi(\mathsf{a}, P) = \hat{\chi}(0,\ \epsilon,\ \omega(\mathsf{a},\ P))$*, where for all integers t, communication processes R and sets of triples S, we have:*

$$
\hat{\chi}(t,\ R,\ S) = \begin{cases}
R.0 & \textit{if } S = \emptyset \\
a'(x).\hat{\chi}(t,\ R.\overline{a''}\langle x\rangle,\ S \backslash \{(t, a', a'')\}) & \textit{if } (t, a', a'') \in S \\
\quad \textit{where } x \textit{ is a fresh variable} & \\
R.\hat{\chi}(t+1,\ \epsilon,\ S) & \textit{otherwise}
\end{cases}
$$

Although communication processes are not standard processes (that is, defined by the grammar in Figure 4.1), the function χ always returns a standard process; this process is intended to control synchronisation of a process P in which barriers have been eliminated.

Accordingly, $\hat{\chi}$ is parametrised with $S = \omega(\mathsf{a},\ P)$, that is, the set of triples (t, a', a'') defining the input and output channels used by δ to eliminate barriers. For each barrier t, an input $a'(x)$ is appended to the output, and the function is recursively called with $R.\overline{a''}\langle x\rangle$ and $S\backslash\{(t, a', a'')\}$. Once all barriers under a particular barrier t have been exhausted, the process R is appended to the output and the barriers under the next barrier $t+1$ are processed. It follows, for example, given n processes in parallel all containing a barrier t (and no other barriers), that the synchronisation process is $a_1(x_1).\ldots.a_n(x_n).\overline{b_1}\langle x_1\rangle.\ldots.\overline{b_n}\langle x_n\rangle$ for some channel names $a_1, \ldots, a_n, b_1, \ldots, b_n$ and variables $x_1, \ldots, x_n$. A more complex example will be considered in Example 5.3.

Finally, a process defined in the standard ProVerif language can be derived using our compiler (Definition 5.4) and ProVerif can then be used to automatically reason with processes containing barriers.

Definition 5.4 (Synchrony compiler)**.** *The* synchrony compiler Δ *maps a process P to another process, as follows:* $\Delta(P) = \nu\,\tilde{a}.(\delta(\mathsf{a},\ P) \mid \chi(\mathsf{a}, P))$*, where* a *is an infinite sequence of distinct channel names such that* $\mathsf{a} \cap (\mathrm{fn}(P) \cup \mathrm{bn}(P)) = \emptyset$ *and* $\tilde{a} = \mathrm{fn}(\delta(\mathsf{a},\ P)) \cap \mathsf{a}$*.*

The application of our compiler is demonstrated in Example 5.3.

Example 5.3. *Consider the process* $1::\overline{c}\langle k\rangle.2::\overline{c}\langle m\rangle \mid 2::\overline{c}\langle n\rangle \mid 3::\overline{c}\langle s\rangle$ *and let* $\mathsf{a} = (a_1, a_2, \ldots)$*. We have:*

$$
\begin{aligned}
\delta(\mathsf{a},\ P) &= \delta((a_1, a_3, \ldots),\ 1::\overline{c}\langle k\rangle.2::\overline{c}\langle m\rangle) \mid \delta((a_2, a_4, \ldots),\ 2::\overline{c}\langle n\rangle \mid 3::\overline{c}\langle s\rangle) \\
&= \overline{a_1}\langle a_1\rangle.a_3(x).\delta((a_5, a_7, \ldots),\ \overline{c}\langle k\rangle.2::\overline{c}\langle m\rangle) \mid \\
&\qquad \delta((a_2, a_6, \ldots),\ 2::\overline{c}\langle n\rangle) \mid \delta((a_4, a_8, \ldots),\ 3::\overline{c}\langle s\rangle) \\
&= \overline{a_1}\langle a_1\rangle.a_3(x).\overline{c}\langle k\rangle.\delta((a_5, a_7, \ldots),\ 2::\overline{c}\langle m\rangle) \mid \\
&\qquad \overline{a_2}\langle a_2\rangle.a_6(y).\delta((a_{10}, a_{14}, \ldots),\ \overline{c}\langle n\rangle) \mid \overline{a_4}\langle a_4\rangle.a_8(z).\delta((a_{12}, a_{16}, \ldots),\ \overline{c}\langle s\rangle) \\
&= \overline{a_1}\langle a_1\rangle.a_3(x).\overline{c}\langle k\rangle.\overline{a_5}\langle a_5\rangle.a_7(x').\delta((a_9, a_{11}, \ldots),\ \overline{c}\langle m\rangle) \mid \\
&\qquad \overline{a_2}\langle a_2\rangle.a_6(y).\overline{c}\langle n\rangle \mid \overline{a_4}\langle a_4\rangle.a_8(z).\overline{c}\langle s\rangle \\
&= \overline{a_1}\langle a_1\rangle.a_3(x).\overline{c}\langle k\rangle.\overline{a_5}\langle a_5\rangle.a_7(x').\overline{c}\langle m\rangle \mid \overline{a_2}\langle a_2\rangle.a_6(y).\overline{c}\langle n\rangle \mid \overline{a_4}\langle a_4\rangle.a_8(z).\overline{c}\langle s\rangle
\end{aligned}
$$

$$
\begin{aligned}
\omega(\mathsf{a},\ P) &= \omega((a_1, a_3, \dots),\ 1::\overline{c}\langle k\rangle.2::\overline{c}\langle m\rangle) \cup \omega((a_2, a_4, \dots),\ 2::\overline{c}\langle n\rangle \mid 3::\overline{c}\langle s\rangle) \\
&= \{(1, a_1, a_3)\} \cup \omega((a_5, a_7, \dots),\ \overline{c}\langle k\rangle.2::\overline{c}\langle m\rangle) \cup \\
&\qquad \omega((a_2, a_6, \dots),\ 2::\overline{c}\langle n\rangle) \cup \omega((a_4, a_8, \dots),\ 3::\overline{c}\langle s\rangle) \\
&= \{(1, a_1, a_3), (2, a_2, a_6), (3, a_4, a_8)\} \cup \omega((a_5, a_7, \dots),\ 2::\overline{c}\langle m\rangle) \cup \\
&\qquad \omega((a_{10}, a_{14}, \dots),\ \overline{c}\langle n\rangle) \cup \omega((a_{12}, a_{16}, \dots),\ \overline{c}\langle s\rangle) \\
&= \{(1, a_1, a_3), (2, a_5, a_7), (2, a_2, a_6), (3, a_4, a_8)\} \cup \omega((a_9, a_{11}, \dots),\ \overline{c}\langle m\rangle) \\
&= \{(1, a_1, a_3), (2, a_5, a_7), (2, a_2, a_6), (3, a_4, a_8)\}
\end{aligned}
$$

$$
\begin{aligned}
\chi(\mathsf{a}, P) &= \hat{\chi}(1,\ \epsilon,\ \{(1, a_1, a_3), (2, a_5, a_7), (2, a_2, a_6), (3, a_4, a_8)\}) \\
&= a_1(x).\hat{\chi}(1,\ \overline{a_3}\langle x\rangle,\ \{(2, a_5, a_7), (2, a_2, a_6), (3, a_4, a_8)\}) \\
&= a_1(x).\overline{a_3}\langle x\rangle.\hat{\chi}(2,\ \epsilon,\ \{(2, a_5, a_7), (2, a_2, a_6), (3, a_4, a_8)\}) \\
&= a_1(x).\overline{a_3}\langle x\rangle.a_5(x').\hat{\chi}(2,\ \overline{a_7}\langle x'\rangle,\ \{(2, a_2, a_6), (3, a_4, a_8)\}) \\
&= a_1(x).\overline{a_3}\langle x\rangle.a_5(x').a_2(y).\hat{\chi}(2,\ \overline{a_7}\langle x'\rangle.\overline{a_6}\langle y\rangle,\ \{(3, a_4, a_8)\}) \\
&= a_1(x).\overline{a_3}\langle x\rangle.a_5(x').a_2(y).\overline{a_7}\langle x'\rangle.\overline{a_6}\langle y\rangle.\hat{\chi}(3,\ \epsilon,\ \{(3, a_4, a_8)\}) \\
&= a_1(x).\overline{a_3}\langle x\rangle.a_5(x').a_2(y).\overline{a_7}\langle x'\rangle.\overline{a_6}\langle y\rangle.a_4(z).\hat{\chi}(3,\ \overline{a_8}\langle z\rangle,\ \emptyset) \\
&= a_1(x).\overline{a_3}\langle x\rangle.a_5(x').a_2(y).\overline{a_7}\langle x'\rangle.\overline{a_6}\langle y\rangle.a_4(z).\overline{a_8}\langle z\rangle
\end{aligned}
$$

Observe $\mathsf{a} \cap (\mathrm{fn}(P) \cup \mathrm{bn}(P)) = \emptyset$, $\tilde{a} = a_1, \dots, a_8$ *and hence it follows that* $\Delta(P)$ *is defined as the process:*

$$
\begin{aligned}
\nu\, \tilde{a}.(&\overline{a_1}\langle a_1\rangle.a_3(x).\overline{c}\langle k\rangle.\overline{a_5}\langle a_5\rangle.a_7(x').\overline{c}\langle m\rangle \mid \overline{a_2}\langle a_2\rangle.a_6(y).\overline{c}\langle n\rangle \mid \overline{a_4}\langle a_4\rangle.a_8(z).\overline{c}\langle s\rangle \mid \\
&a_1(x).\overline{a_3}\langle x\rangle.a_5(x').a_2(y).\overline{a_7}\langle x'\rangle.\overline{a_6}\langle y\rangle.a_4(z).\overline{a_8}\langle z\rangle)
\end{aligned}
$$

By inspection we can observe that the process enforces the expected behaviour; that is, the name k *must be output before* m, n *and* s. *This can be witnessed by annotating the process with events and analysing using ProVerif (see Appendix D).*

We will now revisit Example 5.2 to show why uniformity is an insufficient proof technique to prove observational equivalence of certain biprocesses containing barriers.

Example 5.4. *Recall the biprocess* $P = \overline{c}\langle m\rangle.1::\overline{c}\langle \mathrm{diff}[m,n]\rangle \mid 1::\overline{c}\langle \mathrm{diff}[n,m]\rangle$ *from Example 5.2. The process* $\Delta(P)$ *is uniform, but* $\Delta(P) \rightarrow^* Q$ *where* $Q = \overline{c}\langle \mathrm{diff}[m,n]\rangle \mid \overline{c}\langle \mathrm{diff}[n,m]\rangle$*, and* Q *is not uniform.*

The next section introduces a methodology to overcome this limitation of uniformity as a proof technique for observational equivalence.

5.2 Automated reasoning for equivalence

In this section, the compiler (Section 5.1) is extended to overcome the limitations of strong uniformity as a proof technique. This can be achieved in a static manner by *swapping* data between processes which share the same structure, as will now be demonstrated.

Example 5.5. *Recall the biprocess* $P = \overline{c}\langle \mathrm{diff}[m,n]\rangle \mid \overline{c}\langle \mathrm{diff}[n,m]\rangle$ *from Example 5.1 and observe that* $P = R\sigma \mid R\tau$*, where* $R = \overline{c}\langle x\rangle$*,* $\sigma = \{^{\mathrm{diff}[m,n]}/_x\}$ *and* $\tau = \{^{\mathrm{diff}[n,m]}/_x\}$*. By extending* diff *to substitutions, the process* $Q = R\,\mathrm{diff}[\sigma,\tau] \mid R\,\mathrm{diff}[\tau,\sigma]$ *is such that* $\mathrm{fst}(P) \equiv \mathrm{fst}(Q)$ *and* $\mathrm{snd}(P) \equiv \mathrm{snd}(Q)$*; that is, the biprocess* Q *preserves the underlying structure of* P*. It follows immediately that* $\mathrm{fst}(P) \sim \mathrm{fst}(Q)$ *and* $\mathrm{snd}(P) \sim \mathrm{snd}(Q)$*. Furthermore,* Q *satisfies strong uniformity and hence* P *satisfies observational equivalence by transitivity.*

This approach can be generalised to swapping data between sub-processes which share the same structure under the same barrier, since the semantics of barriers provide certain guarantees about dynamic process behaviour.

Example 5.6. *Recall the biprocess* $P = \overline{c}\langle m\rangle.1::\overline{c}\langle \mathrm{diff}[m,n]\rangle \mid 1::\overline{c}\langle \mathrm{diff}[n,m]\rangle$ *from Example 5.2. Intuitively,* P *can only reduce to* $R = 1::\overline{c}\langle \mathrm{diff}[m,n]\rangle \mid 1::\overline{c}\langle \mathrm{diff}[n,m]\rangle$ *in one step. Now consider the biprocess* $Q = \overline{c}\langle m\rangle.1::\overline{c}\langle \mathrm{diff}[m,m]\rangle \mid 1::\overline{c}\langle \mathrm{diff}[n,n]\rangle$ *which is obtained from* P *by swapping the second components of the* diff *operator. It follows that* $\mathrm{fst}(\Delta(P)) \sim \mathrm{fst}(\Delta(Q))$ *and* $\mathrm{snd}(\Delta(P)) \sim \mathrm{snd}(\Delta(Q))$*. Now since* $\Delta(Q)$ *trivially satisfies strong uniformity,* $\Delta(P)$ *satisfies observational equivalence by transitivity.*

These examples intuitively demonstrate the notion of swapping, and, moreover, its application for automatically proving observational equivalence. The remainder of this section will formalise swapping as an extension to our compiler.

The revised barrier elimination function permits the possibility of swapping during synchronisation. In particular, at the start of each barrier synchronisation, all free names and variables under that barrier (in our formalism this will include those names and variables bound previously) are output and a possibly distinct set of names and variables are subsequently input. This behaviour is fully captured by renaming the names and variables under that barrier in accordance with the binder used for the aforementioned input (Definition 5.5). By ensuring that swapping occurs in a suitable manner, an approximation of a process's observational behaviour can be derived; this will be achieved by the introduction of a *swapping process*. Before presenting the formal definition of our revised barrier elimination function δ', we provide an intuitive example of how δ' works without swapping.

Example 5.7. *Recall the process* $\Delta(P)$ *from Example 5.3, where* $P = 1::\overline{c}\langle k\rangle.2::\overline{c}\langle m\rangle \mid 2::\overline{c}\langle n\rangle \mid 3::\overline{c}\langle s\rangle$. *Let* $\mathsf{a} = (a_1, a_2, \dots)$ *and consider the process* $Q = \nu\, a_1, \dots, a_8.(\delta'(\mathsf{a}, P) \mid \chi(\mathsf{a}, P))$. *We have* $\delta'(\mathsf{a}, P)$ *defined as follows:*

$$\overline{a_1}\langle(\!|c, k, m|\!)\rangle.a_3(x).\overline{\pi_1(x)}\langle\pi_2(x)\rangle.\overline{a_5}\langle(\!|\pi_1(x), \pi_3(x)|\!)\rangle.a_7(x').\overline{\pi_1(x')}\langle\pi_2(x')\rangle \mid$$
$$\overline{a_2}\langle(\!|c, n|\!)\rangle.a_6(y).\overline{\pi_1(y)}\langle\pi_2(y)\rangle \mid \overline{a_4}\langle(\!|c, s|\!)\rangle.a_8(z).\overline{\pi_1(z)}\langle\pi_2(z)\rangle$$

In this instance, observe $\Delta(P) \sim Q$, *which can be witnessed using ProVerif (see Appendix D).*

Definition 5.5 (Barrier elimination function with swapping)**.** *The* barrier elimination function with swapping δ' *maps an infinite sequence of channel names* a *and a process* P *to another process, as follows,* $\delta'(\mathsf{a}, P) = \hat{\delta}'(\mathsf{a},\ \tau,\ P)$ *for the substitution* $\tau = \{\}$ *(that is,*

the empty substitution), where:

$$
\begin{array}{lcl}
\hat{\delta}'(\mathsf{c},\ \sigma,\ 0) & = & 0 \\
\hat{\delta}'(\mathsf{c},\ \sigma,\ !Q) & = & !Q\sigma \\
\hat{\delta}'(\mathsf{c},\ \sigma,\ \nu\, n.Q) & = & \nu\, n.\hat{\delta}'(\mathsf{c},\ \{^{z}/_{u} \mid \{^{z}/_{u}\} \in \sigma \text{ and } u \neq n\},\ Q) \\
\hat{\delta}'(\mathsf{c},\ \sigma,\ M(x).Q) & = & M\sigma(x).\hat{\delta}'(\mathsf{c},\ \{^{z}/_{u} \mid \{^{z}/_{u}\} \in \sigma \text{ and } u \neq x\},\ Q) \\
\hat{\delta}'(\mathsf{c},\ \sigma,\ \overline{M}\langle N\rangle.Q) & = & \overline{M}\langle N\rangle\sigma.\hat{\delta}'(\mathsf{c},\ \sigma,\ Q) \\
\hat{\delta}'(\mathsf{c},\ \sigma,\ t::Q) & = & \overline{a'}\langle(\!|u_1,\ldots,u_l|\!)\sigma\rangle.a''(y).\hat{\delta}'(\mathsf{tail}(\mathsf{tail}(\mathsf{c})),\ \tau,\ Q) \\
 & & \textit{where } a' = \mathsf{head}(\mathsf{c}),\ a'' = \mathsf{head}(\mathsf{tail}(\mathsf{c})) \textit{ and} \\
 & & \tau = \{^{\pi_1(y)}/_{u_1},\ldots,^{\pi_l(y)}/_{u_l}\} \textit{ for some names} \\
 & & \textit{and variables } u_1,\ldots,u_l,\ y \textit{ such that} \\
 & & \mathrm{fn}(Q) \cup \mathrm{fv}(Q) = u_1,\ldots,u_l \textit{ and } y \notin \mathrm{fv}(Q) \\
\hat{\delta}'(\mathsf{c},\ \sigma,\ \text{let } x = D \text{ in } Q \text{ else } R) & = & \text{let } x = D\sigma \text{ in } \hat{\delta}'(\mathsf{odd}(\mathsf{c}),\ \tau,\ Q) \\
 & & \qquad\qquad \text{else } \hat{\delta}'(\mathsf{even}(\mathsf{c}),\ \sigma,\ R) \\
 & & \textit{where } \tau = \{^{z}/_{u} \mid \{^{z}/_{u}\} \in \sigma \textit{ and } u \neq x\} \\
\hat{\delta}'(\mathsf{c},\ \sigma,\ Q \mid R) & = & \hat{\delta}'(\mathsf{odd}(\mathsf{c}),\ \sigma,\ Q) \mid \hat{\delta}'(\mathsf{even}(\mathsf{c}),\ \sigma,\ R)
\end{array}
$$

By definition of the barrier elimination function, at the start of each barrier synchronisation a process outputs all of the free names and variables under that barrier. The synchronisation process (Definition 5.3) would simply return these values, as demonstrated in Example 5.7. Intuitively, however, it is possible to swap values between sub-processes which share the same structure under a particular barrier. Since this preserves the underlying structure of the biprocesses, it is useful for proofs of observational equivalence. In order to track which processes may swap data, the function ω is modified:

$$\omega(\mathsf{c},\ t::Q) \ = \ \{(t,\ a',\ a'',\ \hat{\delta}'(\hat{\mathsf{c}},\ \sigma,\ Q))\} \cup \omega(\hat{\mathsf{c}},\ Q)$$

where $a' = \mathsf{head}(\mathsf{c})$, $a'' = \mathsf{head}(\mathsf{tail}(\mathsf{c}))$, $\hat{\mathsf{c}} = \mathsf{tail}(\mathsf{tail}(\mathsf{c}))$ and $\sigma = \{^{\pi_1(y)}/_{u_1},\ldots,^{\pi_l(y)}/_{u_l}\}$ for some names and variables $u_1,\ldots,u_l,y$ such that $\mathrm{fn}(Q)\cup\mathrm{fv}(Q) = u_1,\ldots,u_l$ and $y \notin \mathrm{fv}(Q)$.

For simplicity, this definition, and Definition 5.5, assume there is exactly one tuple of names and variables $u_1, \ldots, u_l, y$ such that $\text{fn}(Q) \cup \text{fv}(Q) = u_1, \ldots, u_l$ and $y \notin \text{fv}(Q)$; that is, the same names and variables are used by the functions ω and δ'.

The *swapping process* (Definition 5.6) is similar to the synchronisation process, but considers all possible swapping strategies in addition to synchronisation. In particular, the function $\bar{\chi}$ is introduced for this purpose. More precisely, sub-processes under a particular barrier are permitted to swap if they are syntactically equal. The consideration of the bijection $f \in \mathcal{F}$ ensures all swapping strategies are considered.

Definition 5.6 (Swapping process)**.** *Given a process P and an infinite sequence of channel names a, the set of* swapping processes *is $\chi'(\mathsf{a}, P) = \hat{\chi}'(1,\ \epsilon,\ \epsilon,\ \omega(\mathsf{a},\ P))$, where for all integers t, communication processes R, R' and sets S, we have:*

$$\hat{\chi}'(t,\ R,\ R',\ S) = \begin{cases} \{R.R'.0\} & \text{if } S = \emptyset \\ \bar{\chi}(t,\ R,\ R',\ S,\ Q) & \text{if } (t, a, b, Q) \in S \\ \hat{\chi}'(t+1,\ R.R',\ \epsilon,\ S) & \text{otherwise} \end{cases}$$

such that $\bar{\chi}(t, R, R', S, Q)$ is defined as

$$\bigcup_{f \in \mathcal{F}} \hat{\chi}'(t,\ R.a_1(y_1).\cdots.a_n(y_n),\ R'.\overline{b_1}\langle \text{diff}[y_1, y_{f(1)}]\rangle.\cdots.\overline{b_n}\langle \text{diff}[y_{f(n)}, y_n]\rangle,$$
$$S \backslash \{(t, a_1, b_1, Q), \ldots, (t, a_n, b_n, Q)\})$$

where $(t, a_1, b_1, Q), \ldots, (t, a_n, b_n, Q) \in S$, $\mathcal{F}$ is the set of bijections on $\{1, \ldots, n\}$, and variables $y_1, \ldots, y_n$ are fresh.

Correctness of barrier ordering follows as before, and moreover, the notion of swapping is intuitively sound because it occurs between syntactically equal processes.

Finally, a set of processes in the standard ProVerif language, which consider all possible swapping strategies, can be derived.

Definition 5.7 (Swapping compiler)**.** *The* swapping compiler Δ' *maps a process* P *to a set of process, as follows:* $\Delta'(P) = \{\nu\,\tilde{a}.(\delta'(\mathsf{a}, P) \mid R) \mid R \in \chi'(\mathsf{a}, P)\}$*, where* a *is an infinite sequence of distinct channel names such that* $\mathsf{a} \cap (\mathrm{fn}(P) \cup \mathrm{bn}(P)) = \emptyset$ *and* $\tilde{a} = \mathrm{fn}(\delta'(\mathsf{a}, P)) \cap \mathsf{a}$*.*

Intuitively, if there exists a swapping strategy such that the compiled process satisfies observational equivalence, then the original process satisfies observational equivalence. The application of Definition 5.7 is demonstrated by Example 5.8 and has been implemented as a tool called ProSwapper [KSR10].

Example 5.8. *Recall the process* $P = \overline{c}\langle m\rangle.1 :: \overline{c}\langle \mathrm{diff}[m, n]\rangle \mid 1 :: \overline{c}\langle \mathrm{diff}[n, m]\rangle$ *from Example 5.2 and assume* $\mathsf{a} = (a_1, a_2, \dots)$*. We have:*

$$\begin{aligned}
\delta'(\mathsf{a}, P) &= \overline{c}\langle m\rangle.\overline{a_1}\langle(\!|c, m, n|\!)\rangle.a_3(x).\overline{\pi_1(x)}\langle \mathrm{diff}[\pi_2(x), \pi_3(x)]\rangle \mid \\
&\qquad \overline{a_2}\langle(\!|c, n, m|\!)\rangle.a_4(x).\overline{\pi_1(x)}\langle \mathrm{diff}[\pi_2(x), \pi_3(x)]\rangle \\
\omega(\mathsf{a},\ P) &= \{(1, a_1, a_3, \overline{\pi_1(x)}\langle \mathrm{diff}[\pi_2(x), \pi_3(x)]\rangle), (1, a_2, a_4, \overline{\pi_1(x)}\langle \mathrm{diff}[\pi_2(x), \pi_3(x)]\rangle)\} \\
\chi'(\mathsf{a}, P) &= \{a_1(y).a_2(z).\overline{a_3}\langle \mathrm{diff}[y, y]\rangle.\overline{a_4}\langle \mathrm{diff}[z, z]\rangle, \\
&\qquad a_1(y).a_2(z).\overline{a_3}\langle \mathrm{diff}[y, z]\rangle.\overline{a_4}\langle \mathrm{diff}[z, y]\rangle\}
\end{aligned}$$

By Definition 5.7 we derive the set of processes which represent all possible swapping strategies:

$$\begin{aligned}
\{&\nu\, a_1, a_2, a_3, a_4.(\delta'(\mathsf{a}, P) \mid a_1(y).a_2(z).\overline{a_3}\langle \mathrm{diff}[y, y]\rangle.\overline{a_4}\langle \mathrm{diff}[z, z]\rangle), \\
&\qquad \nu\, a_1, a_2, a_3, a_4.(\delta'(\mathsf{a}, P) \mid a_1(y).a_2(z).\overline{a_3}\langle \mathrm{diff}[y, z]\rangle.\overline{a_4}\langle \mathrm{diff}[z, y]\rangle)\}
\end{aligned}$$

and since the latter process satisfies strong uniformity (see Appendix E), we believe the process $\Delta(P)$ *satisfies observational equivalence. (Note that, in order to express results about* P*, rather than* $\Delta(P)$*, a formal semantics for barriers must be considered; for further discussion, see Section 6.1.)*

5.3 Privacy in electronic voting

This section will demonstrate the suitability of the methodology for analysing vote privacy in electronic voting protocols. In particular, an automated analysis of vote privacy in the electronic voting protocol by Fujioka, Okamoto & Ohta [FOO92] is presented. Moreover, the generality of our approach has been demonstrated by Backes, Hriţcu & Maffei [BHM08] who prove a stronger property – namely, coercion resistance – of the protocol by Juels, Catalano & Jakobsson [JCJ02, JCJ05, JCJ10] which has been implemented by Clarkson, Chong & Myers [CCM08, CCM07] as Civitas. Our technique has also been adopted by Dahl, Delaune & Steel [DDS10] to analyse privacy in vehicular ad-hoc networks. (Note that Dahl, Delaune & Steel use an earlier version [DRS08] of this work, and Backes, Hriţcu & Maffei make use of the fundamental principles, possibly based upon preliminary results presented at Dagstul [Smy07].) In Appendix E, the model of privacy in vehicular ad-hoc networks [DDS10] is revisited and automatically analysed using ProSwapper.)

5.3.1 Case study: FOO

The FOO protocol, by Fujioka, Okamoto & Ohta [FOO92], is a seminal work based upon blind signatures. Delaune, Kremer & Ryan [KR05, DKR06, DKR09, DKR10b] have studied the protocol and present a hand-based proof of vote privacy, and Chothia *et al.* [COPD07] provide an automated analysis using μCRL. We will use the results of Section 5.2 to present an automated analysis; although this result is not new, it is useful to demonstrate our technique. The protocol description and equational theory were presented in Section 3.2.2 and we proceed immediately to an applied pi model of the voter process.

Model in applied pi

The formal specification of the voter process in FOO (Definition 5.8) follows immediately from our protocol description (Section 3.2.2). Observe that phases of the protocol are seperated within the voting process using barriers. We do not assert the secrecy of voters' keys and, accordingly, they can be modelled as free names. Of course, these keys *are* required to be secret for other properties. It follows that the process $P_{\mathsf{foo}}\{s_1/x_{\mathsf{vote}}, sk_1/x_{\mathsf{sk}}\} \mid \cdots \mid P_{\mathsf{foo}}\{s_n/x_{\mathsf{vote}}, sk_n/x_{\mathsf{sk}}\}$ models an election with n voters casting votes $s_1, \ldots, s_n$.

Definition 5.8. *The process P_{foo} modelling a voter in FOO is defined as follows*

$$
\begin{aligned}
P_{\mathsf{foo}} \;=\; & \nu\, k.\nu\, k'.\\
& \text{let } M = \mathsf{commit}(k, x_{\mathsf{vote}}) \text{ in}\\
& \text{let } \hat{M} = \mathsf{blind}(k', M) \text{ in}\\
& \overline{c}\langle(\!|\mathsf{pk}(x_{\mathsf{sk}}), \mathsf{sign}(x_{\mathsf{sk}}, \hat{M})|\!)\rangle.c(y).\\
& \text{if } \mathsf{checksign}(\mathsf{pk}(sk_R), y) = \mathsf{true} \text{ then}\\
& \text{if } \mathsf{getmsg}(y) = \hat{M} \text{ then}\\
& \text{let } M' = \mathsf{unblind}(k', y) \text{ in}\\
& 1::\overline{c}\langle(\!|M, M'|\!)\rangle.2::c(z).\\
& \text{if } (\!|\pi_2(z), \pi_3(z)|\!) = (\!|M, M'|\!) \text{ then}\\
& \overline{c}\langle(\!|\pi_1(z), k|\!)\rangle
\end{aligned}
$$

where x_{sk} is variable referring to the voter's signing key and x_{vote} is a variable referring to voter's vote.

Analysis: Vote privacy

Consider two voters $\mathcal{A}$, $\mathcal{B}$ and two candidates s, s'. Based upon [KR05, DKR06, DKR09, DKR10b], we formalise vote privacy for two voters with the assertion that an adversary cannot distinguish between a situation in which voter $\mathcal{A}$ votes for candidate s and voter $\mathcal{B}$ votes for candidate s', from another one in which $\mathcal{A}$ votes s' and $\mathcal{B}$ votes s. Formally,

this is written as the equivalence:

$$P_{\mathsf{foo}}\{sk_A/x_{\mathsf{sk}}, s/x_{\mathsf{vote}}\} \mid P_{\mathsf{foo}}\{sk_B/x_{\mathsf{sk}}, s'/x_{\mathsf{vote}}\} \sim P_{\mathsf{foo}}\{sk_A/x_{\mathsf{sk}}, s'/x_{\mathsf{vote}}\} \mid P_{\mathsf{foo}}\{sk_B/x_{\mathsf{sk}}, s/x_{\mathsf{vote}}\}$$

which can be checked using the methodology introduced. To provide further insight into how the swapping compiler works, let us consider how to informally prove this equivalence. Let us call the left hand side Lhs, and the right hand side Rhs. By definition of equivalence we must show that the Lhs is indistinguishable from the Rhs. Indeed, as the Lhs evolves, the corresponding evolution of the Rhs is the one that mimics $\mathcal{A}$ moves on the Lhs with $\mathcal{A}$ moves on the Rhs, and $\mathcal{B}$ moves on the Lhs with $\mathcal{B}$ moves on the Rhs, up to the first barrier. After the first synchronisation, $\mathcal{A}$ moves on the Lhs are mimicked by $\mathcal{B}$ moves on the Rhs, and $\mathcal{B}$ moves on the Lhs are mimicked by $\mathcal{A}$ moves on the Rhs. The reason for the 'swap' in mimicking is to ensure that no context can distinguish the actions performed. Before the synchronisation, the output data produced by $\mathcal{A}$ on both the Lhs and the Rhs are indistinguishable (and similarly for $\mathcal{B}$); observe that on the Lhs the output by $\mathcal{A}$ reveals

$$(\!|\mathsf{pk}(sk_A),\ \mathsf{sign}(sk_A, \mathsf{blind}(k'_a, \mathsf{commit}(k_a, s))),\ \mathsf{blind}(k'_a, \mathsf{commit}(k_a, s))|\!)$$

which can be matched on the Rhs by

$$(\!|\mathsf{pk}(sk_A),\ \mathsf{sign}(sk_A, \mathsf{blind}(k'_a, \mathsf{commit}(k_a, s'))),\ \mathsf{blind}(k'_a, \mathsf{commit}(k_a, s'))|\!)$$

where names k_a, k'_a are under restriction in both the Lhs and the Rhs. (Note that indistinguishability between the Lhs and the Rhs is due to the properties of blinding.) After the first and second synchronisation, $\mathcal{A}$ will reveal

$$(\!|\mathsf{sign}(sk_R, \mathsf{commit}(k_a, s)),\ \mathsf{commit}(k_a, s)|\!) \quad \text{and} \quad (\!|\ell,\ k_a|\!)$$

on the Lhs and

$$(\!|\mathsf{sign}(sk_R, \mathsf{commit}(k_a, s')),\ \mathsf{commit}(k_a, s')|\!) \quad \text{and} \quad (\!|\ell,\ k_a|\!)$$

on the Rhs, where ℓ is chosen by the adversarial environment; that is, $\mathcal{A}$ reveals her vote s on the Lhs and her vote s' on the Rhs (similarly, $\mathcal{B}$ will reveal s' on the Lhs and s on the Rhs). Hence, after the first synchronisation the actions mimicked are swapped. The compiler manages the swapping, and hence ProSwapper can be used to automatically analyse privacy in FOO (see Appendix E).

5.4 Summary

This chapter extends the process syntax of ProVerif to include barriers, for the purpose of synchronisation, and defines an automated methodology for proving equivalence. The result has been implemented as ProSwapper and the suitability of the tool is demonstrated by analysing vote privacy in the FOO electronic voting protocol. Furthermore, the fundamental principles of our methodology have been adopted by the wider research community: Dahl, Delaune & Steel [DDS10] use an preliminary version of this chapter (formally presented in [DRS08]) to analyse privacy in vehicular ad-hoc networks, and Backes, Hriţcu & Maffei [BHM08] use the basic concept (presumably based upon [Smy07]) to show coercion resistance in electronic voting.

Part IV

Evaluation

6

Further work and conclusion

In this thesis, we advance the capabilities of the formal verification community by *defining symbolic definitions* of security properties (Part II) and *developing procedures for evaluation* of security properties (Part III). The key contributions are:

- A definition of election verifiability for electronic voting protocols.
- A definition of user-controlled anonymity for Direct Anonymous Attestation schemes.
- An automated analysis technique for observational equivalence.

This final chapter presents future work and a further appraisal of results.

6.1 Further work

We begin by discussing possible future research directions arising from Chapters 3–5 and identifying open questions that have emerged from this thesis.

6.1.1 Election verifiability in electronic voting

The definition of election verifiability includes three aspects: individual, universal and eligibility verifiability. The conditions which these properties must satisfy are believed to be necessary, but may not be sufficient. Ultimately, a verifiable election should convince the most sceptic of voters and/or election observers that votes have been recorded, tallied

and declared correctly. Establishing whether voting protocols which satisfy our definition, are also considered to achieve this objective, remains an open research problem.

The definition of election verifiability is based upon earlier work [SRKK10] in which automated reasoning was considered. Extending Chapter 3 to include an automated procedure for evaluating election verifiability would appear to be a logical step; however, this has been precluded by the lack of efficient tools for evaluating our definition in the general case. In particular, the tests must be embedded within processes (cf. [SRKK10, §4]) and parametrised by an arbitrary number of voters, but such parametrisation is not currently supported by ProVerif. Providing automated tool support for processes which require parametrisation is further work and, moreover, such a tool would be useful for analysing security properties in a variety of cryptographic protocols.

The study of election verifiability identified eligibility verifiability – a property that has been largely neglected by the literature and is only satisfied by a few protocols – as a mechanism to detect ballot stuffing. As an alternative to eligibility verifiability, some electronic voting protocols provide a weaker notion of eligibility – which cannot be checked by voters or observers – under various trust assumptions. For example, such an eligibility property may assert that if the election officials follow the protocol, then each ballot published on the bulletin board was cast by a registered voter and at most one ballot is tallied per voter. Indeed, this assumption was thought to be reasonable in the presidential election at the Catholic University of Louvain which used Helios. However, similar assumptions must not be made about national elections. Accordingly, the study of electronic voting protocols which simultaneously satisfy election verifiability and other desirable properties, such as privacy and usability, remains an open problem.

6.1.2 Anonymity in Direct Anonymous Attestation

Direct Anonymous Attestation is a relatively new concept and its properties merit further study. In particular, user-controlled traceability and correctness have received limited at-

tention. Furthermore, new properties (for example, *non-frameability* [Che10, Che11]) are emerging. Extending this work to include a complete definition of DAA properties would be an interesting direction for future research. Moreover, establishing a unified definition which includes all properties (that is, correctness, non-frameability, user-controlled anonymity and user-controlled traceability) would be of interest to reduce the verification workload. As a starting point, this could be achieved by suitably developing the notion of a Direct Anonymous Attestation process specification to distinguish between operations performed by the host and those performed by the TPM. This distinction is not necessary for our definition of user-controlled anonymity because this property can only be achieved if both the host and TPM are trusted. By contrast, a corrupt host – even in collaboration with a corrupt TPM (where the TPM is known to be rogue) – should not be able to violate traceability properties and therefore an alternative process specification, which distinguishes between actions performed by the host and TPM, would be required.

For user-controlled anonymity it is necessary to ensure a distinct basename is used during phase 2 (when $bsn \neq \perp$). Since the applied pi calculus does not record state, this is achieved by an abstraction. Proving that this abstraction does not lose generality is an open problem. In addition, a stateful variant of the applied pi calculus would be useful.

The study of user-controlled anonymity in Direct Anonymous Attestation schemes is particular interesting due to the real-world deployment of the TPM. However, verification of cryptographic protocols does not ensure their secure deployment because vulnerabilities may be introduced during implementation [GLP05, BFGT06, BFGT08, APW09]. Automatically deriving models from implementations and verifying security properties using ProVerif has been considered [BFGT06, BFGT08, BFGS08] and verified reference implementations have been introduced to aid the development of secure systems [BFG06, BCFZ08, MGR09]. In the context of DAA, automatically verifying implementations (for example, [Smy06, SS08, SGPV09, CPS10]) and developing a secure reference implementation remains as future work.

6.1.3 Observational equivalence and barriers

Evaluating whether the synchronous compiler (Definition 5.4) enforces the informally discussed semantic behaviour of barrier synchronisation is difficult to establish due to the complexity of definition. In this instance, it would be more suitable to formally define the semantics in the calculus of ProVerif and prove that the synchronous compiler encodes an approximation of the desired observational behaviour. Moreover, by extending the definition of observational equivalence to consider barriers, stronger results could be stated; that is, results could be stated about P rather than $\Delta(P)$ (see, for example, Examples 5.6 & 5.8 and Section 5.3.1).

Proving soundness of the proof technique is ongoing work. At a high level it would be sufficient to show that for all biprocesses P, if there exists $Q \in \Delta'(P)$ such that Q satisfies strong uniformity, then $\Delta(P)$ satisfies observational equivalence. Accordingly, it would be sufficient to prove that for all biprocesses P, we have for all $Q \in \Delta'(P)$, that $\mathrm{fst}(\Delta(P)) \sim \mathrm{fst}(Q)$ and $\mathrm{snd}(\Delta(P)) \sim \mathrm{snd}(Q)$. (As previously stated, the formal semantics of barriers are not defined and thus we cannot state observational equivalence results involving processes with barriers; that is, we must rely upon the synchronous compiler.)

The definition of labelled bisimilarity [AF01, §4.3], which complements observational equivalence (Definition 2.1), has proven to be useful for handwritten proofs of equivalence. A similar notion to complement Definition 4.1 may be useful to prove observational equivalence in cases where automated reasoning is not possible. Furthermore, general procedures which consider arbitrary processes (rather than biprocesses) could be sought.

6.1.4 Emerging directions

The formalisation of user-controlled anonymity in Direct Anonymous Attestation (Chapter 4) is similar, but distinct, from the formalisation of privacy in electronic voting (Section 5.3) and vehicular ad-hoc networks (Appendix E.2). In electronic voting, vote privacy

for two voters $\mathcal{A}$, $\mathcal{B}$ is formalised as the indistinguishability between a situation in which voter $\mathcal{A}$ votes for candidate s and voter $\mathcal{B}$ votes for candidate s', from another in which $\mathcal{A}$ votes s' and $\mathcal{B}$ votes s. Using terminology from our formalisation of user-controlled anonymity, this essentially corresponds to two voters participating in the *challenge* phase. By comparison, in DAA we consider a single signer during the challenge phase. It is necessary to consider two voters for vote privacy because the election outcome will be made public and this would represent an observable distinction if only a single voter participated in the challenge, that is, we can distinguish between an election outcome of s and s'. Establishing a generic class of definitions suitable for analysing privacy in a variety of settings would be desirable to obviate the need for numerous protocol dependent privacy definitions (for example, [KR05, Cho06, BHM08, DKR09, ACRR10, BCH10, DDS10]). Moreover, such a framework would allow the relative degrees of privacy to be studied.

6.2 Conclusion

This thesis will aid the secure design of cryptographic protocols and facilitate the evaluation of existing schemes by advancing formal method verification techniques. Our definition of election verifiability has been shown to be suitable for the analysis of a wide range of electronic voting protocols and its future application will allow the thorough analysis of schemes prior to deployment, thereby helping ensure integral elections. The study of user-controlled anonymity in RSA-based Direct Anonymous Attestation discovered a vulnerability and established a fix which potentially avoided violations of privacy for TPM users. Our definition of user-controlled anonymity will also facilitate the analysis of future DAA schemes, helping prevent privacy invasions. Finally, the adoption of our reasoning technique for observational equivalence, by the research community, gives us confidence that the methodology is of value and, moreover, can be used to evaluate numerous security properties including vote privacy and coercion resistance in electronic voting, and privacy in vehicular ad-hoc networks.

Part V

Appendices

A

Election verifiability results

This appendix contains proofs supporting Chapter 3.

A.1 Election verifiability in raising hands

This appendix presents a proof of election verifiability in raising hands (Example 3.1). The proof illustrates an interesting aspect of our definition: we do not require a voter to identify her credential, it is sufficient for a voter to identify a credential. If desired, a stronger definition of election verifiability can be constructed with the additional soundness condition $\Phi_i^{IV}\sigma \wedge \Phi_i^{IV}\{\hat{w}'/\hat{w}\}\sigma \Rightarrow \pi_i(\hat{w})\sigma \simeq \pi_i(\hat{w}')\sigma$.

A.1.1 Proof of Proposition 3.1

Proof. Suppose $m \in \mathbb{N}$ and tests Φ^{IV}, Φ_m^{UV}, Φ_m^{EV} are given as follows:

$$
\begin{aligned}
\Phi^{IV} &\mathrel{\widehat{=}} y =_E (\!|\mathsf{pk}(r), \mathsf{sign}(r,v)|\!) \wedge \mathsf{checksign}(x_{pk}, w) =_E \mathsf{true} \\
\Phi_m^{UV} &\mathrel{\widehat{=}} \hat{v} =_E (\!|\mathsf{getmsg}(\pi_2(\pi_1(\hat{y}))), \ldots, \mathsf{getmsg}(\pi_2(\pi_m(\hat{y})))|\!) \\
\Phi_m^{EV} &\mathrel{\widehat{=}} \textstyle\bigwedge_{1\le i\le m} \big(\mathsf{checksign}(x_{pk}, \pi_i(\hat{w})) =_E \mathsf{true} \wedge \pi_1(\pi_i(\hat{y})) =_E \mathsf{getmsg}(\pi_i(\hat{w})) \\
&\qquad \wedge \mathsf{checksign}(\pi_1(\pi_i(\hat{y})), \pi_2(\pi_i(\hat{y}))) =_E \mathsf{true} \\
&\qquad\quad \wedge \mathsf{snd}(\mathsf{snd}(\mathsf{snd}(\pi_i(\hat{y})))) =_E \varnothing\big) \wedge \mathsf{snd}^m(\hat{w}) =_E \varnothing \wedge \mathsf{snd}^m(\hat{y}) =_E \varnothing
\end{aligned}
$$

where $r = \mathrm{rv}(\Phi^{IV})$. We will now show for all names $s_1, \ldots, s_n$ that the conditions of Definition 3.7 hold.

(3.1) Suppose C is a context, B is a process, σ is a substitution, $\tilde{n}$ is a tuple of names and i, j are integers where $C[\mathsf{VP}^{+}_{\mathsf{n}}(s_1, \ldots, s_n)]$ $(\rightarrow^{*} \xrightarrow{\alpha} \rightarrow^{*})^{*}$ B, $\varphi(B) \equiv \nu\tilde{n}.\sigma$, $\Phi^{IV}\{^{s_i}/_{v}, ^{r_i}/_{r}, ^{\pi_i(\hat{w})}/_{w}\}\sigma$ and $\Phi^{IV}\{^{s_j}/_{v}, ^{r_j}/_{r}, ^{\pi_j(\hat{w})}/_{w}\}\sigma$. It follows that $\pi_1(y)\sigma =_E \mathsf{pk}(r_i)\sigma =_E \mathsf{pk}(r_j)\sigma$. Since the record variables r_i, r_j are handles for fresh nonces generated by name restriction in the process A_{ex} and sent to $V_{\mathsf{ex},i}$, $V_{\mathsf{ex},j}$ using a private channel, it follows that $i = j$.

(3.2) We prove a stronger result: namely, the condition holds for all substitutions σ. Suppose σ is an arbitrary substitution such that $\Phi^{UV}_{m}\sigma$ and $\Phi^{UV}_{m}\{^{\hat{v}'}/_{\hat{v}}\}\sigma$ hold. We have

$$\hat{v}\sigma =_E (\!|\mathsf{getmsg}(\pi_2(\pi_1(\hat{y}))), \ldots, \mathsf{getmsg}(\pi_2(\pi_m(\hat{y})))|\!)\sigma =_E \hat{v}'\sigma$$

and immediately derive $\hat{v}\sigma \simeq \hat{v}'\sigma$.

(3.3) Again, we will show that the condition holds for all substitutions σ. Suppose σ is an arbitrary substitution such that $\bigwedge_{1 \leq i \leq n} \Phi^{IV}_{i}\{^{\pi_i(\hat{y})}/_{y}\}\sigma$ and $\Phi^{UV}_{m}\sigma$ hold, where $n = m$. By $\bigwedge_{1 \leq i \leq n} \Phi^{IV}_{i}\{^{\pi_i(\hat{y})}/_{y}\}\sigma$ it must be the case for all $1 \leq i \leq n$ that $\pi_i(\hat{y})\sigma =_E (\!|\mathsf{pk}(r_i), \mathsf{sign}(r_i, s_i)|\!)\sigma$ and hence $\mathsf{getmsg}(\pi_2(\pi_i(\hat{y})))\sigma =_E s_i$. Moreover, since $n = m$ and $\Phi^{UV}_{m}\sigma$, we have

$$\hat{v}\sigma =_E (\!|\mathsf{getmsg}(\pi_2(\pi_1(\hat{y}))), \ldots, \mathsf{getmsg}(\pi_2(\pi_m(\hat{y})))|\!)\sigma =_E (\!|s_1, \ldots, s_n|\!)$$

The result $(\!|s_1, \ldots, s_n|\!) \simeq \hat{v}\sigma$ follows.

(3.5) We prove a stronger result: namely, for any substitution σ the condition holds. Suppose σ is an arbitrary substitution such that $\Phi^{EV}_{m}\sigma$ and $\Phi^{EV}_{m}\{^{x'}/_{x} \mid x \in X \backslash \hat{y}\}\sigma$ hold. For all $1 \leq i \leq m$, we have

$$\begin{aligned}\mathsf{getmsg}(\pi_i(\hat{w}))\sigma =_E \pi_1(\pi_i(\hat{y}))\sigma \wedge \pi_1(\pi_i(\hat{y}))\sigma =_E \mathsf{getmsg}(\pi_i(\hat{w}'))\sigma \\ \wedge \mathsf{checksign}(\pi_i(\hat{w}), x_{pk})\sigma =_E \mathsf{true} \wedge \mathsf{checksign}(\pi_i(\hat{w}'), x_{pk})\sigma =_E \mathsf{true}\end{aligned}$$

and, by inspection of the equational theory, it must be the case that $\pi_i(\hat{w})\sigma = \pi_i(\hat{w}')\sigma =$

$\mathsf{sign}(K_i, \pi_1(\pi_i(\hat{y})))\sigma$, where $x_{pk}\sigma = \mathsf{pk}(K_i)$. By $\Phi_m^{EV}\sigma$ and $\Phi_m^{EV}\{^{x'}/_x \mid x \in X\backslash\hat{y}\}\sigma$ we also have $\mathsf{snd}^m(\hat{w}) =_E \mathsf{snd}^m(\hat{w}') =_E \varnothing$. It must be the case that $\hat{w}\sigma = \hat{w}'\sigma = (\!|\mathsf{sign}(K_1, \pi_1(\pi_1(\hat{y}))), \ldots, \mathsf{sign}(K_m, \pi_1(\pi_m(\hat{y})))|\!)\sigma$ and hence $\hat{w}\sigma \simeq \hat{w}'\sigma$.

(3.6) Suppose C is a context and B is a process such that $C[\mathsf{VP}_{\mathsf{n}}^+(s_1, \ldots, s_n)]$ $(\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*$ B, $\varphi(B) \equiv \nu\tilde{n}.\sigma$, and $\Phi_m^{EV}\sigma \wedge \Phi_m^{EV}\{^{x'}/_x \mid x \in X\backslash\hat{w}\}\sigma$. For all $1 \leq i \leq m$, we have $\mathsf{checksign}(x_{pk}, \pi_i(\hat{w}))\sigma =_E \mathsf{true}$, where $x_{pk}\sigma = \mathsf{pk}(sk_A)$ and $sk_A \in \tilde{n}$. By inspection of the equational theory it is the case that $\pi_i(\hat{w})\sigma =_E \mathsf{sign}(sk_A, M_i)$ for some term M_i. Since the signing key is under restriction and, by inspection of the voting process, it follows that for all $1 \leq i \leq m$ we have $M_i =_E \mathsf{pk}(sk_{V_i})$, where $sk_{V_i} \in \tilde{n}$. By $\Phi_m^{EV}\sigma$ and $\Phi_m^{EV}\{^{x'}/_x \mid x \in X\backslash\hat{w}\}\sigma$ we also have $\pi_1(\pi_i(\hat{y}))\sigma =_E \mathsf{getmsg}(\pi_i(\hat{w}))\sigma =_E \pi_1(\pi_i(\hat{y}'))\sigma$, that is,

$$\pi_1(\pi_i(\hat{y}))\sigma =_E \mathsf{pk}(sk_{V_i}) =_E \pi_1(\pi_i(\hat{y}'))\sigma.$$

Moreover, $\mathsf{checksign}(\pi_1(\pi_i(\hat{y})), \pi_2(\pi_i(\hat{y})))\sigma =_E \mathsf{true} \wedge \mathsf{checksign}(\pi_1(\pi_i(\hat{y}')), \pi_2(\pi_i(\hat{y}')))\sigma =_E \mathsf{true}$. Since the signing keys $sk_{V_1}, \ldots, sk_{V_m}$ are under restriction it follows for all $1 \leq i \leq m$ that

$$\pi_2(\pi_i(\hat{y}))\sigma =_E \mathsf{sign}(sk_{V_i}, s_i) =_E \pi_2(\pi_i(\hat{y}'))\sigma$$

Furthermore, for all $1 \leq i \leq m$, we have $\mathsf{snd}(\mathsf{snd}(\mathsf{snd}(\pi_i(\hat{y})))) =_E \mathsf{snd}(\mathsf{snd}(\mathsf{snd}(\pi_i(\hat{y}')))) =_E \varnothing$ and hence

$$\pi_i(\hat{y})\sigma =_E (\!|\mathsf{pk}(sk_{V_i}), \mathsf{sign}(sk_{V_i}, s_i)|\!) =_E \pi_i(\hat{y}')\sigma$$

Since $\mathsf{snd}^m(\hat{y})\sigma =_E \mathsf{snd}^m(\hat{y}')\sigma =_E \varnothing$, we have $\hat{y}\sigma =_E \hat{y}'\sigma$ and hence $\hat{y}\sigma \simeq \hat{y}'\sigma$.

(3.7) Suppose C is a context and B is a process such that $C[\mathsf{VP}_{\mathsf{n}}^+(s_1, \ldots, s_n)]$ $(\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*$ B, $\varphi(B) \equiv \nu\tilde{n}.\sigma$, and $\bigwedge_{1\leq i\leq n} \Phi_i^{IV}\{^{\pi_i(\hat{y})}/_y\}\sigma \wedge \Phi_n^{EV}\{^{\hat{w}'}/_{\hat{w}}\}\sigma \wedge \mathsf{snd}^n(\hat{w})\sigma =_E \varnothing$ holds. For all $1 \leq i \leq n$, we have $\mathsf{checksign}(x_{pk}, \pi_i(\hat{w}))\sigma =_E \mathsf{true}$, where $x_{pk}\sigma = \mathsf{pk}(sk_A)$ and $sk_A \in \tilde{n}$. By inspection of the equational theory it is the case that $\pi_i(\hat{w})\sigma =_E \mathsf{sign}(sk_A, M_i)$ for some term M_i. Since the signing key is under restriction and, by inspection of the voting process, it follows for all $1 \leq i \leq n$ that $M_i =_E \mathsf{pk}(sk_{V_i})$, where $sk_{V_i} \in \tilde{n}$. Moreover,

$\mathsf{snd}^n(\hat{w})\sigma =_E \varnothing$ and hence

$$\hat{w}\sigma =_E (\!|\mathsf{sign}(sk_A, \mathsf{pk}(sk_{V_1})), \ldots, \mathsf{sign}(sk_A, \mathsf{pk}(sk_{V_n}))|\!)$$

Similarly, we have $1 \leq i \leq n$ that $\mathsf{checksign}(x_{pk}, \pi_i(\hat{w}'))\sigma =_E \mathsf{true}$, where $x_{pk}\sigma = \mathsf{pk}(sk_A)$ and $sk_A \in \tilde{n}$. As before, $\pi_i(\hat{w}')\sigma =_E \mathsf{sign}(sk_A, N_i)$ by inspection of the equational theory. Since the signing key is under restriction and, by inspection of the voting process (in particular, observe that A_{ex} outputs exactly n messages of the form $\mathsf{sign}(sk_A, N_i)$), there exists a permutation χ defined over $\{1, \ldots, n\}$ such that for all $1 \leq i \leq n$ we have $N_i =_E \mathsf{pk}(sk_{V_{\chi(i)}})$. Moreover, by $\Phi_n^{EV}\{^{\hat{w}'}/_{\hat{w}}\}\sigma$ we have $\mathsf{snd}^n(\hat{w}') =_E \varnothing$ and hence

$$\hat{w}'\sigma =_E (\!|\mathsf{sign}(sk_A, \mathsf{pk}(sk_{V_{\chi(1)}})), \ldots, \mathsf{sign}(sk_A, \mathsf{pk}(sk_{V_{\chi(n)}}))|\!)$$

The result $\hat{w}\sigma \simeq \hat{w}'\sigma$ follows.

(3.8) Suppose $s_1, \ldots, s_n$ are names and consider the context

$$\begin{aligned} C \;\hat{=}\; & c(w_1).\ldots.c(w_n).\overline{c}\langle(\!|w_1, \ldots, w_n|\!)\rangle. \\ & c(y_1).\ldots.c(y_n).\overline{c}\langle(\!|y_1, \ldots, y_n|\!)\rangle. \\ & \overline{c}\langle(\!|\mathsf{getmsg}(\pi_2(y_1)), \ldots, \mathsf{getmsg}(\pi_2(y_n))|\!)\rangle \end{aligned}$$

We have $C[\mathsf{VP}_n^+(s_1, \ldots, s_n)]\ (\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*\ B$ such that $\varphi(B) \equiv \nu sk_A, sk_{V_1}, \ldots, sk_{V_n}.\sigma$, where

$$\sigma = \{^{sk_{V_1}}/_{r_1}, \ldots, ^{sk_{V_n}}/_{r_n}, (\!|s_1, \ldots, s_n|\!)/_{\hat{v}}, (\!|\mathsf{sign}(sk_A, \mathsf{pk}(sk_{V_1})), \ldots, \mathsf{sign}(sk_A, \mathsf{pk}(sk_{V_n}))|\!)/_{\hat{w}},$$
$$^{\mathsf{pk}(sk_A)}/_{x_{pk}}, (\!|(\!|\mathsf{pk}(sk_{V_1}), \mathsf{sign}(sk_{V_1}, s_1)|\!), \ldots, (\!|\mathsf{pk}(sk_{V_n}), \mathsf{sign}(sk_{V_n}, s_n)|\!)|\!)/_{\hat{y}}\}$$

It can trivially be seen that $\bigwedge_{1 \leq i \leq n} \Phi_i^{IV}\{\pi_i(\hat{y})/_y\}\sigma \wedge \Phi_m^{UV}\sigma \wedge \Phi_m^{EV}\sigma$. □

A.2 Election verifiability in Civitas

This appendix presents a proof of election verifiability in Civitas. The following lemmata are introduced to demonstrate useful properties of our equational theory (pp59). We make use of the notation $\hat{M} \overset{\bullet}{\simeq} \hat{M}'$ to denote that the ciphertext tuples $\hat{M}$, $\hat{M}'$ are defined over the same plaintexts for some public key K, that is, $\hat{M} =_E (\!|\mathsf{penc}(K, R_1, N_1), \ldots, \mathsf{penc}(K, R_n, N_n)|\!)$, $\hat{M}' =_E (\!|\mathsf{penc}(K, R'_1, N'_1), \ldots \mathsf{penc}(K, R'_n, N'_n)|\!)$ for some terms $N_1, N'_1, R_1, R_1, \ldots, N_n, N'_n, R_n, R_n$ and integer $n \in \mathbb{N}$ such that there exists a permutation χ defined over $\{1, \ldots, n\}$, where for all $1 \leq i \leq n$ we have $N_i =_E N'_{\chi(i)}$. The relation $\overset{\bullet}{\simeq}$ is trivially seen to be an equivalence relation. Moreover, if $\hat{M} \overset{\bullet}{\simeq} \hat{N}$ and $\hat{M} \simeq \hat{M}'$, then $\hat{M}' \overset{\bullet}{\simeq} \hat{N}$.

Lemma A.1. *Given terms L, M, N, if $\mathsf{pet}(L, M, N) =_E \mathsf{true}$, then $M \overset{\bullet}{\simeq} N$.*

Lemma A.2. *Given terms $L, \hat{M}, \hat{N}$, if $\mathsf{checkMix}(L, \hat{M}, \hat{N}) =_E \mathsf{true}$, then $\hat{M} \overset{\bullet}{\simeq} \hat{N}$.*

Lemma A.3. *Given terms $L, \hat{M}, \hat{N}$, if $\mathsf{checkMixPair}(L, \hat{M}, \hat{N}) =_E \mathsf{true}$, then $(\!|\pi_i(M_1), \ldots, \pi_i(M_{|\tilde{M}|})|\!) \overset{\bullet}{\simeq} (\!|\pi_i(N_1), \ldots, \pi_i(N_{|\hat{N}|})|\!)$, where $i \in \{1, 2\}$.*

A.2.1 Proof of Theorem 3.3

Proof. Suppose $m \in \mathbb{N}$ and the tests $\Phi^{IV}, \Phi^{UV}_m, \Phi^{EV}_m$ are given above. We will now show for all names $s_1, \ldots, s_n$ that the conditions of Definition 3.7 hold.

(3.1) Suppose C is a context, B is a process and i, j are integers such that $C[\mathsf{VP}^+_{\mathsf{n}}(s_1, \ldots, s_n)]$ $(\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*$ B, $\varphi(B) \equiv \nu\tilde{n}.\sigma$ and $\Phi^{IV}\{^{s_i}/_v, ^{\tilde{r}_i}/_{\tilde{r}}, ^{\pi_i(\hat{w})}/_w\}\sigma \wedge \Phi^{IV}\{^{s_j}/_v, ^{\tilde{r}_j}/_{\tilde{r}}, ^{\pi_j(\hat{w})}/_w\}\sigma$. It follows that $\pi_1(y)\sigma =_E \mathsf{penc}(x_{\mathsf{pk_T}}, r_{m,i}, s_i)\sigma =_E \mathsf{penc}(x_{\mathsf{pk_T}}, r_{m,j}, s_j)\sigma$ and, by inspection of the equational theory, it is the case that $r_{m,i}\sigma = r_{m,j}\sigma$. Since the record variables $r_{m,i}$, $r_{m,j}$ are handles for fresh nonces created by name restriction in the voter process, it follows immediately from $r_{m,i}\sigma = r_{m,j}\sigma$ that $i = j$.

(3.2) We prove a stronger result: namely, for any substitution σ the condition holds. Suppose

$\Phi_m^{UV}\sigma \wedge \Phi_m^{UV}\{^{\hat{v}'}/_{\hat{v}}\}\sigma$ and hence

$$\hat{v}\sigma =_E (\!|\mathsf{dec}(z_{\mathsf{partial},1}, \pi_1(z_{\mathsf{bal},1})), \ldots, \mathsf{dec}(z_{\mathsf{partial},m}, \pi_1(z_{\mathsf{bal},m}))|\!)\sigma =_E \hat{v}'\sigma$$

It follows immediately that $\hat{v}\sigma \simeq \hat{v}'\sigma$.

(3.3) Again, we will show that the condition holds for all substitutions σ. Suppose $\Phi^{IV}\{^{s_i}/_{v}, {}^{\tilde{r}_i}/_{\tilde{r}}, {}^{\pi_i(\hat{w})}/_{w}, {}^{\pi_i(\hat{y})}/_{y}\}\sigma$ holds for $1 \leq i \leq n$ and hence

$$\bigwedge_{1 \leq i \leq n} \pi_1(\pi_i(\hat{y}))\sigma =_E \mathsf{penc}(x_{\mathsf{pk_T}}, r_{m,i}, s_i)\sigma.$$

Moreover, suppose $\Phi_m^{UV}\sigma$ holds and $n = m$, therefore

$$\mathsf{checkMixPair}(z_{\mathsf{mixPairPf}}, (\!|\pi_1(\pi_1(\hat{y})), \pi_2(\pi_1(\hat{y}))|\!), \ldots, (\!|\pi_1(\pi_m(\hat{y})), \pi_2(\pi_m(\hat{y}))|\!), z_{\mathsf{bal},1}, \ldots, z_{\mathsf{bal},m})\sigma =_E \mathsf{true}$$

holds. By inspection of the equational theory we have

$$\pi_1(z_{\mathsf{bal},i})\sigma =_E \mathsf{penc}(x_{\mathsf{pk_T}}, r_{m,\chi(i)} \circ R_i, s_{\chi(i)})\sigma$$

where $1 \leq i \leq n$ for some permutation χ defined over $\{1, \ldots, n\}$ and terms $R_1, \ldots, R_n$ (note $R_1, \ldots, R_n$ appear in $z_{\mathsf{mixPairPf}}\sigma$). By our hypothesis, for all $1 \leq i \leq n$, we have

$$\mathsf{checkPartialPf}(x_{\mathsf{pk_T}}, \pi_1(z_{\mathsf{bal},i}), z_{\mathsf{partial},i}, z_{\mathsf{partialPf},i})\sigma =_E \mathsf{true}$$

and hence $z_{\mathsf{partial},i}\sigma$ is a partial decryption for $\pi_1(z_{\mathsf{bal},i})\sigma$. It follows, for all $1 \leq i \leq n$, that

$$\mathsf{dec}(z_{\mathsf{partial},i}, \pi_1(z_{\mathsf{bal},i}))\sigma =_E s_{\chi(i)}$$

Finally, by hypothesis, we also have

$$(\!|\mathsf{dec}(z_{\mathsf{partial},1}, \pi_1(z_{\mathsf{bal},1})), \ldots, \mathsf{dec}(z_{\mathsf{partial},m}, \pi_1(z_{\mathsf{bal},m}))|\!)\sigma =_E \hat{v}\sigma$$

and hence it follows that $(\!|s_1, \ldots, s_n|\!) \simeq \hat{v}\sigma$.

(3.5) Suppose C is a context and B is a process such that $C[\mathsf{VP}^+_{\mathsf{n}}(s_1, \ldots, s_n)]\ (\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^*$ B, $\varphi(B) \equiv \nu\tilde{n}.\sigma$, and $\Phi^{EV}_m\sigma \wedge \Phi^{EV}_m\{^{x'}/_x \mid x \in X \backslash \hat{y}\}\sigma$. For all $1 \leq i \leq m$, we have $\mathsf{checkBallot}(\pi_3(\pi_i(\hat{y})), \pi_1(\pi_i(\hat{y})), \pi_2(\pi_i(\hat{y})))\sigma =_E \mathsf{true}$ and it follows by inspection of the equational theory that

$$\pi_2(\pi_i(\hat{y}))\sigma =_E \mathsf{penc}(K_i, S_i, M_i)$$

for some terms K_i, S_i, M_i. Since $\mathsf{checkMixPair}(z_{\mathsf{mixPairPf}}, (\!|\pi_1(\pi_1(\hat{y})), \pi_2(\pi_1(\hat{y}))|\!), \ldots,$ $(\!|\pi_1(\pi_m(\hat{y})), \pi_2(\pi_m(\hat{y}))|\!), z_{\mathsf{bal},1}, \ldots, z_{\mathsf{bal},m})\sigma =_E \mathsf{true}$ and $\mathsf{checkMixPair}(z_{\mathsf{mixPairPf}}', (\!|\pi_1(\pi_1(\hat{y})),$ $\pi_2(\pi_1(\hat{y}))|\!), \ldots, (\!|\pi_1(\pi_m(\hat{y})), \pi_2(\pi_m(\hat{y}))|\!), z'_{\mathsf{bal},1}, \ldots, z'_{\mathsf{bal},m})\sigma =_E \mathsf{true}$; it follows by Lemma A.3 and transitivity of $\overset{\bullet}{\simeq}$ that

$$(\!|\pi_2(z_{\mathsf{bal},1}), \ldots, \pi_2(z_{\mathsf{bal},m})|\!)\sigma \overset{\bullet}{\simeq} (\!|\pi_2(z'_{\mathsf{bal},1}), \ldots, \pi_2(z'_{\mathsf{bal},m})|\!)\sigma.$$

Moreover, for all $1 \leq i \leq m$, we have $\mathsf{pet}(z_{\mathsf{petPf},i}, \pi_2(z_{\mathsf{bal},i}), \bar{z}_{\mathsf{cred},i})\sigma =_E \mathsf{true}$ and $\mathsf{pet}(z'_{\mathsf{petPf},i},$ $\pi_2(z'_{\mathsf{bal},i}), \bar{z}'_{\mathsf{cred},i})\sigma =_E \mathsf{true}$; by Lemma A.1 it follows that

$$(\!|\bar{z}_{\mathsf{cred},1}, \ldots, \bar{z}_{\mathsf{cred},m}|\!)\sigma \overset{\bullet}{\simeq} (\!|\bar{z}'_{\mathsf{cred},1}, \ldots, \bar{z}'_{\mathsf{cred},m}|\!)\sigma.$$

We have $(\!|z_{\mathsf{cred},1}, \ldots, z_{\mathsf{cred},m}|\!)\sigma \simeq (\!|\bar{z}_{\mathsf{cred},1}, \ldots, \bar{z}_{\mathsf{cred},m}|\!)\sigma$, $(\!|z'_{\mathsf{cred},1}, \ldots, z'_{\mathsf{cred},m}|\!)\sigma \simeq (\!|\bar{z}'_{\mathsf{cred},1}, \ldots,$ $\bar{z}'_{\mathsf{cred},m}|\!)\sigma$ and hence we trivially derive

$$(\!|z_{\mathsf{cred},1}, \ldots, z_{\mathsf{cred},m}|\!)\sigma \overset{\bullet}{\simeq} (\!|z'_{\mathsf{cred},1}, \ldots, z'_{\mathsf{cred},m}|\!)\sigma.$$

Since $\mathsf{checkMix}(z_{\mathsf{mixPf}}, \mathsf{getmsg}(\pi_1(\hat{w})), \ldots, \mathsf{getmsg}(\pi_m(\hat{w})), z_{\mathsf{cred},1}, \ldots, z_{\mathsf{cred},m})\sigma =_E \mathsf{true}$ and

$\mathsf{checkMix}(z_{\mathsf{mixPf}}{}', \mathsf{getmsg}(\pi_1(\hat{w}')), \ldots, \mathsf{getmsg}(\pi_m(\hat{w}')), z'_{\mathsf{cred},1}, \ldots, z'_{\mathsf{cred},m})\sigma =_E \mathsf{true}$; it follows by Lemma A.2 that

$$(\!|\mathsf{getmsg}(\pi_1(\hat{w})), \ldots, \mathsf{getmsg}(\pi_m(\hat{w}))|\!)\sigma \stackrel{\bullet}{\simeq} (\!|\mathsf{getmsg}(\pi_1(\hat{w}')), \ldots, \mathsf{getmsg}(\pi_m(\hat{w}'))|\!)\sigma.$$

For all $1 \leq i \leq m$, we have $\mathsf{checksign}(x_{spk_R}, \pi_i(\hat{w}))\sigma =_E \mathsf{true}$ and $\mathsf{checksign}(x_{spk_R}, \pi_i(\hat{w}'))\sigma =_E \mathsf{true}$, where $x_{spk_R}\sigma = \mathsf{pk}(ssk_R)$ and $ssk_R \in \tilde{n}$. By inspection of the equational theory it is the case that $\pi_i(\hat{w})\sigma =_E \mathsf{sign}(ssk_R, M_i)\sigma$ and $\pi_i(\hat{w}')\sigma =_E \mathsf{sign}(ssk_R, M'_i)\sigma$ for some terms M_i, M'_i. Furthermore, since, for all $1 \leq i \leq m$, we have $\mathsf{getmsg}(\pi_i(\hat{w}))\sigma =_E M_i$, $\mathsf{getmsg}(\pi_i(\hat{w}'))\sigma =_E M'_i$ and because $(\!|\mathsf{getmsg}(\pi_1(\hat{w})), \ldots, \mathsf{getmsg}(\pi_m(\hat{w}))|\!)\sigma \stackrel{\bullet}{\simeq} (\!|\mathsf{getmsg}(\pi_1(\hat{w}')), \ldots, \mathsf{getmsg}(\pi_m(\hat{w}'))|\!)\sigma$, it follows that $\tilde{M} \stackrel{\bullet}{\simeq} \tilde{M}'$. Now, since the signing key is under restriction and, by inspection of the voting process and its possible outputs, it follows for all $1 \leq i \leq m$ that

$$\mathsf{getmsg}(\pi_i(\hat{w}))\sigma =_E \mathsf{penc}(\mathsf{pk}(sk_R), m''_{\chi(i)}, d_{\chi(i)})$$
$$\mathsf{getmsg}(\pi_i(\hat{w}'))\sigma =_E \mathsf{penc}(\mathsf{pk}(sk_R), m''_{\chi'(i)}, d_{\chi'(i)})$$

where d_i, m''_i are names under restriction in the registrar process R, $x_{pk_R}\sigma =_E \mathsf{pk}(sk_R)$ and χ, χ' are permutations defined over $\{1, \ldots, n\}$. Finally, we have $\mathsf{snd}^m(\hat{w}) =_E \mathsf{snd}^m(\hat{w}') =_E \varnothing$ and hence conclude $\hat{w}\sigma \simeq \hat{w}'\sigma$.

(3.6) Suppose C is a context and B is a process such that $C[\mathsf{VP}^+_{\mathsf{n}}(s_1, \ldots, s_n)]$ $(\rightarrow^* \stackrel{\alpha}{\rightarrow} \rightarrow^*)^*$ B, $\varphi(B) \equiv \nu\tilde{n}.\sigma$, and $\Phi^{EV}_m\sigma \wedge \Phi^{EV}_m\{^{x'}/_x \mid x \in X \backslash \hat{w}\}\sigma$. For all $1 \leq i \leq m$, we have $\mathsf{checksign}(x_{spk_R}, \pi_i(\hat{w}))\sigma =_E \mathsf{true}$, where $x_{spk_R}\sigma = \mathsf{pk}(ssk_R)$ and $ssk_R \in \tilde{n}$. By inspection of the equational theory it is the case that

$$\pi_i(\hat{w})\sigma =_E \mathsf{sign}(ssk_R, M_i)\sigma$$

for some term M_i. Since the signing key is under restriction and, by inspection of the

voting process, it follows that for all $1 \leq i \leq m$ we have

$$M_i =_E \mathsf{penc}(\mathsf{pk}(sk_R), m_i'', d_i)$$

where d_i, m_i'' are names under restriction in the registrar process R and $x_{pk_R}\sigma =_E \mathsf{pk}(sk_R)$. Since $\mathsf{checkMix}(z_{\mathsf{mixPf}}, \mathsf{getmsg}(\pi_1(\hat{w})), \ldots, \mathsf{getmsg}(\pi_m(\hat{w})), z_{\mathsf{cred},1}, \ldots, z_{\mathsf{cred},m})\sigma =_E \mathsf{true}$, $\mathsf{checkMix}(z_{\mathsf{mixPf}}{}', \mathsf{getmsg}(\pi_1(\hat{w})), \ldots, \mathsf{getmsg}(\pi_m(\hat{w})), z'_{\mathsf{cred},1}, \ldots, z'_{\mathsf{cred},m})\sigma =_E \mathsf{true}$ and for all $1 \leq i \leq m$ we have $\mathsf{getmsg}(\pi_i(\hat{w}))\sigma =_E M_i$; it follows that

$$(\!|z_{\mathsf{cred},1}, \ldots, z_{\mathsf{cred},m}|\!)\sigma \overset{\bullet}{\simeq} (\!|z'_{\mathsf{cred},1}, \ldots, z'_{\mathsf{cred},m}|\!)\sigma$$

by Lemma A.2. We have $(\!|z_{\mathsf{cred},1}, \ldots, z_{\mathsf{cred},m}|\!)\sigma \simeq (\!|\bar{z}_{\mathsf{cred},1}, \ldots, \bar{z}_{\mathsf{cred},m}|\!)\sigma$ and $(\!|z'_{\mathsf{cred},1}, \ldots, z'_{\mathsf{cred},m}|\!)\sigma \simeq (\!|\bar{z}'_{\mathsf{cred},1}, \ldots, \bar{z}'_{\mathsf{cred},m}|\!)\sigma$; it trivially follows that

$$(\!|\bar{z}_{\mathsf{cred},1}, \ldots, \bar{z}_{\mathsf{cred},m}|\!)\sigma \overset{\bullet}{\simeq} (\!|\bar{z}'_{\mathsf{cred},1}, \ldots, \bar{z}'_{\mathsf{cred},m}|\!)\sigma.$$

Moreover, for all $1 \leq i \leq m$, we have $\mathsf{pet}(z_{\mathsf{petPf},i}, \pi_2(z_{\mathsf{bal},i}), \bar{z}_{\mathsf{cred},i})\sigma =_E \mathsf{true}$ and $\mathsf{pet}(z'_{\mathsf{petPf},i}, \pi_2(z'_{\mathsf{bal},i}), \bar{z}'_{\mathsf{cred},i})\sigma =_E \mathsf{true}$; hence by Lemma A.1 it follows that

$$(\!|\pi_2(z_{\mathsf{bal},1}), \ldots, \pi_2(z_{\mathsf{bal},m})|\!)\sigma \overset{\bullet}{\simeq} (\!|\pi_2(z'_{\mathsf{bal},1}), \ldots, \pi_2(z'_{\mathsf{bal},m})|\!)\sigma.$$

By $\mathsf{checkMixPair}(z_{\mathsf{mixPairPf}}, (\!|\pi_1(\pi_1(\hat{y})), \pi_2(\pi_1(\hat{y}))|\!), \ldots, (\!|\pi_1(\pi_m(\hat{y})), \pi_2(\pi_m(\hat{y}))|\!), z_{\mathsf{bal},1}, \ldots, z_{\mathsf{bal},m})\sigma =_E \mathsf{true}$, $\mathsf{checkMixPair}(z_{\mathsf{mixPairPf}}{}', (\!|\pi_1(\pi_1(\hat{y}')), \pi_2(\pi_1(\hat{y}'))|\!), \ldots, (\!|\pi_1(\pi_m(\hat{y}')), \pi_2(\pi_m(\hat{y}'))|\!), z'_{\mathsf{bal},1}, \ldots, z'_{\mathsf{bal},m})\sigma =_E \mathsf{true}$ and Lemma A.3 we have

$$(\!|\pi_2(\pi_1(\hat{y})), \ldots, \pi_2(\pi_m(\hat{y}))|\!)\sigma \overset{\bullet}{\simeq} (\!|\pi_2(\pi_1(\hat{y}')), \ldots, \pi_2(\pi_m(\hat{y}'))|\!)\sigma.$$

For all $1 \leq i \leq m$, we have $\mathsf{checkBallot}(\pi_3(\pi_i(\hat{y})), \pi_1(\pi_i(\hat{y})), \pi_2(\pi_i(\hat{y})))\sigma =_E \mathsf{true}$ and $\mathsf{checkBallot}(\pi_3(\pi_i(\hat{y}')), \pi_1(\pi_i(\hat{y}')), \pi_2(\pi_i(\hat{y}')))\sigma =_E \mathsf{true}$. By inspection of the equational

theory and because $(\!|\pi_2(\pi_1(\hat{y})), \ldots, \pi_2(\pi_m(\hat{y}))|\!)\sigma \stackrel{\bullet}{\simeq} (\!|\pi_2(\pi_1(\hat{y}')), \ldots, \pi_2(\pi_m(\hat{y}'))|\!)\sigma \stackrel{\bullet}{\simeq} (\!|\mathsf{penc}(\mathsf{pk}(sk_R), m''_1, d_1), \ldots, \mathsf{penc}(\mathsf{pk}(sk_R), m''_n, d_n)|\!)$ it is the case that

$$\begin{aligned}\pi_3(\pi_i(\hat{y}))\sigma &=_E \mathsf{ballotPf}(PK_{T_i}, R_i, N_i, \mathsf{pk}(sk_R), S_i, d_{\chi(i)})\\ \pi_3(\pi_i(\hat{y}'))\sigma &=_E \mathsf{ballotPf}(PK'_{T_i}, R'_i, N'_i, \mathsf{pk}(sk_R), S'_i, d_{\chi'(i)})\end{aligned}$$

for some terms $PK_{T_i}, R_i, N_i, S_i, PK'_{T_i}, R'_i, N'_i, S'_i$ and permutations χ, χ' defined over $\{1, \ldots, n\}$. Since for all $1 \leq i \leq m$ the name d_i is under restriction in the voting process specification, it follows that

$$\begin{aligned}\pi_3(\pi_i(\hat{y}))\sigma &=_E \mathsf{ballotPf}(\mathsf{pk}(sk_T), m_{\chi(i)}, s_{\chi(i)}, \mathsf{pk}(sk_R), m'_{\chi(i)}, d_{\chi(i)})\\ \pi_3(\pi_i(\hat{y}'))\sigma &=_E \mathsf{ballotPf}(\mathsf{pk}(sk_T), m_{\chi'(i)}, s_{\chi'(i)}, \mathsf{pk}(sk_R), m'_{\chi'(i)}, d_{\chi'(i)})\end{aligned}$$

(that is, $\pi_3(\pi_i(\hat{y}))\sigma$ and $\pi_3(\pi_i(\hat{y}'))\sigma$ are the signature proofs of knowledge output by the voters) and, moreover, by the validity of the proof, we have

$$\begin{aligned}\pi_1(\pi_i(\hat{y}))\sigma &=_E \mathsf{penc}(\mathsf{pk}(sk_T), m_{\chi(i)}, s_{\chi(i)})\\ \pi_1(\pi_i(\hat{y}'))\sigma &=_E \mathsf{penc}(\mathsf{pk}(sk_T), m_{\chi'(i)}, s_{\chi'(i)})\end{aligned}$$

$$\begin{aligned}\pi_2(\pi_i(\hat{y}))\sigma &=_E \mathsf{penc}(\mathsf{pk}(sk_R), m'_{\chi(i)}, d_{\chi(i)})\\ \pi_2(\pi_i(\hat{y}'))\sigma &=_E \mathsf{penc}(\mathsf{pk}(sk_R), m'_{\chi'(i)}, d_{\chi'(i)})\end{aligned}$$

Since $\bigwedge_{i=1}^{m} \pi_4(\hat{y}) =_E \varnothing$ and $\bigwedge_{i=1}^{m} \pi_4(\hat{y}') =_E \varnothing$, it follows for all $1 \leq i \leq m$ that $\pi_i(\hat{y}) =_E \pi_i(\hat{y})$. Finally, $\mathsf{snd}^m(\hat{y}) =_E \mathsf{snd}^m(\hat{y}') =_E \varnothing$ and hence we conclude $\hat{y}\sigma \simeq \hat{y}'\sigma$.

(3.7) Suppose C is a context and B is a process such that $C[\mathsf{VP}^+_{\mathsf{n}}(s_1, \ldots, s_n)]$ $(\rightarrow^* \stackrel{\alpha}{\rightarrow} \rightarrow^*)^*$ B, $\varphi(B) \equiv \nu\tilde{n}.\sigma$, and $\bigwedge_{1\leq i\leq n} \Phi_i^{IV}\{^{\pi_i(\hat{y})}/_{y}\}\sigma \wedge \Phi_n^{EV}\{^{\hat{w}'}/_{\hat{w}}\}\sigma \wedge \mathsf{snd}^n(\hat{w})\sigma =_E \varnothing$ holds. For all $1 \leq i \leq n$, we have $\pi_i(\hat{w})\sigma = \pi_2(r_{cred_i})\sigma$. Since $\mathsf{snd}^n(\hat{w})\sigma =_E \varnothing$ and, by inspection of the voting process, we have

$$\hat{w}\sigma =_E (\mathsf{sign}(ssk_R, \mathsf{penc}(\mathsf{pk}(sk_R), m''_1, d_1)), \ldots, \mathsf{sign}(ssk_R, \mathsf{penc}(\mathsf{pk}(sk_R), m''_n, d_n)))$$

In addition, we have $\pi_2(\pi_i(\hat{y}))\sigma =_E \mathsf{penc}(\mathsf{pk}(sk_R), m'_i, d_i)$ for all $1 \leq i \leq n$ and by similar reasoning to the above (see Condition 3.5) we derive $\hat{w}\sigma \simeq \hat{w}'\sigma$.

(3.8) This can be witnessed by modelling the complete JCJ-Civitas protocol as the context $C[\_]$. □

B

Violating anonymity in the RSA-based DAA specification

This appendix recalls the mathematical description of the RSA-based Direct Anonymous Attestation protocol [BCC04] and demonstrates that the specification does not satisfy privacy.

B.1 Primitives and building blocks

This section recalls the mathematical primitives which form the building blocks of RSA-based DAA. First we introduce some notation.

Notation. The binary string of length l is denoted $\{0,1\}^l$. Concatenation of binary strings α and β is written $\alpha\|\beta$. The u least significant bits of the binary string α are $LSB_u(\alpha) = \alpha - 2^u\lfloor\frac{\alpha}{2^u}\rfloor$ and the u most significant bits of the binary string α are $MSB_u(\alpha) = \lfloor\frac{\alpha}{2^u}\rfloor$. It should be noted that $\alpha = MSB_u(\alpha)\|LSB_u(\alpha) = 2^u MSB_u(\alpha) + LSB_u(\alpha)$.

B.1.1 Protocols to prove knowledge

This section summarises the proofs of knowledge used by RSA-based DAA; the concrete algorithms can be found in the original papers. Since the RSA-based DAA protocol applies proofs of knowledge to the group of quadratic residues modulo a safe prime product,

the prover must demonstrate that elements are indeed quadratic residues, because the verifier is unable to do so. The prover is therefore required to show that the square root of the element exists; this can be achieved by executing $PK\{(\alpha) : y^2 = (g^2)^\alpha\}$ or $PK\{(\alpha) : y = \pm g^\alpha\}$ instead of $PK\{(\alpha) : y = g^\alpha\}$, such that $\alpha = \log_{g^2} y^2$, which is equivalent to $\alpha = \log_g y$ in the case where $y \in QR_n$ [BCC04].

Demonstrating possession of a discrete logarithm. A proof of knowledge of an element $y \in G$ with respect to base $g \in G$ is denoted $PK\{(\alpha) : y = g^\alpha\}$ [CEG88, CEGP87]. Furthermore, it can be generalised to prove knowledge of $y \in G$ with respect to several bases $g_0, \ldots, g_v \in G$, as denoted by $PK\{(\alpha_0, \ldots, \alpha_v) : y = g_0^{\alpha_0} \cdots g_v^{\alpha_v}\}$.

A proof of knowledge of a discreet logarithm $y \in G$ with respect to bases $g \in G$ such that $\alpha \in \pm\{0,1\}^l$ is denoted $PK\{(\alpha) : y = g^\alpha \wedge (-2^l < \alpha < 2^l)\}$ [CM98b, CM98a, CM99, BCDG88]. To enable the prover to successfully complete the protocol, it is necessary to use a tighter bound $\alpha \in \pm\{0,1\}^{(l-2)/l_\phi}$, where l_ϕ controls the statistical zero-knowledge property. Since the protocol uses bit challenges, it is not very efficient. Boudot presents an enhanced solution [Bou00] and Camenisch & Michels [CM98b, CM98a] provide a modification which allows a proof that $(b - 2^l < \alpha < b + 2^l)$ for a fixed offset b.

Proving equality of discrete logarithms. A proof of equality of discreet logarithms of group elements $y_0, y_1 \in G$ with respect to bases $g \in G$ and $h \in G$ (that is, the prover knows α such that $\log_g y_0 \equiv \log_h y_1$) is denoted $PK\{(\alpha) : y_0 = g^\alpha \wedge y_0 = h^\alpha\}$ [CP94, CP93, Cha90]. Generalisations to prove equalities among $y_0, \ldots, y_v \in G$ to bases $g_0, \ldots, g_v \in G$ are trivial [CS97b].

Proving equality of discrete logarithms in different groups. A proof of equality of discrete logarithms $y_0, y_1 \in G_0$ to the bases $g_0 \in G_0$ and $g_1 \in G_1$, where G_0 and G_1 are *different* groups can be constructed as follows. Let the orders of the groups be q_0, respectively q_1, and let l be an integer such that $2^{l+1} < \min(q_0, q_1)$. The prover can convince the verifier that $\log_{g_0} y_0 \equiv \log_{g_1} y_1$ if α is an element in the tighter range

$\{0,1\}^{(l-2)/l_\phi}$ by executing $PK\{(\alpha) : y_0 \stackrel{G_0}{=} g_0^\alpha \wedge y_1 \stackrel{G_1}{=} g_1^\alpha \wedge (-2^l < \alpha < 2^l)\}$. The prover and verifier engage in an initial setup during which the prover commits to $\tilde{y} = g^\beta h^\alpha \pmod n$, where $G = \langle g \rangle = \langle h \rangle$ – the order of which is unknown (to the verifier) – and $\beta \in_R G$. The prover then carries out $PK\{(\alpha, \beta) : y_0 \stackrel{G_0}{=} g_0^\alpha \wedge y_1 \stackrel{G_1}{=} g_1^\alpha \wedge \tilde{y} \stackrel{G}{=} g^\alpha h^\beta \wedge (-2^l < \alpha < 2^l)\}$ in collaboration with the verifier [CM99].

B.1.2 Camenisch-Lysyanskaya signature scheme

RSA-based DAA is based upon the Camenisch-Lysyanskaya (CL) signature scheme [CL03, Lys02] which is secure under the strong RSA assumption. A summary of the scheme for signing blocks of L messages $m_0, \ldots, m_{L-1}$, and a variant for signatures on committed values, are presented below.

Key generation. On input length l_n of the special RSA modulus, choose safe primes p and q of length $\lceil \frac{l_n}{2} \rceil$. Let $n = pq$ and select $R_0, \ldots, R_{L-1}, S, Z \in_R QR_n$.

Message space. The message space is the set $\{(m_0, \ldots, m_{L-1}) : m_i \in \pm\{0,1\}^{l_m}\}$, where l_m is a parameter and L is the number of blocks.

Signature scheme for blocks. On input message blocks $m_0, \ldots, m_{L-1}$, choose a random prime e of length $l_e \geq l_m + 1$ and select a random number v of length $l_v > l_n + l_m + l_r$, where l_r is a security parameter. Compute A such that $Z \equiv R_0^{m_0} \cdots R_{L-1}^{m_{L-1}} S^v A^e \pmod n$. The signature on blocks $(m_0, \ldots, m_{L-1})$ is defined as (A, e, v).

Signature scheme for committed values. On input message blocks $m_0, \ldots, m_{L-1}$, select $v' \in_R \{0,1\}^{l_n + l_\phi}$ and compute the commitment $U = R_0^{m_0} \cdots R_{L-1}^{m_{L-1}} S^{v'} \pmod n$, where l_ϕ is a security parameter. Send the U to the signer. On receipt of U the signer chooses a random prime e of length $l_e \geq l_m + 1$, selects $v'' \in_R [2^{l_v - 1}, 2^{l_v} - 1]$ and computes A such that $Z \equiv U S^{v''} A^e \pmod n$. The signature (A, e, v'') on the commitment U is

returned. The signature on the block of messages $m_0, \ldots, m_{L-1}$ is $(A, e, v = v' + v'')$. In order to keep $m_0, \ldots, m_{L-1}$ secret, v must remain secret, while A and e can be public.

Verification algorithm. To verify that the tuple (A, e, v) is indeed a signature on the block of messages $m_0, \ldots, m_{L-1}$, check that $Z \equiv R_0^{m_0} \cdots R_{L-1}^{m_{L-1}} S^v A^e \pmod{n}$ and ensure $2^{l_e} > e > 2^{l_e - 1}$.

B.2 Protocol description

RSA-based DAA [BCC04] was the first concrete Direct Anonymous Attestation scheme, and is of particular significance because support is mandated in the TPM specification version 1.2 which has been implemented and deployed in over 300 million computers. This section presents a detailed protocol description.

B.2.1 Security parameters

The security parameters $l_n, l_f, l_v, l_e, l'_e, l_\phi, l_r, l_H, l_\Gamma, l_\rho$ are defined. The purpose of each will now be discussed. The number in parentheses represents the proposed values of these parameters, which have been adopted from the original schema [BCC04]. The parameter l_n (2048) is the size of the RSA modulus and l_f (104) is the size of the TPM's secret $\mathtt{tsk} = (f_0, f_1)$ values. The size of the random v part of the certificate is specified by l_v (2536); the size of prime e is l_e (368); and l'_e (120) is the size of the interval from which the e's are selected. l_ϕ (80) controls the statistical zero-knowledge property, l_r (80) is needed for the reduction in the proof of security and l_H (160) is the length of the output from the hash function used for the Fiat-Shamir heuristic. The parameter l_Γ is the size of the modulus Γ and finally, l_ρ is the size of the order ρ of the subgroup of $\mathbb{Z}_\Gamma^*$ that is used for rogue tagging. The scheme requires that: $l_e > l_\phi + l_H + max(l_f + 4, l'_e + 2)$, $l_v > l_n + l_\phi + l_H + max(l_f + l_r + 3, l_\phi + 2)$ and $l_\rho = 2l_f$. Finally, let $H(\cdot)$ and $H_\Gamma(\cdot)$ be two collision resistant hash functions such that $H(\cdot) : \{0,1\}^* \rightarrow \{0,1\}^{l_H}$ and $H_\Gamma(\cdot)$:

$\{0,1\}^* \rightarrow \{0,1\}^{l_\Gamma + l_\phi}$. It should be noted that collision resistant hash functions exist under the strong RSA assumption.

B.2.2 Setup algorithm

The description of how an issuer creates a key pair, and a non-interactive proof that the key values are correctly formed, are shown below (adapted from [BCC04]). The latter provides an assurance to the host that privacy requirements will be preserved.

1. Choose a special RSA modulus $n = pq$ with $p = 2p' + 1$ and $q = 2q' + 1$, where p, p', q, q' are all primes and n has l_n bits.

2. Select a random generator g' of QR_n.

3. Choose $x_g, x_h, x_s, x_z, x_0, x_1 \in_R [1, \phi(n)]$ and compute:

$$g = g'^{x_g} \bmod n \qquad h = g'^{x_h} \bmod n \qquad S = h^{x_s} \bmod n$$
$$Z = h^{x_z} \bmod n \qquad R_0 = S^{x_0} \bmod n \qquad R_1 = S^{x_1} \bmod n$$

4. Generate a group of prime order. Pick random primes ρ and Γ such that $\Gamma = r\rho + 1$ for some r with $\rho \nmid r$, $2^{l_\Gamma - 1} < \Gamma < 2^{l_\Gamma}$ and $2^{l_\rho - 1} < \rho < 2^{l_\rho}$. Select $\gamma \in_R \mathbb{Z}^*_\Gamma$, where $\gamma^{(\Gamma-1)/\rho} \not\equiv 1 \pmod{\Gamma}$.

5. Produce a non-interactive *proof* that g, h, S, Z, R_0, R_1 are computed correctly, that is, $g, h \in \langle g' \rangle$, $S, Z \in \langle h \rangle$ and $R_0, R_1 \in \langle S \rangle$.

   (a) Choose randoms

$$\tilde{x}_{(g,1)}, \ldots, \tilde{x}_{(g,l_H)} \in_R [1, \phi(n)] \qquad \tilde{x}_{(h,1)}, \ldots, \tilde{x}_{(h,l_H)} \in_R [1, \phi(n)]$$
$$\tilde{x}_{(s,1)}, \ldots, \tilde{x}_{(s,l_H)} \in_R [1, \phi(n)] \qquad \tilde{x}_{(z,1)}, \ldots, \tilde{x}_{(z,l_H)} \in_R [1, \phi(n)]$$
$$\tilde{x}_{(0,1)}, \ldots, \tilde{x}_{(0,l_H)} \in_R [1, \phi(n)] \qquad \tilde{x}_{(1,1)}, \ldots, \tilde{x}_{(1,l_H)} \in_R [1, \phi(n)]$$

(b) Compute *for* $i = 1$ *to* l_H

$$\tilde{g}_{(g,i)} = g'^{\tilde{x}_{(g,i)}} \bmod n \qquad \tilde{h}_{(h,i)} = g'^{\tilde{x}_{(h,i)}} \bmod n$$
$$\tilde{S}_{(s,i)} = h^{\tilde{x}_{(s,i)}} \bmod n \qquad \tilde{Z}_{(z,i)} = h^{\tilde{x}_{(z,i)}} \bmod n$$
$$\tilde{R}_{(0,i)} = S^{\tilde{x}_{(0,i)}} \bmod n \qquad \tilde{R}_{(1,i)} = S^{\tilde{x}_{(1,i)}} \bmod n$$

(c) Compute

$$\begin{aligned} c = H(n\|g'\|g\|h\|S\|Z\|R_0\|R_1\|\tilde{g}_{(g,1)}\|\ldots\|\tilde{g}_{(g,l_H)}\| \\ \tilde{h}_{(h,1)}\|\ldots\|\tilde{h}_{(h,l_H)}\|\tilde{S}_{(s,1)}\|\ldots\|\tilde{S}_{(s,l_H)}\|\tilde{Z}_{(z,1)}\|\ldots\|\tilde{Z}_{(z,l_H)}\| \\ \tilde{R}_{(0,1)}\|\ldots\|\tilde{R}_{(0,l_H)}\|\tilde{R}_{(1,1)}\|\ldots\|\tilde{R}_{(1,l_H)}) \end{aligned}$$

(d) Compute *for* $i = 1$ *to* l_H, where c_i is the ith bit of c

$$\hat{x}_{(g,i)} = \tilde{x}_{(g,i)} - c_i x_g \bmod \phi(n) \qquad \hat{x}_{(h,i)} = \tilde{x}_{(h,i)} - c_i x_h \bmod \phi(n)$$
$$\hat{x}_{(s,i)} = \tilde{x}_{(s,i)} - c_i x_s \bmod \phi(n) \qquad \hat{x}_{(z,i)} = \tilde{x}_{(z,i)} - c_i x_z \bmod \phi(n)$$
$$\hat{x}_{(0,i)} = \tilde{x}_{(0,i)} - c_i x_0 \bmod \phi(n) \qquad \hat{x}_{(1,i)} = \tilde{x}_{(1,i)} - c_i x_1 \bmod \phi(n)$$

(e) Let

$$\begin{aligned} \texttt{proof} = (c, \hat{x}_{(g,1)}, \ldots, \hat{x}_{(g,l_H)}, \hat{x}_{(h,1)}, \ldots, \hat{x}_{(h,l_H)}, \hat{x}_{(s,1)}, \ldots, \hat{x}_{(s,l_H)}, \\ \hat{x}_{(z,1)}, \ldots, \hat{x}_{(z,l_H)}, \hat{x}_{(0,1)}, \ldots, \hat{x}_{(0,l_H)}, \hat{x}_{(1,1)}, \ldots, \hat{x}_{(1,l_H)}) \end{aligned}$$

6. Finally, the public key $(n, g', g, h, S, Z, R_0, R_1, \gamma, \Gamma, \rho)$ and `proof` are published. The private key values p', q' are stored by the issuer.

Verification of the Issuer's public key

A valid public key is essential to ensure the privacy of the host. Incorrectly formed g, h, S, Z, R_0, R_1 values could potentially break this property. Furthermore, if γ does not generate the subgroup $\mathbb{Z}_\Gamma^*$ the issuer can link transactions (signatures presented to the verifier). The requirement that n is a special RSA modulus ensures that Step 6 of the join algorithm is zero-knowledge. This is of concern to the issuer and is not important to the security of the host. Verification of an issuer's public key is shown below (adapted from [BCC04]). Note that this verification need only be performed once and not necessarily by every user of a system (that is, in a multi-user environment, it is sufficient for a single user to perform the verification).

1. Verify the `proof`, that is check $g, h \in \langle g' \rangle$, $S, Z \in \langle h \rangle$ and $R_0, R_1 \in \langle S \rangle$. On input:

$$\texttt{proof} = (c, \hat{x}_{(g,1)}, \ldots, \hat{x}_{(g,l_H)}, \hat{x}_{(h,1)}, \ldots, \hat{x}_{(h,l_H)}, \hat{x}_{(s,1)}, \ldots, \hat{x}_{(s,l_H)}, \\ \hat{x}_{(z,1)}, \ldots, \hat{x}_{(z,l_H)}, \hat{x}_{(0,1)}, \ldots, \hat{x}_{(0,l_H)}, \hat{x}_{(1,1)}, \ldots, \hat{x}_{(1,l_H)})$$

   (a) Compute *for* $i = 1$ *to* l_H, where c_i is the ith bit of c

$$\hat{g}_{(g,i)} = g^{c_i} g'^{\hat{x}_{(g,i)}} \bmod n \qquad \hat{h}_{(h,i)} = h^{c_i} g'^{\hat{x}_{(h,i)}} \bmod n$$
$$\hat{S}_{(s,i)} = S^{c_i} h^{\hat{x}_{(s,i)}} \bmod n \qquad \hat{Z}_{(z,i)} = Z^{c_i} h^{\hat{x}_{(z,i)}} \bmod n$$
$$\hat{R}_{(0,i)} = R_0^{c_i} S^{\hat{x}_{(0,i)}} \bmod n \qquad \hat{R}_{(1,i)} = R_1^{c_i} S^{\hat{x}_{(1,i)}} \bmod n$$

   (b) Verify

$$c \stackrel{?}{=} H(n \| g' \| g \| h \| S \| Z \| R_0 \| R_1 \| \hat{g}_{(g,1)} \| \ldots \| \hat{g}_{(g,l_H)} \| \\ \hat{h}_{(h,1)} \| \ldots \| \hat{h}_{(h,l_H)} \| \hat{S}_{(s,1)} \| \ldots \| \hat{S}_{(s,l_H)} \| \hat{Z}_{(z,1)} \| \ldots \| \hat{Z}_{(z,l_H)} \| \\ \hat{R}_{(0,1)} \| \ldots \| \hat{R}_{(0,l_H)} \| \hat{R}_{(1,1)} \| \ldots \| \hat{R}_{(1,l_H)})$$

2. Check that Γ and ρ are primes. Ensure $\rho \mid (\Gamma - 1)$, $\rho \nmid \frac{\Gamma-1}{\rho}$ and $\gamma^{\rho} \equiv 1 \pmod{\Gamma}$.

3. Check that all public key values have the required length.

B.2.3 Join algorithm

The purpose of the join algorithm is to enable a host/TPM to acquire a CL-signature on a secret `tsk` value which can later be used as an anonymous attestation identity credential. Let $PK_I = (n, g', g, h, S, Z, R_0, R_1, \gamma, \Gamma, \rho)$ be the public key of the issuer and PK'_I the long term public key of the issuer used to authenticate PK_I. The value $\texttt{bsn}_I$ is a unique basename assigned to each issuer and `cnt` is an internal counter stored within the TPM. The counter records the number of times the TPM has executed the join algorithm. Prior to executing the join algorithm, the host is assumed to verify that PK_I is authenticated by PK'_I.

The TPM computes `tsk` using the issuer's public key, its secret seed `DAASeed` and counter value `cnt`. Computing `tsk` from the secret seed `DAASeed` – as opposed to a random nonce – reduces the computational and storage requirements of the TPM. The counter value allows the TPM to obtain different DAA keys using the same `DAASeed`; alternatively, the TPM is allowed to re-run the join algorithm using the same `cnt` value. The TPM splits the `tsk` value into two l_f bit messages; the pair (f_0, f_1) allows computation of smaller exponentials and permits the use of a smaller prime e. The TPM commits to the message pair (f_0, f_1), that is, the TPM derives $U = R_0^{f_0} R_1^{f_1} S^{v'} \pmod{n}$, where v' is a random commitment factor. The TPM also computes $N_I = \zeta_I^{f_0 + f_1 2^{l_f}} \pmod{\Gamma}$ for rogue tagging purposes. The TPM forwards U, N_I to the host who sends them to the issuer. The TPM and issuer establish a one-way authenticated channel to assure the issuer of the origin of U. The issuer checks whether `tsk` stems from a rogue TPM or if N_I has been used too many times previously, in which case it aborts. The platform convinces the issuer that U, N_I are correctly formed, and that the f_i's are of the appropriate lengths. To grant a certificate, the issuer executes the CL-signature protocol and sends the platform (A, e, v'').

The issuer also provides a signature proof of knowledge (Step 7) that $A \in \langle h \rangle$. The host verifies the proof and is assured that A can be statistically hidden in $\langle h \rangle$, preventing an adversarial issuer from violating privacy. Note that an adversarial issuer could compute $A = b\left(\frac{Z}{US^{v''}}\right)^{1/e} \pmod{n}$, where $b^e = 1$ and $b \notin \langle h \rangle$. Since the sign algorithm contains $T = AS^w$ for some random w, the adversarial would be able to link T to A (by testing $T \in \langle h \rangle$) and thus violate privacy [BCC04]. The pair (A, e) is stored by the host and can be publicly known. The host forwards v'' to the TPM; the TPM recovers $v = v' + v''$, that is, a signature on the TPM's secret $\mathtt{tsk} = (f_0, f_1)$. The explicit details of the join algorithm are provided below (adapted from [BCC04]).

1. The host computes $\zeta_I = (H_\Gamma(0 \| \mathtt{bsn}_I))^{(\Gamma-1)/\rho} \pmod{\Gamma}$ and sends ζ_I to the TPM.

2. The TPM checks whether $\zeta_I^\rho \stackrel{?}{\equiv} 1 \pmod{\Gamma}$. Let $i = \lfloor \frac{l_\rho + l_\phi}{l_H} \rfloor$ ($i = 1$ for the parameters specified in Section B.2.1). The TPM computes:

$$\mathtt{tsk} = H\Big(H\big(\mathtt{DAASeed} \| H(PK'_I)\big) \| \mathtt{cnt} \| 0\Big) \| \ldots \| H\Big(H\big(\mathtt{DAASeed} \| H(PK'_I)\big) \| \mathtt{cnt} \| i\Big) \pmod{\rho},$$
$$f_0 = LSB_{l_f}(\mathtt{tsk}), \quad f_1 = MSB_{l_f}(\mathtt{tsk}), \quad v' \in_R \{0,1\}^{l_n + l_\phi},$$
$$U = R_0^{f_0} R_1^{f_1} S^{v'} \pmod{n}, \quad N_I = \zeta_I^{f_0 + f_1 2^{l_f}} \pmod{\Gamma}$$

   The TPM forwards U and N_I to the host who sends them to the issuer.

3. The issuer checks that the U value stems from the TPM that owns a given public endorsement key (PK_{ek}):

   (a) The issuer chooses $n_e \in_R \{0,1\}^{l_\phi}$, encrypts n_e with PK_{ek} and sends the encryption to the TPM.

   (b) The TPM decrypts the value, revealing n_e, computes $a_U = H(U \| n_e)$ and returns a_U to the issuer.

(c) The issuer checks $a_U \stackrel{?}{=} H(U\|n_e)$.

4. The issuer ensures for all $(\tilde{f}_0, \tilde{f}_1)$ on the rogue list $N_I \stackrel{?}{\neq} \zeta_I^{\tilde{f}_0+\tilde{f}_1 2^{l_f}} \pmod{\Gamma}$. The issuer also checks that the N_I has not been used too many times. If the issuer finds the platform to be rogue, it aborts.

5. The platform proves knowledge of f_0, f_1 and v'. It executes:

$$SPK\{(f_0, f_1, v') : \quad U \equiv R_0^{f_0} R_1^{f_1} S^{v'} \pmod{n} \quad \wedge \quad N_I \equiv \zeta_I^{f_0+f_1 2^{l_f}} \pmod{\Gamma}$$
$$\wedge\, f_0, f_1 \in \{0,1\}^{l_f+l_\phi+l_H+2} \quad \wedge \quad v' \in \{0,1\}^{l_n+l_\phi+l_h+2}\}(n_t\|n_i)$$

(a) The TPM chooses $r_{f_0}, r_{f_1} \in_R \{0,1\}^{l_f+l_\phi+l_H}$ and $r_{v'} \in_R \{0,1\}^{l_n+l_\phi+l_H+2}$. It computes $\tilde{U} = R_0^{r_{f_0}} R_1^{r_{f_1}} S^{r_{v'}} \pmod{n}$, $\tilde{N}_I = \zeta_I^{r_{f_0}+r_{f_1}+2^{l_f}} \pmod{\Gamma}$ and sends $\tilde{U}, \tilde{N}_I$ to the host.

(b) The issuer selects $n_i \in_R \{0,1\}^{l_H}$ and sends it to the host.

(c) The host computes $c_h = H(n\|R_0\|R_1\|S\|U\|N_I\|\tilde{U}\|\tilde{N}_I\|n_i)$ and sends c_h to the TPM.

(d) The TPM picks $n_t \in_R \{0,1\}^{l_\phi}$, computes $c = H(c_h\|n_t)$, and calculates $s_{f_0} = r_{f_0}+c\cdot f_0$, $s_{f_1} = r_{f_1}+c\cdot f_1$ and $s_{v'} = r_{v'}+c\cdot v'$. The TPM sends $(c, n_t, s_{f_0}, s_{f_1}, s_{v'})$ to the host, who forwards to the issuer.

(e) The issuer computes:

$$\hat{U} = \pm U^{-c} R_0^{s_{f_0}} R_1^{s_{f_1}} S^{s_{v'}} \pmod{n} \text{ and } \hat{N}_I = N_I^{-c} \zeta_I^{s_{f_0}+s_{f_1} 2^{l_f}} \pmod{\Gamma}$$

and verifies the proof by checking:

$$c \stackrel{?}{=} H\big(H(n\|R_0\|R_1\|S\|U\|N_I\|\tilde{U}\|\tilde{N}_I\|n_i)\|n_t\big),$$
$$s_{f_0}, s_{f_1} \stackrel{?}{\in} \{0,1\}^{l_f+l_\phi+l_H+1} \quad \text{and} \quad s_{v'} \stackrel{?}{\in} \{0,1\}^{l_n+2l_\phi+l_H+1}$$

6. The issuer chooses $v'' \in_R [2^{l_v}, 2^{l_v}-1]$, a prime $e \in_R [2^{l_e-1}, 2^{l_e-1}+2^{l'_e-1}]$ and computes $Z = US^{v''}A^e \pmod{n}$. That is, the issuer produces a CL-signature on U.

7. To convince the host that A was correctly formed, the issuer runs the protocol:

$$SPK\{(d) : A \equiv \pm \left(\frac{Z}{US^{v''}}\right)^d \pmod{n}\}(n_h)$$

   (a) The host selects $n_h \in_R \{0,1\}^{l_\phi}$ and sends n_h to the issuer.

   (b) The issuer chooses $r_e \in_R [0, \phi(n)]$ and computes:

$$\tilde{A} = \left(\frac{Z}{US^{v''}}\right)^{r_e} \pmod{n},$$
$$c' = H(n\|Z\|S\|U\|v''\|A\|\tilde{A}\|n_h), \quad s_e = r_e - c'/e \mod \phi(n)$$

   and sends c', s_e and (A, e, v'') to the host.

   (c) The host verifies that e is prime and $e \in [2^{l_e-1}, 2^{l_e-1} + 2^{l'_e-1}]$, computes:

$$\hat{A} = A^{c'}\left(\frac{Z}{US^{v''}}\right)^{s_e} \pmod{n}$$

   and checks that:

$$c' \stackrel{?}{=} H(n\|Z\|S\|U\|v''\|A\|\hat{A}\|n_h)$$

   (d) The host forwards v'' to the TPM, which in turn stores $v = v'' + v'$.

B.2.4 Sign algorithm

The sign algorithm enables the host to generate a signature proof of knowledge of attestation on a message m. Intuitively, if a verifier is presented with such a proof, it is convinced that it is communicating with a trusted platform and the message is genuine. In addition, the host must convince the verifier that it is not rogue. The message m may

be either a public part of an Attestation Identity Key (AIK) produced by the TPM or an arbitrary message. If m is an AIK, the key can later be used to sign PCR data or to certify a non-migratable key. Where m is an arbitrary message its purpose is application dependent. It may, for example, be a session key. To distinguish between these two modes of operation, a variable b is defined. When $b = 0$, the message was generated by the TPM and when $b = 1$, the message was input to the TPM. Let $n_v \in \{0,1\}^{l_H}$ be a nonce generated by the verifier and `bsn` the verifier's basename. The algorithm is described below (adapted from [BCC04, BCC05]), as a result of which the signature $\sigma = (\zeta, T, N_V, c, n_t, s_{\bar{v}}, s_{f_0}, s_{f_1}, s_e)$ will be produced. Many of the secrets involved in the process are actually known to the host. In fact only f_0, f_1, v need to remain secret to the TPM.

1. (a) Depending on whether linkability is desirable, the host computes ζ as follows:

$$\zeta \in_R \langle\gamma\rangle \quad \text{or} \quad \zeta = (H_\Gamma(0\|\texttt{bsn}))^{(\Gamma-1)/\rho} \pmod{\Gamma}$$

   and sends ζ to the TPM.

   (b) The TPM checks $\zeta \stackrel{?}{\in} \langle\gamma\rangle$; that is, it verifies $\zeta^\rho \stackrel{?}{\equiv} 1 \pmod{\Gamma}$.

2. (a) The host picks $w \in_R \{0,1\}^{l_n+l_\phi}$ and computes $T = AS^w \pmod{n}$.

   (b) The TPM computes $N_V = \zeta^{f_0+f_1 2^{l_f}} \pmod{\Gamma}$ and sends N_V to the host.

3. The platform produces a signature of knowledge that T commits to an attestation certificate and N_V was computed using the same secret value f_0, f_1:

$$SPK\{(f_0, f_1, \bar{v}, e) : Z \equiv \pm T^e R_0^{f_0} R_1^{f_1} S^{\bar{v}} \pmod{n} \quad \wedge \quad N_V \equiv \pm\zeta^{f_0+f_1 2^{l_f}} \pmod{\Gamma}$$
$$\wedge \quad f_0, f_1 \in \{0,1\}^{l_f+l_\phi+l_H+2} \quad \wedge \quad (e - 2^{l_e}) \in \{0,1\}^{l'_e+l_\phi+l_H+1}\}(n_t\|n_v\|b\|m)$$

   (a) i. The TPM chooses random integers $r_v \in_R \{0,1\}^{l_v+l_\phi+l_H}$ and

$r_{f_0}, r_{f_1} \in_R \{0,1\}^{l_f+l_\phi+l_H}$ and computes:

$$\tilde{T}_{1t} = R_0^{r_{f_0}} R_1^{r_{f_1}} S^{r_v} \pmod{n}$$

$$\tilde{r}_f = r_{f_0} + r_{f_1} 2^{l_f} \pmod{\rho} \quad \tilde{N}_V = \zeta^{\tilde{r}_f} \pmod{\Gamma}$$

The TPM sends $\tilde{T}_{1t}$ and $\tilde{N}_V$ to the host[1].

ii. The host selects random integers:

$$r_e \in_R \{0,1\}^{l'_e+l_\phi+l_H} \qquad r_{\bar{v}} \in_R \{0,1\}^{l_e+l_n+2l_\phi+l_H+1}$$

and computes $\tilde{T} = \tilde{T}_t T^{r_e} S^{r_{\bar{v}}} \pmod{n}$.

(b) i. The host computes:

$$c_h = H(n\|g\|g'\|h\|R_0\|R_1\|S\|Z\|\gamma\|\Gamma\|\rho\|\zeta\|T\|N_V\|\tilde{T}\|\tilde{N}_V\|n_v)$$

and sends c_h to the TPM.

ii. The TPM picks $n_t \in_R \{0,1\}^{l_\phi}$, computes $c = H\big(H(c_h\|n_t)\|b\|m\big)$ and sends c, n_t to the host.

(c) i. The TPM computes

$$s_v = r_v + c \cdot v \qquad s_{f_0} = r_{f_0} + c \cdot f_0 \qquad s_{f_1} = r_{f_1} + c \cdot f_1$$

and sends s_v, s_{f_0}, s_{f_1} to the host.

ii. The host computes:

$$s_e = r_e + c \cdot (e - 2^{l_e-1}) \qquad s_{\bar{v}} = s_v + r_v - c \cdot w \cdot e$$

[1]Note that $\zeta^{\tilde{r}_f} \equiv \zeta^{f_0+f_1 2^{l_f}} \pmod{\Gamma}$ and the order of $\rho < \Gamma$ the calculation $\zeta^{\tilde{r}_f} \pmod{\Gamma}$ will involve a smaller exponential thus providing a performance gain.

4. The host outputs the signature $\sigma = (\zeta, T, N_V, c, n_t, s_{\bar{v}}, s_{f_0}, s_{f_1}, s_e)$.

The main difference between the DAA signing algorithm and the signature generation of prior schemes [CL01, Lys02, CL03] is that DAA distributes the computation between the TPM and the host. The TPM only produces a signature proof of knowledge $SPK\{(f_0, f_1, v) : (Z/A^e) \equiv R_0^{f_0} R_1^{f_1} S^v \pmod{n} \wedge N_V \equiv \zeta^{f_0 + f_1 2^{l_f}} \pmod{\Gamma}\}(n_t \| n_v \| b \| m)$, which the host extends to a full DAA signature. Note that the signature produced by the TPM is not anonymous, as the value (Z/A^e) would fully identify the host.

B.2.5 Verification algorithm

The verification algorithm defines the method by which a verifier checks a signature σ on message m with respect to the issuer's public key $(n, g', g, h, S, Z, R_0, R_1, \gamma, \Gamma, \rho)$. The algorithm is described below (adapted from [BCC04, BCC05]).

1. The verifier computes:

$$\hat{T} = Z^{-c} T^{s_e + c2^{l_e - 1}} R_0^{s_{f_0}} R_1^{s_{f_1}} S^{s_{\bar{v}}} \pmod{n} \qquad \hat{N}_V = N_V^{-c} \zeta^{s_{f_0} + s_{f_1}} \pmod{\Gamma}$$

2. The verifier checks:

$$c \stackrel{?}{=} H\Big(H\big(H(n \| g \| g' \| h \| R_0 \| R_1 \| S \| Z \| \gamma \| \Gamma \| \rho \| \zeta \| T \| N_V \| \hat{T} \| \hat{N}_V \| n_v) \| n_t\big) \| b \| m\Big),$$
$$N_V, \zeta \stackrel{?}{\in} \langle \gamma \rangle, \qquad s_{f_0}, s_{f_1} \stackrel{?}{\in} \{0,1\}^{l_f + l_\phi + l_H + 1} \quad \text{and} \quad s_e \stackrel{?}{\in} \{0,1\}^{l'_e + l_\phi + l_H + 1}$$

   Note that the check $N_V, \zeta \stackrel{?}{\in} \langle \gamma \rangle$ can be done by raising N_V and ζ to the order of γ (which is ρ) and verifying the result is one; that is, checking that $N_V^\rho \stackrel{?}{\equiv} 1 \pmod{\Gamma}$ and $\zeta^\rho \stackrel{?}{\equiv} 1 \pmod{\Gamma}$.

3. If ζ was derived from the verifier's basename, check that $\zeta \stackrel{?}{\equiv} (H_\Gamma(0 \| \mathtt{bsn}_I))^{(\Gamma-1)/\rho} \pmod{\Gamma}$.

4. The verifier checks for all $(\tilde{f}_0, \tilde{f}_1)$ on the rogue list that $N_I \stackrel{?}{\neq} \zeta_I^{\tilde{f}_0 + \tilde{f}_1 2^{l_f}} \pmod{\Gamma}$.

B.3 Specification analysis: Violating anonymity

Suppose a trusted platform, with a public endorsement key PK_{ek}, executes the join and sign algorithm using the same basename $\texttt{bsn}$ (such that $\texttt{bsn} \neq \perp$). Further assume that the issuer has a public key $(n, g', g, h, S, Z, R_0, R_1, \gamma, \Gamma, \rho)$, which the trusted platform has verified, and the security parameter l_f is used. During Step 2 of the join algorithm, the platform computes $N_I = \zeta_I^{f_0+f_1 2^{l_f}} \pmod{\Gamma}$, where $\zeta_I = (H_\Gamma(0\|\texttt{bsn}))^{(\Gamma-1)/\rho} \pmod{\Gamma}$, $f_0 = LSB_{l_f}(\texttt{tsk})$ and $f_1 = MSB_{l_f}(\texttt{tsk})$. The value N_I is sent to the issuer. The platform then authenticates using the identity PK_{ek}. During Step 2 of the sign algorithm, the platform computes $N_V = \zeta^{f_0+f_1 2^{l_f}} \pmod{\Gamma}$, where $\zeta = (H_\Gamma(0\|\texttt{bsn}))^{(\Gamma-1)/\rho} \pmod{\Gamma}$, $f_0 = LSB_{l_f}(\texttt{tsk})$ and $f_1 = MSB_{l_f}(\texttt{tsk})$. The value N_V is sent to the verifier. It follows immediately that $N_I = N_V$ and, moreover, N_I can be linked to the identity PK_{ek}; that is, the privacy of the trusted platform is violated.

B.4 Specification solution: Restoring anonymity

The protocol can be fixed by refining Step 1 of the sign algorithm to use

$$\zeta = (H_\Gamma(1\|\texttt{bsn}))^{(\Gamma-1)/\rho} \pmod{\Gamma}$$

when linkability is required. To maintain correctness of the protocol, Step 3 of the verification algorithm must check $\zeta \stackrel{?}{\equiv} (H_\Gamma(1\|\texttt{bsn}_I))^{(\Gamma-1)/\rho} \pmod{\Gamma}$. Appendix C.2 shows that the revised protocol satisfies user-controlled anonymity in the formal model.

C

RSA-based DAA with user-controlled anonymity

Section 4.4.5 presents an analysis of user-controlled anonymity in the RSA-based DAA protocol and discovers a vulnerability. The proof of this result appears in Section C.1. A security fix was presented in Section 4.4.6, and we prove that the solution satisfies user-controlled anonymity in Section C.2.

C.1 Proof of Theorem 4.2

Proof. Let $\mathsf{DAA}_{\mathsf{RSA}}$ be the augmented Direct Anonymous Attestation biprocess derived from the specification $\langle \mathsf{Join}_{\mathsf{RSA}}, \mathsf{Sign}_{\mathsf{RSA}} \rangle$. Let us consider the evaluation context

$$\begin{aligned} C[\_] \;=\; & \overline{c}\langle(\!|\mathsf{pk}(sk_I), \mathtt{bsn}|\!)\rangle. \\ & c(w).c(x).\overline{c}\langle\mathsf{penc}(w, n, n_e)\rangle.c(z_a).\text{if } z_a = \mathsf{hash}(n_e, \pi_2(x)) \text{ then} \\ & \overline{c}\langle n_i\rangle.c(z_s).\overline{c}\langle\mathsf{clsign}(sk_I, e, v'', \pi_2(x))\rangle. \\ & c(w').c(x').\overline{c}\langle\mathsf{penc}(w', n, n_e)\rangle.c(z'_a).\text{if } z'_a = \mathsf{hash}(n_e, \pi_2(x')) \text{ then} \\ & \overline{c}\langle n_i\rangle.c(z'_s).\overline{c}\langle\mathsf{clsign}(sk_I, e, v'', \pi_2(x'))\rangle. \\ & \overline{c}\langle(\!|\mathtt{bsn}, \mathsf{msg}|\!)\rangle.\overline{c}\langle n_v\rangle.c(y).\text{if } \pi_1(x) = \pi_3(y) \text{ then } \overline{b}\langle\mathsf{fail}\rangle \text{ else } 0 \mid \_ \end{aligned}$$

We have

$$\mathrm{fst}(C[\mathsf{DAA}_{\mathsf{RSA}}]) \to^* C'[\text{if } \mathsf{commit}(\mathtt{tsk}, \zeta_I) = \mathsf{commit}(\mathtt{tsk}, \zeta) \text{ then } \overline{b}\langle\mathsf{fail}\rangle \text{ else } 0]$$

and

$$\mathrm{snd}(C[\mathsf{DAA}_{\mathsf{RSA}}]) \rightarrow^* C'[\text{if } \mathsf{commit}(\mathtt{tsk}, \zeta_I) = \mathsf{commit}(\mathtt{tsk}', \zeta) \text{ then } \overline{b}\langle\mathsf{fail}\rangle \text{ else } 0]$$

where

$$\begin{aligned}
\mathtt{tsk} &= \mathsf{hash}(\mathsf{hash}(\mathtt{DAASeed}, \mathsf{hash}(\mathsf{pk}(sk_I))), \mathtt{cnt}, 0) \\
\mathtt{tsk}' &= \mathsf{hash}(\mathsf{hash}(\mathtt{DAASeed}', \mathsf{hash}(\mathsf{pk}(sk_I))), \mathtt{cnt}', 0) \\
\zeta_I &= \mathsf{hash}(0, \mathtt{bsn}) \\
\zeta &= \mathsf{hash}(0, \mathtt{bsn})
\end{aligned}$$

It follows that $\mathrm{fst}(C[\mathsf{DAA}_{\mathsf{RSA}}]) \not\approx \mathrm{snd}(C[\mathsf{DAA}_{\mathsf{RSA}}])$ because $\mathrm{fst}(C[\mathsf{DAA}_{\mathsf{RSA}}])$ can output on channel b, but $\mathrm{snd}(C[\mathsf{DAA}_{\mathsf{RSA}}])$ cannot; and therefore $\mathsf{DAA}_{\mathsf{RSA}}$ does not satisfy user-controlled anonymity. □

C.2 Analysis: Restoring user-controlled anonymity

This section provides the scripts used to automatically verify that the revised RSA-based DAA protocol (Section 4.4.6) satisfies user-controlled anonymity. The free name declarations, function definitions and equations for the RSA-based DAA process specification appear in Listing C.1; the join algorithm $\mathsf{Join}_{\mathsf{RSA}'}$ is presented in Listing C.2, and the sign algorithm $\mathsf{Sign}_{\mathsf{RSA}'}$ appears in Listing C.3. Finally, the augmented Direct Anonymous Attestation biprocess $\mathsf{DAA}_{\mathsf{RSA}'}$ is presented in Listing C.4. (The nonce `XXTERMINATION` is introduced to avoid an over-approximation issue.) ProVerif can be used to automatically verify observational equivalence of Listing C.4 and hence the revised RSA-based DAA protocol satisfies user-controlled anonymity.

Optimised analysis. The ProVerif script in Listing C.4 has an execution time of 6 minutes 43 seconds By removing the name restrictions on the secret part of the endorsement key for each signer, the execution time is reduced to 92 seconds. This optimised script is presented in Listing C.5.

```
fun accept/0.
fun zero/0.
fun one/0.
fun FJoin/0.
fun FSign/0.
fun clgetnonce/1.
fun clgetprime/1.
fun hash/1.
fun pk/1.
fun commit/2.
fun circ/2.
fun dec/2.
fun open/2.
fun checkclsign/3.
fun checkspk/3.
fun clcommit/3.
fun clopen/3.
fun penc/3.
fun spk/3.
fun clsign/4.

equation dec(k,penc(pk(k),r,m)) = m.
equation clgetprime(clsign(xsk,xprime,xrand,xmsg)) = xprime.
equation clgetnonce(clsign(xsk,xprime,xrand,xmsg)) = xrand.
equation checkclsign(pk(xsk),xmsg,clsign(xsk,xprime,xrand,xmsg))
                = accept.
equation open(xrand,commit(xrand,xplain)) = xplain.
equation clopen(x,xrand,clcommit(x,xrand,xplain)) = xplain.
equation clopen(pk(xsk),xrand,clsign(xsk,yprime,yrand,
    clcommit(pk(xsk),xrand,xmsg))) = clsign(xsk,yprime,xrand,xmsg).
equation checkspk(FJoin,(xzeta,xpk,commit(xtsk,xzeta),
                                clcommit(xpk,xv,xtsk),xmsg),
    spk(FJoin,(xtsk,xv),(xzeta,xpk,commit(xtsk,xzeta),
                                clcommit(xpk,xv,xtsk),xmsg))) = accept.
equation checkspk(FSign,(xzeta,pk(xsk),commit(xtsk,xzeta),
        clcommit(pk(xsk),xw,clsign(xsk,xe,xv,xtsk)),xmsg),
    spk(FSign,(xtsk,xw),(xzeta,pk(xsk),commit(xtsk,xzeta),
        clcommit(pk(xsk),xw,clsign(xsk,xe,xv,xtsk)),xmsg))) = accept.
```

Listing C.1: Free name declarations, function definitions and equations for $\mathsf{DAA}_{\mathsf{RSA}'}$

```
let join =
    in(aj,((pkI,bsnI),DAASeed,cnt,skM));

    new v';
    let zetaI = hash((zero,bsnI)) in
    let tsk = hash((hash((DAASeed,hash(pkI))),cnt,zero)) in
    let NI = commit(tsk,zetaI) in
    let U = clcommit(pkI,v',tsk) in
    out(c,(NI,U));

    in(c,encNe);
    let ne = dec(skM,encNe) in
    out(c,hash((U,ne)));

    in(c,ni);
    new nt;
    out(c,(nt,spk(FJoin,(tsk,v'),(zetaI,pkI,NI,U,(nt,ni)))));

    in(c,sig);
    let cre = clopen(pkI,v',sig) in
    if checkclsign(pkI,tsk,cre) = accept then

    out(aj',(cre,tsk)).
```

Listing C.2: ProVerif script modelling $\mathsf{Join}_{\mathsf{RSA}'}$

```
let sign =
    in(as,((pkI,bsnI),bsnV,m,cre,tsk,xxTERMINATION));
    in(c,nv);
    new nt;new w;

    if bsnV = bottom then (
        new zeta;
        let creHat = clcommit(pkI,w,cre) in
        let NV = commit(tsk,zeta) in
        out(as',(zeta,pkI,NV,creHat,nt,
                 spk(FSign,(tsk,w),(zeta,pkI,NV,creHat,(nt,nv,m)))))
    ) else (
        let zeta = hash((one,bsnV)) in
        let creHat = clcommit(pkI,w,cre) in
        let NV = commit(tsk,zeta) in
        out(as',(zeta,pkI,NV,creHat,nt,
                 spk(FSign,(tsk,w),(zeta,pkI,NV,creHat,(nt,nv,m)))))
    ).
```

Listing C.3: ProVerif script modelling $\mathsf{Sign}_{\mathsf{RSA}'}$

```
free c.
free chlP.
free chlM.

fun bottom/0.

let SignerP =
    new aj;new aj';new as;new as';( !join )|( !sign )|(
        new cnt;new DAASeed;new skM;out(c,pk(skM));
        ! out(aj,(wparams,DAASeed,cnt,skM));
        in(aj',(cre,tsk));
        (!
            in(c,(xmsg,=bottom));new XXTERMINATION;
            out(as,(wparams,bottom,xmsg,cre,tsk,XXTERMINATION));
            in(as',z);out(c,z)
        )|(!
            in(c,(xmsg,xbsn));new XXTERMINATION;
            out(as,(wparams,(chlP,xbsn),xmsg,cre,tsk,XXTERMINATION));
            in(as',z);out(c,z)
        )|(
           out(wb,(cre,tsk))
        )
    ).

let Challenge =
    new as;new as';( sign ) | (
        in(bA,(creA,tskA));
        in(bB,(creB,tskB));
        let cre = choice[creA,creB] in
        let tsk = choice[tskA,tskB] in
        (
            in(c,(xmsg,=bottom));new XXTERMINATION;
            out(as,(wparams,bottom,xmsg,cre,tsk,XXTERMINATION));
            in(as',x);out(c,x)
        )|(
            in(c,(xmsg,xbsn));new XXTERMINATION;
            out(as,(wparams,(chlM,xbsn),xmsg,cre,tsk,XXTERMINATION));
            in(as',x);out(c,x)
        )
    ).

process
    in(c,wparams);new bA;new bB;
    (let wb = bA in SignerP)|
    (let wb = bB in SignerP)|
    (Challenge)
```

Listing C.4: ProVerif script modelling user-controlled anonymity in $\mathsf{DAA}_{\mathsf{RSA}'}$

```
free c.
free chlP.
free chlM.
free skA,skB.

fun bottom/0.

let SignerP =
    new aj;new aj';new as;new as';( !join )|( !sign )|(
        new cnt;new DAASeed;
        ! out(aj,(wparams,DAASeed,cnt,skM));
        in(aj',(cre,tsk));
        (!
            in(c,(xmsg,=bottom));new XXTERMINATION;
            out(as,(wparams,bottom,xmsg,cre,tsk,XXTERMINATION));
            in(as',z);out(c,z)
        )|(!
            in(c,(xmsg,xbsn));new XXTERMINATION;
            out(as,(wparams,(chlP,xbsn),xmsg,cre,tsk,XXTERMINATION));
            in(as',z);out(c,z)
        )|(
           out(wb,(cre,tsk))
        )
    ).

let Challenge =
    new as;new as';( sign ) | (
        in(bA,(creA,tskA));
        in(bB,(creB,tskB));
        let cre = choice[creA,creB] in
        let tsk = choice[tskA,tskB] in
        (
            in(c,(xmsg,=bottom));new XXTERMINATION;
            out(as,(wparams,bottom,xmsg,cre,tsk,XXTERMINATION));
            in(as',x);out(c,x)
        )|(
            in(c,(xmsg,xbsn));new XXTERMINATION;
            out(as,(wparams,(chlM,xbsn),xmsg,cre,tsk,XXTERMINATION));
            in(as',x);out(c,x)
        )
    ).

process
    in(c,wparams);new bA;new bB;
    (let (wb,skM) = (bA,skA) in SignerP)|
    (let (wb,skM) = (bB,skB) in SignerP)|
    (Challenge)
```

Listing C.5: Optimised ProVerif script modelling user-controlled anonymity in $\mathsf{DAA}_{\mathsf{RSA}'}$

ProVerif scripts supporting Chapter 5

The proof of claims made in Example 5.3 can be automatically verified using Listing D.1 as input to ProVerif, and Example 5.7 can be checked using Listing D.2.

```
free c,k,m,n,s.

query ev:eventM() ==> ev:eventK().
query ev:eventN() ==> ev:eventK().
query ev:eventS() ==> ev:eventK().

process
    new a1;new a2;new a3;new a4;new a5;new a6;new a7;new a8;
    (
        out(a1,a1);in(a3,x);          (* sync 1 *)
        event eventK();
        out(c,k);
        out(a5,a5);in(a7,x');         (* sync 2 *)
        event eventM();
        out(c,m)
    )|(
        out(a2,a2);in(a6,y);          (* sync 2 *)
        event eventN();
        out(c,n)
    )|(
        out(a4,a2);in(a8,y);          (* sync 3 *)
        event eventS();
        out(c,s)
    )|(
        in(a1,x);out(a3,x);                                  (* sync 1 *)
        in(a5,x');in(a2,y); out(a7,x');out(a6,y);            (* sync 2 *)
        in(a4,z);out(a8,z)                                   (* sync 3 *)
    )
```

Listing D.1: Analysis of barrier synchronisation behaviour in support of Example 5.3

```
free c,k,m,n,s.

fun bot/0.
fun pair/2.

reduc fst(pair(x,y)) = x.
reduc snd(pair(x,y)) = y.

process
    new a1;new a2;new a3;new a4;new a5;new a6;new a7;new a8;
    (
        (* start sync 1 *)
        out(a1,choice[a1,pair(c,pair(k,pair(m,bot)))]);
        in(a3,x);
        (*  end  sync 1 *)

        out(choice[c,fst(x)],choice[k,fst(snd(x))]);

        (* start sync 2 *)
        out(a5,choice[a5,pair(fst(x),pair(fst(snd(snd(x))),bot))]);
        in(a7,x');
        (*  end  sync 2 *)

        out(choice[c,fst(x')],choice[m,fst(snd(x'))])
    )|(
        (* start sync 2 *)
        out(a2,choice[a2,pair(c,pair(n,bot))]);
        in(a6,y);
        (*  end  sync 2 *)

        out(choice[c,fst(y)],choice[n,fst(snd(y))])
    )|(
        (* start sync 3 *)
        out(a4,choice[a4,pair(c,pair(s,bot))]);
        in(a8,z);
        (*  end  sync 3 *)

        out(choice[c,fst(z)],choice[s,fst(snd(z))])
    )|(
        in(a1,x);out(a3,x);                              (* sync 1 *)
        in(a5,x');in(a2,y);out(a7,x');out(a6,y);         (* sync 2 *)
        in(a4,z);out(a8,z)                               (* sync 3 *)
    )
```

Listing D.2: Analysis of $\Delta(P) \sim Q$ in support of Example 5.7

E

Privacy results using ProSwapper

Chapter 5 introduces a methodology to automatically analyse equivalence properties between processes which use barrier synchronisation. This technique has been implemented as a tool called ProSwapper which is available online: `http://www.bensmyth.com/proswapper.php`. More precisely, ProSwapper implements the compiler specified by Definition 5.7. It takes processes with barriers as input and outputs a process, defined over the standard ProVerif syntax, with the possible swapping processes defined by process macros. If ProVerif can prove equivalence for a particular swapping process, then the input script should satisfy observational equivalence. The ProSwapper output included in this appendix was produced by ProSwapper version 0.2 alpha.

Several of the examples in Chapter 5 have been automatically verified using ProSwapper. Listing E.1 models Example 5.8 (pp106) which can be compiled using ProSwapper to produce Listing E.2, and automatically analysed using ProVerif. (For readability, the swapping process used to prove observational equivalence is included and the alternative swapping processes are omitted.) The remainder of this appendix considers vote privacy in the FOO electronic voting protocol (Section E.1) and privacy in vehicular ad-hoc network (Section E.2)

```
free c,m,n.

process
    (
        out(c,m);
        sync 1;
        out(c,choice[m,n])
    )|(
        sync 1;
        out(c,choice[n,m])
    )
```

Listing E.1: ProSwapper input script in support of Example 5.8

```
free c,m,n.

process
    new aaa'1;new aaa'2;new aaa'3;new aaa'4;
    (
        out(c,m);
        out(aaa'1,(n,m,c));
        in(aaa'2,(xxx'1,xxx'2,xxx'3));
        out(xxx'3,choice[xxx'2,xxx'1])
    )|(
        out(aaa'3,(m,n,c));
        in(aaa'4,(xxx'1,xxx'2,xxx'3));
        out(xxx'3,choice[xxx'2,xxx'1])
    )|(
        in(aaa'3,y1);in(aaa'1,y2);
        out(aaa'4,choice[y1,y2]);out(aaa'2,choice[y2,y1])
    )
```

Listing E.2: ProSwapper output script in support of Example 5.8

E.1 Privacy in electronic voting

Section 5.3.1 presents an analysis of privacy in the FOO electronic voting protocol. This section provides the scripts used to automatically perform this analysis. The free name declarations, function definitions and equations appear in Listing E.3. The encoding (Listing E.4) of vote privacy follows immediately from pp109 and this script can be provided as input to ProSwapper to produce Listing E.5. This listing can be automatically analysed with ProVerif.

E.2 Privacy in vehicular ad-hoc networks

Dahl, Delaune & Steel [DDS10] adopt the notion of swapping defined in [DRS08] to analyse privacy in the CMIX vehicular ad-hoc networking protocol [FRF+07]. In this section, [DDS10] will be revisited to show that ProSwapper can be used to automatically prove privacy. First, we recall the model used by Dahl, Delaune & Steel. Listing E.6 contains the free name declarations, function definitions and equations as specified by [DDS10]; and Listings E.7 & E.8 present the ProVerif script manually derived using [DRS08] (see [DDS10] for the original encoding without swapping). Listing E.9 presents a variant of Listings E.7 & E.8 written by Dahl, suitable for input to ProSwapper; the relative complexity of the encodings should illustrate the benefits of our abstraction. The ProSwapper input script can be compiled to produce Listing E.10, which can be automatically analysed using ProVerif. The execution times are comparable: ProVerif takes 44ms to analyse Listings E.7/E.8, and 30ms to analyse Listing E.10 (the associated compile time is negligible).

```
free c.

free skR,skA,skB.

fun s/0.
fun s'/0.
fun true/0.
fun blind/2.
fun unblind/2.
fun open/2.
fun commit/2.
fun pk/1.
fun getmsg/1.
fun sign/2.
fun checksign/2.

equation unblind(x,sign(y,blind(x,z))) = sign(y,z).
equation unblind(x,blind(x,y)) = y.
equation open(x,commit(x,y)) = y.
equation checksign(pk(x),sign(x,y)) = true.
equation getmsg(sign(x,y)) = y.
```

Listing E.3: Free name declarations, function definitions and equations for FOO

```
let foo =
    new k;new k';
    let M = commit(k,Xvote) in
    let MM = blind(k',M) in
    out(c,(pk(Xsk),sign(Xsk,MM)));
    in(c,y);
    if checksign(pk(skR),y) = true then
    if getmsg(y) = MM then
    let M' = unblind(k',y) in
    sync 1;
    out(c,(M,M'));
    sync 2;
    in(c,(z,=M,=M'));
    out(c,(z,k)).

process
      (let (Xsk,Xvote) = (skA,choice[s,s']) in foo)
    | (let (Xsk,Xvote) = (skB,choice[s',s]) in foo)
```

Listing E.4: ProSwapper input script modelling vote privacy in FOO

```
process
    new aaa'1;new aaa'2;new aaa'3;new aaa'4;
    new aaa'5;new aaa'6;new aaa'7;new aaa'8;
    (
        let (Xsk,Xvote) = (skA,choice[s(),s'()]) in
        new k;new k';
        let M = commit(k,Xvote) in
        let MM = blind(k',M) in
        out(c, (pk(Xsk),sign(Xsk,MM)));
        in(c, y);
        if checksign(pk(skR),y) = true() then
        if getmsg(y) = MM then
        let M' = unblind(k',y) in
        out(aaa'1, (k,M',M,c));
        in(aaa'2, (xxx'1,xxx'2,xxx'3,xxx'4));
        out(xxx'4, (xxx'3,xxx'2));
        out(aaa'3, (xxx'1,xxx'4,xxx'2,xxx'3));
        in(aaa'4, (xxx'1,xxx'2,xxx'3,xxx'4));
        in(xxx'2, (z,=xxx'4,=xxx'3));
        out(xxx'2, (z,xxx'1))
    )|(
        let (Xsk,Xvote) = (skB,choice[s'(),s()]) in
        new k;new k';
        let M = commit(k,Xvote) in
        let MM = blind(k',M) in
        out(c, (pk(Xsk),sign(Xsk,MM)));
        in(c, y);
        if checksign(pk(skR),y) = true() then
        if getmsg(y) = MM then
        let M' = unblind(k',y) in
        out(aaa'5, (k,M',M,c));
        in(aaa'6, (xxx'1,xxx'2,xxx'3,xxx'4));
        out(xxx'4, (xxx'3,xxx'2));
        out(aaa'7, (xxx'1,xxx'4,xxx'2,xxx'3));
        in(aaa'8, (xxx'1,xxx'2,xxx'3,xxx'4));
        in(xxx'2, (z,=xxx'4,=xxx'3));
        out(xxx'2, (z,xxx'1))
    )|(
        in(aaa'5, y1);in(aaa'1, y2);
        out(aaa'6,choice[y1,y2]);out(aaa'2,choice[y2,y1]);
        in(aaa'7, y3);in(aaa'3, y4);
        out(aaa'8,choice[y3,y3]);out(aaa'4,choice[y4,y4])
    )
```

Listing E.5: ProSwapper output script modelling vote privacy in FOO

```
free c,ack,request.
free enterLeft,enterRight,zone,exitLeft,exitRight.

private free sk.
private free prsu,krsu,kc.
private free pva1,kva1,pva2,kva2,pvb1,kvb1,pvb2,kvb2.

fun pk/1.
fun sign/2.
fun enc/2.
fun senc/3.

reduc getmess(sign(x,y)) = x.
reduc checksign(sign(x,y),pk(y)) = x.
reduc decrypt(enc(x,pk(y)),y) = x.
reduc sdecrypt(senc(x,y,r),y) = x.
```

Listing E.6: Free name declarations, function definitions and equations for [DDS10]

```
let rsu =
  in(c,X);
  let (SignaturePart,CertificatePart) = X in
  let (Pv,PkKv) = checksign(CertificatePart,pk(kc)) in
  let (=request,Ts,=Pv) = checksign(SignaturePart,PkKv) in
  out(c,(enc(sign((Pv,sk,Ts),krsu),PkKv),sign((prsu,pk(krsu)),kc))).

let carBefore =
  out(enter,pv1);

  new ts;
  out(enter,(sign((request,ts,pv1),kv1),sign((pv1,pk(kv1)),kc)));
  in(enter,X);
  let (encedPart,CertificatePart) = X in
  let (Prsu,PkKrsu) = checksign(CertificatePart,pk(kc)) in
  let SignaturePart = decrypt(encedPart,kv1) in
  let (=pv1,sk,=ts) = checksign(SignaturePart,PkKrsu) in

  new r; out(zone,senc(pv1,sk,r));

  out(pcPv1,pv1);
  out(pcKv1,kv1);
  out(pcPv2,pv2);
  out(pcKv2,kv2);
  out(pcSk,sk);
  out(pcExit,exit).

let carAfter =
  new r;new r';

  out(exit,senc(pv1,sk,r));
  out(exit,senc(pv2,sk,r'));

  out(exit,pv2).
```

Listing E.7: ProVerif script modelling privacy in [DDS10] with manual swapping, Part I

```
process
  out(c,pk(kc)); out(c,pk(krsu));
  (
    !rsu
  )|(
    new pcaExit; new pcaKv1; new pcaKv2; new pcaPv1; new pcaPv2; new pcaSk;
    new pcbExit; new pcbKv1; new pcbKv2; new pcbPv1; new pcbPv2; new pcbSk;
    (
      let (kv1,kv2,pv1,pv2) = (kva1,kva2,pva1,pva2) in
      let (pcKv1,pcKv2,pcPv1,pcPv2) = (pcaKv1,pcaKv2,pcaPv1,pcaPv2) in
      let enter = enterLeft in
      let exit = choice[exitLeft,exitRight] in
      let pcExit = pcaExit in
      let pcSk = pcaSk in
      carBefore
    )|(
      let (kv1,kv2,pv1,pv2) = (kvb1,kvb2,pvb1,pvb2) in
      let (pcKv1,pcKv2,pcPv1,pcPv2) = (pcbKv1,pcbKv2,pcbPv1,pcbPv2) in
      let enter = enterRight in
      let exit = choice[exitRight,exitLeft] in
      let pcExit = pcbExit in
      let pcSk = pcbSk in
      carBefore
    )|(
      in(pcaPv1,aPv1); in(pcaKv1,aKv1); in(pcaPv2,aPv2);
      in(pcaKv2,aKv2); in(pcaSk,aSk); in(pcaExit,aExit);
      in(pcbPv1,bPv1); in(pcbKv1,bKv1); in(pcbPv2,bPv2);
      in(pcbKv2,bKv2); in(pcbSk,bSk); in(pcbExit,bExit);
      (
        let pv1 = choice[aPv1,bPv1] in
        let kv1 = choice[aKv1,bKv1] in
        let pv2 = choice[aPv2,bPv2] in
        let kv2 = choice[aKv2,bKv2] in
        let sk = choice[aSk,bSk] in
        let exit = choice[aExit,bExit] in
        carAfter
      )|(
        let pv1 = choice[bPv1,aPv1] in
        let kv1 = choice[bKv1,aKv1] in
        let pv2 = choice[bPv2,aPv2] in
        let kv2 = choice[bKv2,aKv2] in
        let sk = choice[bSk,aSk] in
        let exit = choice[bExit,aExit] in
        carAfter
      )
    )
  )
```

Listing E.8: ProVerif script modelling privacy in [DDS10] with manual swapping, Part II

```
let rsu =
  in(c,X);
  let(SignaturePart,CertificatePart) = X in
  let(Pv,PkKv) = checksign(CertificatePart,pk(kc)) in
  let(=request,Ts,=Pv) = checksign(SignaturePart,PkKv) in
  out(c,(enc(sign((Pv,sk,Ts),krsu),PkKv),sign((prsu,pk(krsu)),kc))).

let car =
  out(enter,pv1);
  new ts;
  out(enter,(sign((request,ts,pv1),kv1),sign((pv1,pk(kv1)),kc)));
  in(enter,X);
  let(encedPart,CertificatePart) = X in
  let(Prsu,PkKrsu) = checksign(CertificatePart,pk(kc)) in
  let SignaturePart = dec(encedPart,kv1) in
  let(=pv1,sk,=ts) = checksign(SignaturePart,PkKrsu) in
  new r;
  out(zone,senc(pv1,sk,r));
  sync 1;
  new r;
  new r';
  out(exit,senc(pv1,sk,r));
  out(exit,senc(pv2,sk,r'));
  out(exit,pv2).

process
  out(c,pk(kc));
  out(c,pk(krsu));
  (
    !rsu
  )|(
    let (kv1,kv2,pv1,pv2) = (kva1,kva2,pva1,pva2) in
    let enter = enterLeft in
    let exit = choice[exitLeft,exitRight] in
    car
  )|(
    let (kv1,kv2,pv1,pv2) = (kvb1,kvb2,pvb1,pvb2) in
    let enter = enterRight in
    let exit = choice[exitRight,exitLeft] in
    car
  )
```

Listing E.9: ProSwapper input script modelling privacy in [DDS10]

```
process new aaa'1;new aaa'2;new aaa'3;new aaa'4;
  out(c,pk(kc));out(c,pk(krsu));
  (!
    in(c,X);
    let (SignaturePart,CertificatePart) = X in
    let (Pv,PkKv) = checksign(CertificatePart,pk(kc)) in
    let (=request,Ts,=Pv) = checksign(SignaturePart,PkKv) in
    out(c,(enc(sign((Pv,sk,Ts),krsu),PkKv),sign((prsu,pk(krsu)),kc)))
  )|(
    let (kv1,kv2,pv1,pv2) = (kva1,kva2,pva1,pva2) in
    let enter = enterLeft in
    let exit = choice[exitLeft,exitRight] in
    out(enter,pv1);new ts;
    out(enter,(sign((request,ts,pv1),kv1),sign((pv1,pk(kv1)),kc)));
    in(enter,X);
    let (encedPart,CertificatePart) = X in
    let (Prsu,PkKrsu) = checksign(CertificatePart,pk(kc)) in
    let SignaturePart = dec(encedPart,kv1) in
    let (=pv1,sk,=ts) = checksign(SignaturePart,PkKrsu) in
    new r;out(zone,senc(pv1,sk,r));
    out(aaa'1,(pv2,sk,pv1,exit));
    in(aaa'2,(xxx'1,xxx'2,xxx'3,xxx'4));
    new r;new r';
    out(xxx'4,senc(xxx'3,xxx'2,r));out(xxx'4,senc(xxx'1,xxx'2,r'));
    out(xxx'4,xxx'1)
  )|(
    let (kv1,kv2,pv1,pv2) = (kvb1,kvb2,pvb1,pvb2) in
    let enter = enterRight in
    let exit = choice[exitRight,exitLeft] in
    out(enter,pv1);new ts;
    out(enter,(sign((request,ts,pv1),kv1),sign((pv1,pk(kv1)),kc)));
    in(enter,X);
    let (encedPart,CertificatePart) = X in
    let (Prsu,PkKrsu) = checksign(CertificatePart,pk(kc)) in
    let SignaturePart = dec(encedPart,kv1) in
    let (=pv1,sk,=ts) = checksign(SignaturePart,PkKrsu) in
    new r;out(zone,senc(pv1,sk,r));
    out(aaa'3,(pv2,sk,pv1,exit));
    in(aaa'4,(xxx'1,xxx'2,xxx'3,xxx'4));
    new r;new r';
    out(xxx'4,senc(xxx'3,xxx'2,r));out(xxx'4,senc(xxx'1,xxx'2,r'));
    out(xxx'4,xxx'1)
  )|(
    in(aaa'3,y1);in(aaa'1,y2);
    out(aaa'4,choice[y1,y2]);out(aaa'2,choice[y2,y1])
  )
```

Listing E.10: ProSwapper output script modelling privacy in [DDS10]

Bibliography

[AB02] Martín Abadi and Bruno Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. In *POPL'02: 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 33–44. ACM Press, 2002.

[AB05a] Martín Abadi and Bruno Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. *Journal of the ACM*, 52(1):102–146, 2005.

[AB05b] Martín Abadi and Bruno Blanchet. Computer-assisted verification of a protocol for certified email. *Science of Computer Programming*, 58(1–2):3–27, 2005.

[AB05c] Xavier Allamigeon and Bruno Blanchet. Reconstruction of Attacks against Cryptographic Protocols. In *CSFW'05: 18th Computer Security Foundations Workshop*, pages 140–154. IEEE Computer Society, 2005.

[Aba00] Martín Abadi. Security Protocols and their Properties. In Friedrich L. Bauer and Ralf Steinbrüggen, editors, *Foundations of Secure Computation*, NATO Science Series, pages 39–60. IOS Press, 2000.

[ABB+04] William Aiello, Steven M. Bellovin, Matt Blaze, Ran Canetti, John Ioannidis, Keromytis Keromytis, and Omer Reingold. Just Fast Keying: Key Agreement in a Hostile Internet. *ACM Transactions on Information and System Security*, 7(2):242–273, 2004.

[ABB+05] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *CAV'05: 17th International Conference on Computer Aided Verification*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.

[ABF07] Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just Fast Keying in the Pi Calculus. *ACM Transactions on Information and System Security*, 10(3), 2007.

[AC04a] Martín Abadi and Véronique Cortier. Deciding Knowledge in Security Protocols Under Equational Theories. In *ICALP'04: 31st International Colloquium on Automata, Languages and Programming*, volume 3142 of *LNCS*, pages 46–58. Springer, 2004.

[AC04b] Alessandro Armando and Luca Compagna. SATMC: A SAT-Based Model Checker for Security Protocols. In *JELIA'04: 9th European Conference on Logics in Artificial Intelligence*, volume 3229 of *LNCS*, pages 730–733. Springer, 2004.

[AC05a] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under (many more) equational Theories. In *CSFW'05: 18th Computer Security Foundations Workshop*, pages 62–76. IEEE Computer Society, 2005.

[AC05b] Alessandro Armando and Luca Compagna. An Optimized Intruder Model for SAT-based Model-Checking of Security Protocols. *Electronic Notes in Theoretical Computer Science*, 125(1):91–108, 2005.

[AC06] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 367(1–2):2–32, 2006.

[ACRR10] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing Unlinkability and Anonymity Using the Applied Pi Calculus. In *CSF'10: 23rd IEEE Computer Security Foundations Symposium*, pages 107–121. IEEE Computer Society, 2010.

[Adi06] Ben Adida. *Advances in Cryptographic Voting Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2006.

[Adi08] Ben Adida. Helios: Web-based Open-Audit Voting. In *USENIX Security'08: 17th USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL'01: 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115. ACM Press, 2001.

[AF06] Martín Abadi and Cédric Fournet. Private email communication, 24th May 2006.

[AG97] Martín Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *CCS'97: 4th ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[AG98a] Martín Abadi and Andrew D. Gordon. A Bisimulation Method for Cryptographic Protocols. In *ESOP'98: 7th European Symposium on Programming*, volume 1381 of *LNCS*, pages 12–26. Springer, 1998.

[AG98b] Martín Abadi and Andrew D. Gordon. A Bisimulation Method for Cryptographic Protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.

[AGHP02] Martín Abadi, Neal Glew, Bill Horne, and Benny Pinkas. Certified Email with a Light On-line Trusted Third Party: Design and Implementation. In *WWW'02: 11th International World Wide Web Conference*, pages 387–395. ACM Press, 2002.

[AJ89] Norbert S. Arenstorf and Harry F. Jordan. Comparing barrier algorithms. *International Journal of Parallel Computing*, 12(2):157–170, 1989.

[AMPQ09] Ben Adida, Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios. In *EVT/WOTE'09: Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*. USENIX Association, 2009.

[APW09] Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. Plaintext Recovery Attacks against SSH. In *S&P'09: 30th IEEE Symposium on Security and Privacy*, pages 16–26. IEEE Computer Society, 2009.

[AR00] Martín Abadi and Phillip Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). In *IFIP TCS'00: 1st International Conference on Theoretical Computer Science*, volume 1872 of *LNCS*, pages 3–22. Springer, 2000.

[AR02] Martín Abadi and Phillip Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[ARR10] Myrto Arapinis, Eike Ritter, and Mark D. Ryan. Private communication, 2010.

[BAF05] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *LICS'05: 20th Annual IEEE Symposium on Logic In Computer Science*, pages 331–340. IEEE Computer Society, 2005.

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

[Bau05] Mathieu Baudet. Deciding security of protocols against off-line guessing attacks. In *CCS'05: 12th ACM Conference on Computer and Communications Security*, pages 16–25. ACM Press, 2005.

[BBN04] Johannes Borgström, Sébastien Briais, and Uwe Nestmann. Symbolic Bisimulation in the Spi Calculus. In *CONCUR'04: 15th International Conference on Concurrency Theory*, volume 3170 of *LNCS*, pages 161–176. Springer, 2004.

[BC08] Bruno Blanchet and Avik Chaudhuri. Automated Formal Analysis of a Protocol for Secure File Sharing on Untrusted Storage. In *S&P'08: 29th IEEE Symposium on Security and Privacy*, pages 417–431. IEEE Computer Society, 2008.

[BCC04] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct Anonymous Attestation. In *CCS'04: 11th ACM Conference on Computer and Communications Security*, pages 132–145. ACM Press, 2004.

[BCC05] Ernie Brickell, Jan Camenisch, and Liqun Chen. The DAA scheme in context. In Chris Mitchell, editor, *Trusted Computing*, volume 6 of *Professional Applications of Computing Series*, pages 143–174. The Institute of Engineering and Technology, 2005.

[BCD09] Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. YAPA: A Generic Tool for Computing Intruder Knowledge. In *RTA'09: 20th International Conference on Rewriting Techniques and Applications*, volume 5595 of *LNCS*, pages 148–163. Springer, 2009.

[BCD10] Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. YAPA: A generic tool for computing intruder knowledge. arXiv ePrint Archive, Report abs/1005.0737: `http://arxiv.org/abs/1005.0737`, 2010.

[BCDG88] Ernest F. Brickell, David Chaum, Ivan Damgård, and Jeroen van de Graaf. Gradual and Verifiable Release of a Secret. In *CRYPTO'87: 7th International Cryptology Conference*, volume 293 of *LNCS*, pages 156–166. Springer, 1988.

[BCFZ08] Karthikeyan Bhargavan, Ricardo Corin, Cédric Fournet, and Eugen Zălinescu. Cryptographically Verified Implementations for TLS. In *CCS'08: ACM Conference on Computer and Communications Security*, pages 459–468. ACM Press, 2008.

[BCH10] Mayla Brusó, Konstantinos Chatzikokolakis, and Jerry den Hartog. Formal verification of privacy for RFID systems. In *CSF'10: 23rd IEEE Computer Security Foundations Symposium*, pages 75–88. IEEE Computer Society, 2010.

[BCL08a] Ernie Brickell, Liqun Chen, and Jiangtao Li. A New Direct Anonymous Attestation Scheme from Bilinear Maps. In *Trust'08: 1st International Conference on Trusted Computing and Trust in Information Technologies*, volume 4968 of *LNCS*, pages 166–178, 2008.

[BCL08b] Ernie Brickell, Liqun Chen, and Jiangtao Li. Simplified Security Notions of Direct Anonymous Attestation and a Concrete Scheme from Pairings. Cryptology ePrint Archive, Report 2008/104, 2008.

[BCL09] Ernie Brickell, Liqun Chen, and Jiangtao Li. Simplified security notions of Direct Anonymous Attestation and a concrete scheme from pairings. *International Journal of Information Security*, 8(5):315–330, 2009.

[Bel99] Mihir Bellare. Practice-Oriented Provable Security. In *Lectures on Data Security: Modern Cryptology in Theory and Practice*, volume 1561 of *LNCS*, pages 1–15. Springer, 1999.

[Ben06] Josh Benaloh. Simple Verifiable Elections. In *EVT'06: Electronic Voting Technology Workshop*. USENIX Association, 2006.

[Ben07] Josh Benaloh. Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. In *EVT'07: Electronic Voting Technology Workshop*. USENIX Association, 2007.

[BFG06] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. Verified Reference Implementations of WS-Security Protocols. In *WS-FM'06: 3rd International Workshop on Web Services and Formal Methods*, volume 4184 of *LNCS*, pages 88–106. Springer, 2006.

[BFGS08] Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Nikhil Swamy. Verified Implementations of the Information Card Federated Identity-Management Protocol. In *ASIACCS'08: 3rd ACM Symposium on Information, Computer and Communications Security*, pages 123–135. ACM Press, 2008.

[BFGT06] Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Stephen Tse. Verified Interoperable Implementations of Security Protocols. In *CSFW'06: 19th IEEE Computer Security Foundations Workshop*, pages 139–152. IEEE Computer Society, 2006.

[BFGT08] Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Stephen Tse. Verified Interoperable Implementations of Security Protocols. *ACM Transactions on Programming Languages and Systems*, 31(1):1–61, 2008.

[BHK05] Yohan Boichut, Pierre-Cyrille Héam, and Olga Kouchnarenko. Automatic Verification of Security Protocols Using Approximations. Technical Report RR-5727, INRIA, 2005.

[BHK06] Yohan Boichut, Pierre-Cyrille Héam, and Olga Kouchnarenko. Handling Algebraic Properties in Automatic Analysis of Security Protocols. In *ICTAC'06: 3rd International Colloquium on Theoretical Aspects of Computing*, volume 4281 of *LNCS*, pages 153–167. Springer, 2006.

[BHK09] Yohan Boichut, Pierre-Cyrille Héam, and Olga Kouchnarenko. Tree Automata for Detecting Attacks on Protocols with Algebraic Cryptographic Primitives. *Electronic Notes in Theoretical Computer Science*, 239:57–72, 2009.

[BHKO04] Yohan Boichut, Pierre-Cyrille Héam, Olga Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay Technique to Automatically Verify Security Protocols, 2004.

[BHM08] Michael Backes, Cătălin Hriţcu, and Matteo Maffei. Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-calculus. In *CSF'08: 21st IEEE Computer Security Foundations Symposium*, pages 195–209. IEEE Computer Society, 2008.

[BJPV09] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *LICS'09: 24th Annual IEEE Symposium on Logic In Computer Science*, pages 39–48. IEEE Computer Society, 2009.

[BL09a] Ernie Brickell and Jiangtao Li. Enhanced Privacy ID: A Remote Anonymous Attestation Scheme for Hardware Devices. *Intel Technology Journal Security*, 13(2):96–111, June 2009.

[BL09b] Ernie Brickell and Jiangtao Li. Enhanced Privacy ID from Bilinear Pairing. Cryptology ePrint Archive, Report 2009/095, 2009.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW'01: 14th IEEE Computer Security Foundations Workshop*, pages 82–96. IEEE Computer Society, 2001.

[Bla02] Bruno Blanchet. From Secrecy to Authenticity in Security Protocols. In *SAS'02: 9th International Static Analysis Symposium*, volume 2477 of *LNCS*, pages 342–359. Springer, 2002.

[Bla04] Bruno Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. In *S&P'04: 25th IEEE Symposium on Security and Privacy*, pages 86–100. IEEE Computer Society, 2004.

[Bla08] Bruno Blanchet. *Vérification automatique de protocoles cryptographiques: modèle formel et modèle calculatoire.* Mémoire d'habilitation à diriger des recherches, Université Paris-Dauphine, 2008.

[Bla09] Bruno Blanchet. Automatic Verification of Correspondences for Security Protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[Bla11] Bruno Blanchet. Using Horn clauses for analyzing security protocols. In Véronique Cortier and Steve Kremer, editor, *Formal Models and Techniques for Analyzing Security Protocols*, chapter 5. IOS Press, 2011.

[BLP05a] Shane Balfe, Amit D. Lakhani, and Kenneth G. Paterson. Securing peer-to-peer networks using trusted computing. In Chris Mitchell, editor, *Trusted Computing*, volume 6 of *Professional Applications of Computing Series*, pages 271–298. The Institute of Engineering and Technology, 2005.

[BLP05b] Shane Balfe, Amit D. Lakhani, and Kenneth G. Paterson. Trusted Computing: Providing Security for Peer-to-Peer Networks. In *P2P'05: 5th IEEE International Conference on Peer-to-Peer Computing*, pages 117–124. IEEE Computer Society, 2005.

[BM92] Leonor M. Barroca and John A. Mcdermid. Formal methods: Use and relevance for the development of safety-critical systems. *The Computer Journal*, 35(6):579–599, 1992.

[BMU08] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-Knowledge in the Applied Pi-calculus and Automated Verification of the Direct Anonymous Attestation Protocol. In *S&P'08: 29th IEEE Symposium on Security and Privacy*, pages 202–215. IEEE Computer Society, 2008.

[BMV03] David A. Basin, Sebastian Mödersheim, and Luca Viganò. An On-the-Fly Model-Checker for Security Protocol Analysis. In *ESORICS'03: 8th European Symposium On Research In Computer Security*, volume 2808 of *LNCS*, pages 253–270. Springer, 2003.

[BMV05] David A. Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *Internation Journal of Information Security*, 4(3):181–208, 2005.

[BN02] Johannes Borgström and Uwe Nestmann. On bisimulations for the spi calculus. In *AMAST'02: 9th International Conference on Algebraic Methodology and Software Technology*, volume 2422 of *LNCS*, pages 287–303. Springer, 2002.

[BN05] Johannes Borgström and Uwe Nestmann. On bisimulations for the spi calculus. *Mathematical Structures in Computer Science*, 15(3):487–552, 2005.

[Bol97] Dominique Bolignano. Towards a Mechanization of Cryptographic Protocol Verification. In *CAV'97: 9th International Conference on Computer Aided Verification*, volume 1254 of *LNCS*, pages 131–142. Springer, 1997.

[Bou00] Fabrice Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In *EUROCRYPT'00: 19th International Conference on the Theory and Applications of Cryptographic Techniques*, volume 1807 of *LNCS*, pages 431–444. Springer, 2000.

[Bow07] Debra Bowen. Secretary of State Debra Bowen Moves to Strengthen Voter Confidence in Election Security Following Top-to-Bottom Review of Voting Systems. California Secretary of State, press release DB07:042 `http://www.sos.ca.gov/elections/voting_systems/ttbr/db07_042_ttbr_system_decisions_release.pdf`, August 2007.

[BP03] Bruno Blanchet and Andreas Podelski. Verification of Cryptographic Protocols: Tagging Enforces Termination. In *FoSSaCS'03: 6th International Conference on Foundations of Software Science and Computational Structures*, volume 2620 of *LNCS*, pages 136–152. Springer, 2003.

[BP05] Bruno Blanchet and Andreas Podelski. Verification of cryptographic protocols: tagging enforces termination. *Theoretical Computer Science*, 333(1-2):67–90, March 2005. Special issue FoSSaCS'03: 6th International Conference on Foundations of Software Science and Computational Structures.

[Bri05] Sébastien Briais. The ABC User's Guide. `http://sbriais.free.fr/tools/abc/`, 2005.

[Bro86] Eugene D. Brooks, III. The Butterfly Barrier. *International Journal of Parallel Programming*, 15(4):295–307, 1986.

[BRS07] Anguraj Baskar, Ramaswamy Ramanujam, and S. P. Suresh. Knowledge-based modelling of voting protocols. In *TARK: 11th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 62–71. ACM Press, 2007.

[BS11] Bruno Blanchet and Ben Smyth. ProVerif: Automatic Cryptographic Protocol Verifier User Manual & Tutorial. `http://www.proverif.ens.fr/`, 2011.

[Bun09] Bundesverfassungsgericht (Germany's Federal Constitutional Court). Use of voting computers in 2005 Bundestag election unconstitutional. Press release 19/2009 `http://www.bundesverfassungsgericht.de/en/press/bvg09-019en.html`, March 2009.

[BVQ10] Josh Benaloh, Serge Vaudenay, and Jean-Jacques Quisquater. Final Report of IACR Electronic Voting Committee. International Association for Cryptologic Research. `http://www.iacr.org/elections/eVoting/finalReportHelios_2010-09-27.html`, Sept 2010.

[CCM07] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a Secure Voting System. Technical Report 2007-2081, Cornell University, May 2007. Revised March 2008.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a Secure Voting System. In *S&P'08: 29th IEEE Symposium on Security and Privacy*, pages 354–368. IEEE Computer Society, 2008.

[CD09] Véronique Cortier and Stéphanie Delaune. A method for proving observational equivalence. In *CSF'09: 22nd IEEE Computer Security Foundations Symposium*, pages 266–276. IEEE Computer Society, 2009.

[CDK09a] Rohit Chadha, Stéphanie Delaune, and Steve Kremer. Epistemic Logic for the Applied Pi Calculus. In *FMOODS/FORTE'09: Joint 11th International Conference on Formal Methods for Open Object-Based Distributed Systems and 29th International Conference on Formal Techniques for Networked and Distributed Systems*, volume 5522 of *LNCS*, pages 182–197. Springer, 2009.

[CDK09b] Stefan Ciobâca, Stéphanie Delaune, and Steve Kremer. Computing Knowledge in Security Protocols under Convergent Equational Theories. In *CADE'09: 22nd International Conference on Automated Deduction*, volume 5663 of *LNCS*, pages 355–370. Springer, 2009.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *CRYPTO'94: 14th International Cryptology Conference*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

[CEC+08] David Chaum, Aleks Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan Sherman, and Poorvi Vora. Scantegrity: End-to-End Voter-Verifiable Optical-Scan Voting. *IEEE Security and Privacy*, 6(3):40–46, 2008.

[CEG88] David Chaum, Jan-Hendrik Evertse, and Jeroen van de Graaf. An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations. In *EUROCRYPT'87: 4th International Conference on the Theory and Applications of Cryptographic Techniques*, volume 304 of *LNCS*, pages 127–141. Springer, 1988.

[CEGP87] David Chaum, Jan-Hendrik Evertse, Jeroen van de Graaf, and René Peralta. Demonstrating Possession of a Discrete Logarithm Without Revealing It. In *CRYPTO'86: 6th International Cryptology Conference*, volume 263 of *LNCS*, pages 200–212. Springer, 1987.

[Ces10] Emanuele Cesena. *Trace Zero Varieties in Pairing-based Cryptography*. PhD thesis, Department of Mathematics, Universitá degli Studi Roma Tre, 2010.

[CF08] Xiaofeng Chen and Dengguo Feng. Direct Anonymous Attestation for Next Generation TPM. *Journal of Computers*, 3(12):43–50, December 2008.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *EUROCRYPT'97: 16th International Conference on the Theory and Applications of Cryptographic Techniques*, volume 1233 of *LNCS*, pages 103–118. Springer, 1997.

[CH02] Jan Camenisch and Els Van Herreweghen. Design and Implementation of the *idemix* Anonymous Credential System. In *CCS'02: 9th ACM Conference on Computer and Communications Security*, pages 21–30. ACM Press, 2002.

[Cha90] David Chaum. Zero-Knowledge Undeniable Signatures. In *EUROCRYPT'90: 7th International Conference on the Theory and Applications of Cryptographic Techniques*, volume 473 of *LNCS*, pages 458–464. Springer, 1990.

[Che10] Liqun Chen. A DAA Scheme Requiring Less TPM Resources. Cryptology ePrint Archive, Report 2010/008, 2010.

[Che11] Liqun Chen. A DAA Scheme Requiring Less TPM Resources. In *INSCRYPT'09: 5th International Conference on Information Security and Cryptology*, volume 6151 of *LNCS*, pages 350–365. Springer, 2011.

[Cho06] Tom Chothia. Analysing the MUTE Anonymous File-Sharing System Using the Pi-Calculus. In *FORTE'06: 26th International Conference on Formal Techniques for Networked and Distributed Systems*, volume 4229 of *LNCS*, pages 115–130. Springer, 2006.

[CJM00] Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. Verifying Security Protocols with Brutus. *ACM Transactions on Software Engineering and Methodology*, 9(4):443–487, 2000.

[CKRT03a] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *LICS'03: 18th Annual IEEE Symposium on Logic In Computer Science*, pages 261–270. IEEE Computer Society, 2003.

[CKRT03b] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *FSTTCS'03: 23rd International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of *LNCS*, pages 124–135. Springer, 2003.

[CKRT05] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. An NP decision procedure for protocol insecurity with XOR. *Theoretical Computer Science*, 338(1-3):247–274, 2005.

[CKS04] Rohit Chadha, Steve Kremer, and Andre Scedrov. Formal analysis of multiparty contract signing. In *CSFW'04: 17th Computer Security Foundations Workshop*, pages 266–279. IEEE Computer Society, 2004.

[CKS06] Rohit Chadha, Steve Kremer, and Andre Scedrov. Formal Analysis of Multiparty Contract Signing. *Journal of Automated Reasoning*, 36(1–2):39–83, 2006.

[CL01] Jan Camenisch and Anna Lysyanskaya. An Efficient System for Nontransferable Anonymous Credentials with Optional Anonymity Revocation. In *EUROCRYPT'01: 20th International Conference on the Theory and Applications of Cryptographic Techniques*, volume 2045 of *LNCS*, pages 93–118. Springer, 2001.

[CL03] Jan Camenisch and Anna Lysyanskaya. A Signature Scheme with Efficient Protocols. In *SCN'02: 3rd Conference on Security in Communication Networks*, volume 2576 of *LNCS*, pages 268–289. Springer, 2003.

[CL10] Liqun Chen and Jiangtao Li. A note on the Chen-Morrissey-Smart DAA scheme. *Information Processing Letters*, 110(12-13):485–488, 2010.

[CLN09] Cas J. F. Cremers, Pascal Lafourcade, and Philippe Nadeau. Comparing State Spaces in Automatic Security Protocol Analysis. In *Formal to Practical Security: Papers Issued from the 2005-2008 French-Japanese Collaboration*, volume 5458 of *LNCS*, pages 70–94. Springer, 2009.

[CLR+10] Emanuele Cesena, Hans Löhr, Gianluca Ramunno, Ahmad-Reza Sadeghi, and Davide Vernizzi. Anonymous Authentication with TLS and DAA. In *TRUST'10: 3rd International Conference on Trust and Trustworthy Computing*, volume 6101 of *LNCS*, pages 47–62. Springer, 2010.

[CLS03] Hubert Comon-Lundh and Vitaly Shmatikov. Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive or. In *LICS'03: 18th Annual IEEE Symposium on Logic In Computer Science*, pages 271–281. IEEE Computer Society, 2003.

[CM98a] Jan Camenisch and Markus Michels. A Group Signature Scheme with Improved Efficiency. In *ASIACRYPT'98: 4th International Conference on the Theory and Application of Cryptology and Information Security*, volume 1514 of *LNCS*, pages 160–174. Springer, 1998.

[CM98b] Jan Camenisch and Markus Michels. A group signature scheme based on an RSA-variant. Technical Report RS-28-27, Basic Research in Computer Science (BRICS), 1998.

[CM99] Jan Camenisch and Markus Michels. Separability and Efficiency for Generic Group Signature Schemes. In *CRYPTO'99: 19th International Cryptology Conference*, volume 1666 of *LNCS*, pages 413–430. Springer, 1999.

[CM05] Cas J.F. Cremers and S. Mauw. Operational Semantics of Security Protocols. In *International Workshop on Scenarios: Models, Transformations and Tools*, volume 3466 of *lncs*, pages 66–89. sp, 2005.

[CMFP+06] Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. On Some Incompatible Properties of Voting Schemes. In *WOTE'06: Workshop on Trustworthy Elections*, 2006.

[CMFP+10] Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. On Some Incompatible Properties of Voting Schemes. In David Chaum, Markus Jakobsson, Ronald L. Rivest, and Peter Y. A. Ryan, editors, *Towards Trustworthy Elections: New Directions in Electronic Voting*, volume 6000 of *LNCS*, pages 191–199. Springer, 2010.

[CMS08a] Liqun Chen, Paul Morrissey, and Nigel P. Smart. On Proofs of Security for DAA Schemes. In *ProvSec'08: 2nd International Conference on Provable Security*, volume 5324 of *LNCS*, pages 156–175. Springer, 2008.

[CMS08b] Liqun Chen, Paul Morrissey, and Nigel P. Smart. Pairings in Trusted Computing. In *Pairing'08: 2nd International Conference on Pairing-Based Cryptography*, volume 5209 of *LNCS*, pages 1–17. Springer, 2008.

[CMS09] Liqun Chen, Paul Morrissey, and Nigel P. Smart. DAA: Fixing the pairing based protocols. Cryptology ePrint Archive, Report 2009/198, 2009.

[CMS10] Liqun Chen, Paul Morrissey, and Nigel P. Smart. DAA: Fixing the pairing based protocols. Unpublished draft, 2010.

[Com05] Luca Compagna. *SAT-based Model-Checking of Security Protocols*. PhD thesis, Joint between Universitá degli Studi di Genova and the University of Edinburgh, 2005.

[COPD07] Tom Chothia, Simona Orzan, Jun Pang, and Muhammad Torabi Dashti. A Framework for Automatically Checking Anonymity with μCRL. In *TGC'06: 2nd Symposium on Trustworthy Global Computing*, volume 4661 of *LNCS*, pages 301–318. Springer, 2007.

[CP93] David Chaum and Torben P. Pedersen. Wallet Databases with Observers. In *CRYPTO'92: 12th International Cryptology Conference*, volume 740 of *LNCS*, pages 89–105. Springer, 1993.

[CP94] Ronald Cramer and Torben P. Pedersen. Improved Privacy in Wallets with Observers. In *EUROCRYPT'93: 10th International Conference on the Theory and Applications of Cryptographic Techniques*, volume 765 of *LNCS*, pages 329–343. Springer, 1994.

[CPS10] Liqun Chen, Dan Page, and Nigel P. Smart. On the Design and Implementation of an Efficient DAA Scheme. In *CARDIS'10: 8th Internation Conference on Smart Card Research and Advanced Application*, volume 6035 of *LNCS*, pages 223–237. Springer, 2010.

[CR09] Liqun Chen and Mark D. Ryan. Attack, Solution and Verification for Shared Authorisation Data in TCG TPM. In *FAST'09: 6th International Workshop on Formal Aspects in Security and Trust*, volume 5983 of *LNCS*, pages 201–216. Springer, 2009.

[Cre06] Cas J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. PhD thesis, Eindhoven University of Technology, 2006.

[Cre07] Cas J.F. Cremers. Scyther 1.0: User Manual. `http://people.inf.ethz.ch/cremersc/scyther`, 2007.

[Cre08a] Cas J. F. Cremers. Unbounded Verification, Falsification, and Characterization of Security Protocols by Pattern Refinement. In *CCS'08: ACM Conference on Computer and Communications Security*, pages 119–128. ACM Press, 2008.

[Cre08b] Cas J.F. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *CAV'08: 20th International Conference on Computer Aided Verification*, volume 5123 of *LNCS*, pages 414–418. Springer, 2008.

[CRS05] David Chaum, Peter Y. A. Ryan, and Steve Schneider. A Practical Voter-Verifiable Election Scheme. In *ESORICS'05: 10th European Symposium On Research In Computer Security*, volume 3679 of *LNCS*, pages 118–139. Springer, 2005.

[CS97a] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups. In *CRYPTO'97: 17th International Cryptology Conference*, volume 1294 of *LNCS*, pages 410–424. Springer, 1997.

[CS97b] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups. In *CRYPTO'97: 17th International Cryptology Conference*, volume 1294 of *LNCS*, pages 410–424. Springer, 1997.

[CS10a] Tom Chothia and Vitaliy Smirnov. A Traceability Attack Against e-Passports. In *FC'10: 14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *LNCS*, pages 20–34, 2010.

[CS10b] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. Cryptology ePrint Archive, Report 2010/625, 2010.

[CS11] Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. In *CSF'11: 24th Computer Security Foundations Symposium*. IEEE Computer Society, 2011. To appear.

[Dag07] Participants of the Dagstuhl Conference on Frontiers of E-Voting. *Dagstuhl Accord*, 2007. `http://www.dagstuhlaccord.org/`.

[Dam95] Mads Dam. On the Decidability of Process Equivalences for the π-calculus. In *AMAST'95: 4th International Conference on Algebraic Methodology and Software Technology*, volume 936 of *LNCS*, pages 169–183. Springer, 1995.

[Dam97] Mads Dam. On the decidability of process equivalences for the π-calculus. *Theoretical Computer Science*, 183(2):215–228, 1997.

[DDS10] Morten Dahl, Stéphanie Delaune, and Graham Steel. Formal Analysis of Privacy for Vehicular Mix-Zones. In *ESORICS'10: 15th European Symposium on Research in Computer Security*, volume 6345 of *LNCS*, pages 55–70. Springer, 2010.

[Del11] Stéphanie Delaune. *Verification of security protocols: from confidentiality to privacy*. Mémoire d'habilitation à diriger des recherches, LSV, ENS Cachan & CNRS & INRIA, 2011.

[DKR06] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-Resistance and Receipt-Freeness in Electronic Voting. In *CSFW'06: 19th Computer Security Foundations Workshop*, pages 28–42. IEEE Computer Society, 2006.

[DKR07] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Symbolic Bisimulation for the Applied Pi Calculus. In *FSTTCS'07: 27th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4855 of *LNCS*, pages 133–145. Springer, 2007.

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

[DKR10a] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Symbolic Bisimulation for the Applied Pi Calculus. *Journal of Computer Security*, 18(2):317–377, 2010.

[DKR10b] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying Privacy-Type Properties of Electronic Voting Protocols: A Taster. In David Chaum, Markus Jakobsson, Ronald L. Rivest, and Peter Y. A. Ryan, editors, *Towards Trustworthy Elections: New Directions in Electronic Voting*, volume 6000 of *LNCS*, pages 289–309. Springer, 2010.

[DLMS99] Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols, 1999.

[DLMS04] Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

[DRS08] Stéphanie Delaune, Mark D. Ryan, and Ben Smyth. Automatic verification of privacy properties in the applied pi-calculus. In *IFIPTM'08: 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security*, volume 263 of *International Federation for Information Processing (IFIP)*, pages 263–278. Springer, 2008.

[DSHJ10] Nitish Dalal, Jenny Shah, Khushboo Hisaria, and Devesh Jinwala. A Comparative Analysis of Tools for Verification of Security Protocols. *International Journal of Communications, Network and System Sciences*, 3(10):779–787, 2010.

[DSV00] Luca Durante, Riccardo Sisto, and Adriano Valenzano. A State-Exploration Technique for Spi-Calculus Testing Equivalence Verification. In *FORTE/PSTV'00: Joint International Conference on Formal Techniques for Networked and Distributed Systems and Protocol Specification, Testing and Verification*, volume 183 of *International Federation for Information Processing (IFIP)*, pages 155–170. Kluwer, 2000.

[DSV03] Luca Durante, Riccardo Sisto, and Adriano Valenzano. Automatic Testing Equivalence Verification of Spi Calculus Specifications. *ACM Transactions on Software Engineering and Methodology*, 12(2):222–284, 2003.

[DY83] Danny Dolev and Andrew Chi-Chih Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[ElG85] Taher ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[EMM05] Santiago Escobar, Catherine Meadows, and José Meseguer. A Rewriting-Based Inference System for the NRL Protocol Analyzer: Grammar Generation. In *FMSE'05: 3rd ACM Workshop on Formal methods in Security Engineering*, pages 1–12. ACM Press, 2005.

[EMM06] Santiago Escobar, Catherine Meadows, and José Meseguer. A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties. *Theoretical Computer Science*, 367(1-2):162–202, 2006.

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *AUSCRYPT'92: Workshop on the Theory and Application of Cryptographic Techniques*, volume 718 of *LNCS*, pages 244–251. Springer, 1992.

[FRF+07] Julien Freudiger, Maxim Raya, Márk Félegyházi, Panos Papadimitratos, and Jean-Pierre Hubaux. Mix-Zones for Location Privacy in Vehicular Networks.

In *WiN-ITS'07: ACM Workshop on Wireless Networking for Intelligent Transportation Systems*, 2007. Invited paper.

[FS87] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86: 6th International Cryptology Conference*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.

[FSVY91] Thom W. Frühwirth, Ehud Y. Shapiro, Moshe Y. Vardi, and Eyal Yardeni. Logic Programs as Types for Logic Programs. In *LICS'91: 6th Annual IEEE Symposium on Logic In Computer Science*, pages 300–309. IEEE Computer Society, 1991.

[Gen95] Rosario Gennaro. Achieving independence efficiently and securely. In *PODC'95: 14th Principles of Distributed Computing Symposium*, pages 130–136. ACM Press, 1995.

[Gen00] Rosario Gennaro. A Protocol to Achieve Independence in Constant Rounds. *IEEE Transactions on Parallel and Distributed Systems*, 11(7):636–647, 2000.

[GGM+08] Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling MIFARE Classic. In *ESORICS'08: 13th European Symposium on Research in Computer Security*, volume 5283 of *LNCS*, pages 97–114. Springer, 2008.

[GHG08] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia. A Practical Attack on the MIFARE Classic. In *CARDIS'08: 8th Smart Card Research and Advanced Application Conference*, volume 5189 of *LNCS*, pages 267–282, 2008.

[GK00] Thomas Genet and Francis Klay. Rewriting for Cryptographic Protocol Verification. In *CADE'00: 17th International Conference on Automated Deduction*, volume 1831 of *LNCS*, pages 271–290. Springer, 2000.

[GLP05] Jean Goubault-Larrecq and Fabrice Parrennes. Cryptographic Protocol Analysis on Real C Code. In *VMCAI'05: 6th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 3385 of *LNCS*, pages 363–379. Springer, 2005.

[GM99] Juan A. Garay and Philip D. MacKenzie. Abuse-Free Multi-party Contract Signing. In *DISC'99: 13th International Symposium on Distributed Computing*, volume 1693 of *LNCS*, pages 151–165. Springer, 1999.

[GRVS09] Flavio D. Garcia, Peter van Rossum, Roel Verdult, and Ronny Wichers Schreur. Wirelessly Pickpocketing a Mifare Classic Card. In *S&P'09: 30th IEEE Symposium on Security and Privacy*, pages 3–15. IEEE Computer Society, 2009.

[GSG99] Stefanos Gritzalis, Diomidis Spinellis, and Panagiotis Georgiadis. Security protocols over open networks and distributed systems: formal methods for their analysis, design, and verification. *Computer Communications*, 22(8):697–709, 1999.

[Han03] Hans Hüttel. Deciding Framed Bisimilarity. *Electronic Notes in Theoretical Computer Science*, 68(6):1–20, 2003. Special issue Infinity'02: 4th International Workshop on Verification of Infinite-State Systems.

[HBH10] Stuart Haber, Josh Benaloh, and Shai Halevi. The Helios e-Voting Demo for the IACR. International Association for Cryptologic Research. `http://www.iacr.org/elections/eVoting/heliosDemo.pdf`, May 2010.

[HFM88] Debra Hensgen, Raphael Finkel, and Udi Manber. Two Algorithms for Barrier Synchronization. *International Journal of Parallel Programming*, 17(1):1–17, 1988.

[Int09] International Organization for Standardization. *ISO/IEC 11889: Information technology – Trusted Platform Module*, 2009.

[JCJ02] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-Resistant Electronic Elections. Cryptology ePrint Archive, Report 2002/165, 2002.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-Resistant Electronic Elections. In *WPES'05: 4th Workshop on Privacy in the Electronic Society*, pages 61–70. ACM Press, 2005. See also `http://www.rsa.com/rsalabs/node.asp?id=2860`.

[JCJ10] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-Resistant Electronic Elections. In David Chaum, Markus Jakobsson, Ronald L. Rivest, and Peter Y. A. Ryan, editors, *Towards Trustworthy Elections: New Directions in Electronic Voting*, volume 6000 of *LNCS*, pages 37–63. Springer, 2010.

[JJ00] Markus Jakobsson and Ari Juels. Mix and Match: Secure Function Evaluation via Ciphertexts. In *ASIACRYPT'00: 6th International Conference on the Theory and Application of Cryptology and Information Security*, volume 1976 of *LNCS*, pages 162–177. Springer, 2000.

[JMP09] Hugo L. Jonker, Sjouke Mauw, and Jun Pang. Measuring Voter-controlled Privacy. In *ARES'09: 4th International Conference on Availability, Reliability and Security*, pages 289–298. IEEE Computer Society, 2009.

[JMW05] Ari Juels, David Molnar, and David Wagner. Security and Privacy Issues in E-passports. In *SecureComm'05: 1st International Conference on Security and Privacy for Emerging Areas in Communications Networks*, pages 74–88. IEEE Computer Society, 2005.

[Jue06] Ari Juels. RFID Security and Privacy: A Research Survey. *Journal of Selected Areas in Communication*, 24(2):381–395, 2006.

[JV06] Hugo L. Jonker and Erik P. de Vink. Formalising Receipt-Freeness. In *ISC'06: 9th Information Security Conference*, volume 4176 of *LNCS*, pages 476–488. Springer, 2006.

[KJBK09] Karl Koscher, Ari Juels, Vjekoslav Brajkovic, and Tadayoshi Kohno. EPC RFID Tag Security Weaknesses and Defenses: Passport Cards, Enhanced Drivers Licenses, and Beyond. In *CCS'09: 16th ACM Conference on Computer and Communications Security*, pages 33–42. ACM Press, 2009.

[KM06] Neal Koblitz and Alfred Menezes. Another Look at "Provable Security". II. In *INDOCRYPT'06: 7th International Conference on Cryptology*, volume 4329 of *LNCS*, pages 148–175. Springer, 2006.

[KM07] Neal Koblitz and Alfred Menezes. Another Look at "Provable Security". *Journal of Cryptology*, 20(1):3–37, 2007.

[KR05] Steve Kremer and Mark D. Ryan. Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. In *ESOP'05: 14th European Symposium on Programming*, volume 3444 of *LNCS*, pages 186–200. Springer, 2005.

[KRS+03] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: Scalable Secure File Sharing on Untrusted Storage. In *FAST'03: 2nd USENIX Conference on File and Storage Technologies*, pages 29–42. USENIX Association, 2003.

[KRS10] Steve Kremer, Mark D. Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In *ESORICS'10: 15th European Symposium on Research in Computer Security*, volume 6345 of *LNCS*, pages 389–404. Springer, 2010.

[KSR10] Petr Klus, Ben Smyth, and Mark D. Ryan. ProSwapper: Improved equivalence verifier for ProVerif. `http://www.bensmyth.com/proswapper.php`, 2010.

[KT09] Ralf Küsters and Tomasz Truderung. An Epistemic Approach to Coercion-Resistance for Electronic Voting Protocols. In *S&P'09: 30th IEEE Symposium on Security and Privacy*, pages 251–266. IEEE Computer Society, 2009.

[KTV10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. A Game-Based Definition of Coercion-Resistance and its Applications. In *CSF'10: 23rd IEEE Computer Security Foundations Symposium*, pages 122–136. IEEE Computer Society, 2010.

[LCM08] Adrian Leung, Liqun Chen, and Chris J. Mitchell. On a Possible Privacy Flaw in Direct Anonymous Attestation (DAA). In *Trust'08: 1st International Conference on Trusted Computing and Trust in Information Technologies*, number 4968 in LNCS, pages 179–190. Springer, 2008.

[LM07] Adrian Leung and Chris J. Mitchell. Ninja: Non Identity Based, Privacy Preserving Authentication for Ubiquitous Environments. In *Ubicomp'07: 9th International Conference on Ubiquitous Computing*, volume 4717 of *LNCS*, pages 73–90. Springer, 2007.

[Low96] Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In *TACAS'96: 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.

[Low99] Gavin Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(1), 1999.

[LRSW00] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym Systems. In *SAC'99: 6th International Workshop on Selected Areas in Cryptography*, volume 1758 of *LNCS*, pages 184–199. Springer, 2000.

[LTV10] Pascal Lafourcade, Vanessa Terrade, and Sylvain Vigier. Comparison of Cryptographic Verification Tools Dealing with Algebraic Properties. In *FAST'09: 6th International Workshop on Formal Aspects in Security and Trust*, volume 5983 of *LNCS*, pages 173–185. Springer, 2010.

[Lub90] Boris D. Lubachevsky. Synchronization Barrier and Related Tools for Shared Memory Parallel Programming. *International Journal of Parallel Programming*, 19(3):225–250, 1990.

[Lys02] Anna Lysyanskaya. *Signature Schemes and Applications to Cryptographic Protocol Design*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2002.

[Mar08] Andrew Martin. The ten-page introduction to Trusted Computing. Technical Report CS-RR-08-11, University of Oxford, 2008.

[MCB10] Simon Meier, Cas J. F. Cremers, and David A. Basin. Strong Invariants for the Efficient Construction of Machine-Checked Protocol Security Proofs. In *CSF'10: 23rd IEEE Computer Security Foundations Symposium*, pages 231–245. IEEE Computer Society, 2010.

[Mea94] Catherine Meadows. A Model of Computation for the NRL Protocol Analyzer. In *CSFW'94: 7th IEEE Computer Security Foundations Workshop*, pages 84–89. IEEE Computer Society, 1994.

[Mea96a] Catherine Meadows. Analyzing the Needham-Schroeder Public Key Protocol: A Comparison of Two Approaches. In *ESORICS'96: 4th European Symposium on Research in Computer Security*, volume 1146 of *LNCS*, pages 351–364. Springer, 1996.

[Mea96b] Catherine Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[Mea00] Catherine Meadows. Open Issues in Formal Methods for Cryptographic Protocol Analysis. In *DISCEX'00: DARPA Information Survivability Conference & Exposition*, pages 237–250. IEEE Computer Society, 2000.

[Mea01] Catherine Meadows. Open Issues in Formal Methods for Cryptographic Protocol Analysis. In *MMM-ACNS'01: International Workshop on Information Assurance in Computer Networks*, volume 2052 of *LNCS*, page 21. Springer, 2001.

[Mea03] Catherine Meadows. Formal Methods for Cryptographic Protocol Analysis: Emerging Issues and Trends. *Selected Areas in Communications*, 21(1):44–54, 2003.

[Mea04] Catherine Meadows. Ordering from Satan's menu: a survey of requirements specification for formal analysis of cryptographic protocols. *Science of Computer Programming*, 50(1–3):3–22, 2004.

[Mer02] Rebecca Mercuri. A Better Ballot Box? *IEEE Spectrum*, 39(10):46–50, 2002.

[MGR09] Aybek Mukhamedov, Andrew D. Gordon, and Mark D. Ryan. Towards a Verified Reference Implementation of a Trusted Platform Module. In *17th International Workshop on Security Protocols*, LNCS. Springer, 2009. To appear.

[Min08] Ministerie van Binnenlandse Zaken en Koninkrijksrelaties (Netherland's Ministry of the Interior and Kingdom Relations). Stemmen met potlood en papier (Voting with pencil and paper). Press release `http://www.minbzk.nl/onderwerpen/grondwet-en/verkiezingen/nieuws--en/112441/stemmen-met-potlood`, May 2008.

[MMS97] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated Analysis of Cryptographic Protocols using Murφ. In *S&P'97: 18th IEEE Symposium on Security and Privacy*, pages 141–151. IEEE Computer Society, 1997.

[MN06] Tal Moran and Moni Naor. Receipt-Free Universally-Verifiable Voting with Everlasting Privacy. In *CRYPTO'06: 26th International Cryptology Conference*, volume 4117 of *LNCS*, pages 373–392. Springer, 2006.

[Mon99] David Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *SAS'99: 6th International Static Analysis Symposium*, volume 1694 of *LNCS*, pages 149–163. Springer, 1999.

[Mon03] David Monniaux. Abstracting cryptographic protocols with tree automata. *Science of Computer Programming*, 47(2–3):177–202, 2003.

[Moo88] Judy H. Moore. Protocol Failures in Cryptosystems. In *Proceedings of the IEEE*, volume 76, pages 594–602. IEEE Computer Society, 1988.

[MPW92a] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, I. *Information and Computation*, 100:1–40, 1992.

[MPW92b] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, II. *Information and Computation*, 100:41–77, 1992.

[MR08] Aybek Mukhamedov and Mark Dermot Ryan. Fair multi-party contract signing using private contract signatures. *Information and Computation*, 206(2–4):272–290, 2008.

[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, 1978.

[Oka98] Tatsuaki Okamoto. Receipt-Free Electronic Voting Schemes for Large Scale Elections. In *SP'97: 5th International Workshop on Security Protocols*, volume 1361 of *LNCS*, pages 25–35. Springer, 1998.

[Pau98] Lawrence C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.

[PBP+10] Reema Patel, Bhavesh Borisaniya, Avi Patel, Dhiren Patel, Muttukrishnan Rajarajan, and Andrea Zisman. Comparative Analysis of Formal Model Checking Tools for Security Protocol Verification. In Natarajan Meghanathan, Selma Boumerdassi, Nabendu Chaki, and Dhinaharan Nagamalai, editors, *Recent Trends in Network Security and Applications*, volume 89 of *Communications in Computer and Information Science*, pages 152–163. Springer, 2010.

[PH10] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management (Version 0.34). Technical report, Department of Computer Science, Technische Universität Dresden, 2010. See `http://dud.inf.tu-dresden.de/Anon_Terminology.shtml` for succeeding versions of this report.

[Pie10] Wolter Pieters. Verifiability of Electronic Voting: Between Confidence and Trust. In Serge Gutwirth, Yves Poullet, and Paul De Hert, editors, *Data Protection in a Profiled World*, chapter 9. Springer, 2010.

[PK01] Andreas Pfitzmann and Marit Köhntopp. Anonymity, Unobservability, and Pseudonymity A Proposal for Terminology. In *International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, pages 1–9. Springer, 2001. An extended version is also available [PH10].

[Pri10] Princeton University. *Princeton Election Server*, 2010. `https://princeton-helios.appspot.com/`.

[PS96] David Pointcheval and Jacques Stern. Security Proofs for Signature Schemes. In *EUROCRYPT'96: 13th International Conference on the Theory and Applications of Cryptographic Techniques*, volume 1070 of *LNCS*, pages 387–398. Springer, 1996.

[PSW00] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Cryptographic Security of Reactive Systems. *Electronic Notes in Theoretical Computer Science*, 32:59–77, 2000.

[Ros95] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *CSFW'95: 8th IEEE Computer Security Foundations Workshop*, pages 98–107. IEEE Computer Society, 1995.

[RR98] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[RS11] Mark D. Ryan and Ben Smyth. Applied pi calculus. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, chapter 6. IOS Press, 2011.

[RT01] Michaël Rusinowitch and Mathieu Turuani. Protocol Insecurity with Finite Number of Sessions is NP-Complete. In *CSFW'01: 14th IEEE Computer Security Foundations Workshop*, pages 174–188. IEEE Computer Society, 2001.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions and composed keys is np-complete. *Theoretical Computer Science*, 1-3(299):451–475, 2003.

[Rub02] Aviel D. Rubin. Security considerations for remote electronic voting. *Communications of the ACM*, 45(12):39–44, 2002.

[Rud07] Carsten Rudolph. Covert Identity Information in Direct Anonymous Attestation (DAA). In *SEC'07: 22nd International Information Security Conference*, volume 232 of *International Federation for Information Processing (IFIP)*, pages 443–448. Springer, 2007.

[Rus93] John Rushby. Formal Methods and the Certification of Critical Systems. Technical Report SRI-CSL-93-7, SRI International, 1993.

[Sal88] Roy G. Saltman. Accuracy, Integrity and Security in Computerized Vote-Tallying. *Communications of the ACM*, 31(10):1184–1191, 1988.

[San93] Davide Sangiorgi. A Theory of Bisimulation for the π-Calculus. In *CONCUR'93: 4th International Conference on Concurrency Theory*, volume 715 of *LNCS*, pages 127–142. Springer, 1993.

[San96] Davide Sangiorgi. A Theory of Bisimulation for the π-Calculus. *Acta Informatica*, 33(1):69–97, 1996.

[SBP01] Dawn Xiaodong Song, Sergey Berezin, and Adrian Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1–2):47–74, 2001.

[Sch09] Berry Schoenmakers. Voting Schemes. In Mikhail J. Atallah and Marina Blanton, editors, *Algorithms and Theory of Computation Handbook, Second Edition, Volume 2: Special Topics and Techniques*, chapter 15. CRC Press, 2009.

[SGPV09] Michaël Sterckx, Benedikt Gierlichs, Bart Preneel, and Ingrid Verbauwhede. Efficient implementation of anonymous credentials on Java Card smart cards. In *WIFS'09: IEEE International Workshop on Information Forensics and Security*, pages 106–110. IEEE Computer Society, 2009.

[Sha49] Claude E Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28:656–715, 1949.

[Smy06] Ben Smyth. Direct Anonymous Attestation (DAA): An implementation and security analysis. Master's thesis, School of Computer Science, University of Birmingham, 2006.

[Smy07] Ben Smyth. Automatic verification of privacy properties in the applied pi calculus. In Liqun Chen, Steve Kremer, and Mark D. Ryan, editors, *Formal Protocol Verification Applied: Abstracts Collection*, number 07421 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007. `http://drops.dagstuhl.de/opus/volltexte/2008/1419/pdf/07421_abstracts_collection.1419.pdf`.

[Son99] Dawn Xiaodong Song. Athena: a New Efficient Automatic Checker for Security Protocol Analysis. In *CSFW'99: 12th IEEE Computer Security Foundations Workshop*, pages 192–202. IEEE Computer Society, 1999.

[SRC07] Ben Smyth, Mark D. Ryan, and Liqun Chen. Direct Anonymous Attestation (DAA): Ensuring privacy with corrupt administrators. In *ESAS'07: 4th European Workshop on Security and Privacy in Ad hoc and Sensor Networks*, volume 4572 of *LNCS*, pages 218–231. Springer, 2007.

[SRKK10] Ben Smyth, Mark D. Ryan, Steve Kremer, and Mounira Kourjieh. Towards automatic analysis of election verifiability properties. In *ARSPA-WITS'10: Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*, volume 6186 of *LNCS*, pages 165–182. Springer, 2010.

[SS08] Mario Strasser and Heiko Stamer. A Software-Based Trusted Platform Module Emulator. In *Trust'08: 1st International Conference on Trusted Computing and Trust in Information Technologies*, volume 4968 of *LNCS*, pages 33–47. Springer, 2008.

[Tar10] Christopher Tarnovsky. Deconstructing a 'Secure' Processor. In *Black Hat DC 2010*, 2010. `https://media.blackhat.com/bh-dc-10/video/Tarnovsky_Chris/BlackHat-DC-2010-Tarnovsky-DeconstructProcessor-video.m4v`.

[TCG07] Trusted Computing Group. *TPM Specification version 1.2*, 2007. `http://www.trustedcomputinggroup.org/specs/TPM/`.

[THG98] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand Spaces: Why is a Security Protocol Correct? In *S&P'98: 19th IEEE Symposium on Security and Privacy*, pages 160–171. IEEE Computer Society, 1998.

[THG99] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security*, 7(1), 1999.

[TMT+08] Mehdi Talbi, Benjamin Morin, Valérie Viet Triem Tong, Adel Bouhoula, and Mohamed Mejri. Specification of Electronic Voting Protocol Properties Using ADM Logic: FOO Case Study. In *ICICS'08: 10th International Conference on Information and Communications Security*, volume 5308 of *LNCS*, pages 403–418. Springer, 2008.

[Tru09] Trusted Computing Group. *Cloud Computing and Security - A Natural Match*, 2009. `http://www.trustedcomputinggroup.org/resources/cloud_computing_and_security__a_natural_match`.

[Tur06] Mathieu Turuani. The CL-Atse Protocol Analyser. In *RTA'06: 17th International Conference on Rewriting Techniques and Applications*, volume 4098 of *LNCS*, pages 277–286. Springer, 2006.

[UK07] *Key issues and conclusions: May 2007 electoral pilot schemes*, May 2007. `http://www.electoralcommission.org.uk/elections/pilots/May2007`.

[UMQ10] Dominique Unruh and Jörn Müller-Quade. Universally Composable Incoercibility. In *CRYPTO'10: 30th International Cryptology Conference*, volume 6223 of *LNCS*, pages 411–428. Springer, 2010.

[Vic94] Björn Victor. *A Verification Tool for the Polyadic π-Calculus*. PhD thesis, Department of Computer Systems, Uppsala University, Sweden, 1994.

[VM94] Björn Victor and Faron Moller. The Mobility Workbench - A Tool for the π-Calculus. In *CAV'94: 6th International Conference on Computer Aided Verification*, volume 818 of *LNCS*, pages 428–440. Springer, 1994.

[War03] Bogdan Warinschi. A Computational Analysis of the Needham-Schröeder-(Lowe) Protocol. In *CSFW'03: 16th IEEE Computer Security Foundations Workshop*, pages 248–262. IEEE Computer Society, 2003.

[War05] Bogdan Warinschi. A computational analysis of the Needham-Schroeder-(Lowe) protocol. *Journal of Computer Security*, 13(3):565–591, 2005.

[Wei99] Christoph Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *CADE'99: 16th International Conference on Automated Deduction*, volume 1632 of *LNCS*, pages 314–328. Springer, 1999.