Discovering Attacks on Security Protocols by Refuting Incorrect Inductive Conjectures

Graham J. Steel

Doctor of Philosophy

Centre for Intelligent Systems and their Applications

School of Informatics

University of Edinburgh

2003

(Graduation date: July 7th, 2004)

Abstract

Cryptographic security protocols are used to allow agents to communicate securely over an insecure network. Although security protocols are usually quite short, they often have subtle flaws in them which allow the protocol to be ‘attacked’ and security breached.

Interactive inductive theorem proving has been used to verify properties of security protocols. However, trying to verify a flawed protocol will result in an attempt to prove an incorrect conjecture. A user might waste a lot of time proposing generalisations and lemmas etc. in a futile attempt to prove a falsehood. In addition, even if he suspects the protocol is flawed, it can be extremely difficult to find the attack (the sequence of messages needed to expose the flaw). What is required is an automated tool which can not only detect incorrect inductive conjectures, but also present a counterexample.

This thesis describes the development of such a tool, CORAL, based on the refutation complete Comon-Nieuwenhuis method for ‘proof by consistency’. We describe the testing of CORAL on some standard protocols known to be flawed, and two case studies on new protocols, in which CORAL discovered five previously unknown attacks. CORAL does not find attacks as fast as some competing approaches, but in its successful modelling of two very different group protocols, shows a flexibility other systems lack. This should make it suitable for a variety of future developments, including the investigation of some more unusual protocols.

Acknowledgements

Firstly, I would like to thank my supervisor, Prof. Alan Bundy for his patient and expert guidance throughout the course of the research described in this thesis. I would also like to thank my second supervisor, Dr. Monika Maidl, and my former second supervisors, Dr. Ewen Denney and Dr. Louise Dennis. In particular, Dr. Maidl's thorough and thoughtful reading of my first draft contributed greatly to the quality of the final thesis.

I am grateful to my examiners, Prof. David Basin and Dr. Jacques Fleuriot, for their rigorous criticism and considered suggestions for improvements to the thesis.

I would like to thank my family for their support, particularly during the protracted period of illness that interrupted my Ph.D. research. Without their help and understanding, I certainly would not have been able to finish this thesis.

Finally, thanks are due to the other members of the *DREAM* group for their encouragement, criticism and good company over the occasional beer.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Graham J. Steel*)

Publications

Some of the work in this thesis has previously been published, in [Steel et al., 2002a, Steel et al., 2002b, Steel et al., 2003a, Steel et al., 2003b].

Table of Contents

1 Introduction 1

1.1 Results . . . 3

1.2 Contribution of this Thesis . . . 4

1.3 Layout of this Thesis . . . 4

2 Cryptographic Security Protocols 7

2.1 General Principles . . . 7

2.2 A Simple Example: Needham-Schroeder Public Key . . . 9

2.3 Protocol Attacks . . . 10

2.3.1 Protocol Goals . . . 11

2.3.2 The Spy . . . 11

2.3.3 Lowe's attack on Needham-Schroeder Public Key . . . 12

2.3.4 Needham-Schroeder Shared Key . . . 13

2.3.5 The Otway-Rees Protocol . . . 15

2.3.6 The Neuman-Stubblebine Protocol . . . 18

2.4 Formal Methods for Protocol Analysis . . . 19

2.4.1 State Exploration Approaches . . . 20

2.4.2 The BAN logic . . . 22

2.4.3 The Spi Calculus . . . 23

2.4.4 Theorem Proving . . . 23

2.4.5 The Strand Space Model . . . 27

2.5 Outlook . . . 29

3 Refuting Incorrect Conjectures **33**

3.1 First-Order Finite Domain Enumerators 33

3.2 Finding Counterexamples by Instantiating with Constructors 34

3.2.1 Protzen's Calculus for Refutation 34

3.2.2 Reif's Counterexample Finder 35

3.2.3 Evaluation of Protzen's and Reif's Approaches 36

3.3 Monroy's Non-theorem Work 37

3.4 Proof by Consistency 38

3.5 Evaluation 40

4 The Theory of the Comon-Nieuwenhuis Method **41**

4.1 Background 41

4.2 First-Order Theorem Proving 43

4.2.1 Modern Provers 45

4.2.2 Memory Allocation 48

4.2.3 Answer Extraction 49

4.3 Overview of Comon-Nieuwenhuis Method 50

4.4 The Comon-Nieuwenhuis Method for Horn Clauses 50

4.5 Non-saturated Sets of Axioms 57

4.6 Finding I-Axiomatisations 59

4.7 Summary 61

5 System Description **63**

5.1 Adapting a First-Order Prover 63

5.1.1 Choosing a First-Order Prover 63

5.1.2 Separation of Axioms and Conjectures 64

5.1.3 The Use of Lemmas in Reduction Rules 65

5.1.4 Recovery of the Counterexample 66

5.2 I-Axiomatisation Checking 66

5.3 Testing 68

5.4 Summary 71

6 Formalisation of the Protocol Veri£cation Problem **73**

6.1 The Nature of the Paulson Model 73
6.2 Example - Needham-Schroeder Public Key 74
6.3 Free Constructors, Deciding Equivalence 74
6.4 Principals, Keys and Nonces 76
6.5 Protocol Messages 78
6.6 Modelling Intruder Knowledge 80
6.7 Term Ordering 81
6.8 Refutation Completeness 83
6.9 I-Axiomatisation 85
6.10 Comparison between Paulson's Formalism and CORAL's Formalism . 85
6.11 Formulating Conjectures in the Formalism 87
6.12 Summary 88

7 Optimisations **89**

7.1 Elimination of Invalid Terms 89
7.2 Restricting the Spy's Messages 90
7.2.1 Spy Only Sends Protocol Messages 90
7.2.2 Spy Only Expects Protocol Messages 91
7.2.3 Spy Only Expects Subterms from Protocol Messages 92
7.2.4 No Two Spy Messages in a Row 92
7.3 Eager Elimination of Unsatisfiable *parts* Literals 93
7.4 Literal Selection 93
7.5 Summary 94

8 Rediscovering Known Attacks **95**

8.1 The Development Set of Protocols 95
8.2 Attacking Needham-Schroeder Public Key 96
8.2.1 How CORAL Finds the NSPK Attack 97
8.3 Attacking a Clark-Jacob Protocol 99
8.4 Attacking Neuman-Stubblebine 101
8.5 Attacking BAN Otway-Rees 103

8.6 Development of Heuristics 105
8.7 The Test Set of Protocols 107
8.8 Results 107
8.9 Summary 109

9 Case Study 1:
The Asokan–Ginzboorg Protocol 111
9.1 Description of the Asokan–Ginzboorg Protocol 111
9.2 Modelling the Protocol 114
9.2.1 Modelling Message 3 114
9.2.2 Ordering Considerations 117
9.3 Modelling Spies in a Wireless Network 118
9.4 Attacking the Protocol 119
9.4.1 Attacks by a Spy Outside the Room 120
9.4.2 Attacks by a Spy Inside the Room 122
9.5 An Improved Version of the Asokan–Ginzboorg Protocol 123
9.6 Summary 124

10 Case Study 2:
The Tanaka–Sato Protocol 127
10.1 Description of the Tanaka–Sato Protocol
(Taghdiri–Jackson version) 127
10.1.1 Commentary 130
10.2 Modelling the Protocol 131
10.3 Attacking the Protocol 133
10.4 An Improved Version of the Protocol 136
10.5 Summary 137

11 Related Work 139
11.1 Athena 140
11.1.1 Description of Athena 140
11.1.2 Results 142
11.1.3 Summary 144

11.2 On-The-Fly Model Checker 145
11.2.1 Description of the OFMC 145
11.2.2 Results 146
11.2.3 Summary 147
11.3 The Casrul System 147
11.4 Other Work on Group Protocols 149
11.5 Paulson's Inductive Approach 151
11.6 Weidenbach's First-Order Formalism 153
11.7 General Inductive Refutation Tools 154
11.8 Summary 155
11.8.1 Ultimate Applicability of the Technique 156
11.8.2 The Way Ahead 156

12 Further Work **159**
12.1 Improving CORAL as a Protocol Analysis Tool 159
12.2 Further Protocol Experiments 161
12.3 Other Encryption Models 162
12.4 Proving Theorems 162
12.5 Other Application Areas 163
12.6 Summary 164

13 Conclusions **165**
13.1 Evaluation of Research Contributions 165
13.2 Final Summary 167

A Protocol Model Files **169**
A.1 The Needham-Schroeder Public Key Protocol 169
A.2 Changes for the Clark-Jacob Protocol 173
A.3 Changes for the Neuman-Stubblebine Protocol 174
A.4 Changes for the Otway-Rees Protocol 178

B The Model for the Asokan–Ginzboorg Protocol **185**

C The Model for the Tanaka–Sato Protocol **189**

D Index of Terms **193**

Bibliography **197**

List of Figures

2.1 Encryption and Decryption 8
2.2 Needham-Schroeder Public Key in a Strand Space Model 28
2.3 Needham-Schroeder Public Key Attacked 28

5.1 CORAL System Operation 68
5.2 CORAL's Results on Non-theorem Examples 70

9.1 Clauses for modelling message 3 115

10.1 Clauses for modelling the 'send' sub-protocol 133

List of Tables

5.1 Results on Non-theorems from the Literature 72

8.1 The Pattern of Search for the NSPK Attack 98
8.2 Profile of CORAL's reduction time 106
8.3 CORAL Attacking Protocols from the Clark-Jacob corpus 110

Chapter 1

Introduction

Inductive theorem provers are frequently employed in the verification of programs, algorithms and protocols. The use of induction allows us to reason about structures with infinite models, e.g. a program with loops in, a recursive function or a protocol that may involve an arbitrary number of participants. However, programs and algorithms often contain bugs, and protocols may be flawed, causing the proof attempt to fail. It can be hard, even for expert users, to interpret a failed proof attempt: is it the proof attempt or the conjecture under investigation which is at fault? In this situation, what is required is an automated tool which can not only detect an incorrect conjecture, but also supply a counterexample to allow the user to identify the flaw.

The aim of cryptographic security protocols is to prescribe a way in which users may communicate securely over an insecure network. A protocol describes an exchange of messages in which the principals involved establish shared secrets, in order perhaps to communicate privately or to protect themselves from impersonators. These protocols are designed to be secure even in the presence of an active attacker, who may intercept or delay messages and send faked messages in order to gain access to secrets. Unsurprisingly, given this hostile operating environment, they have proven very hard to get right. What's more, protocol flaws are often quite subtle. New attacks are often found on protocols many years after they were first proposed.

The work presented in this thesis represents the combination of these two areas of research. L.C. Paulson has proposed a method for verifying the correctness of security protocols using inductive theorem proving, [Paulson, 1998]. This method is attractive

in that it tries to prove security properties in the context of an arbitrary number of possible participants and parallel runs of the protocol. Hence, if a proof of correctness can be found, it provides a strong guarantee that the protocol is, in fact, secure. Paulson's method can also be used to prove properties of group protocols, where an arbitrary number of participants may be involved in a single round. Its simplicity also makes it easy to adapt to different kinds of protocols, for example where elapsed time may have to be considered, or where we may be interested in properties such as plausible deniability. However, finding a proof is a challenging task, requiring some considerable expertise. This means that a failed proof attempt is especially hard to interpret. Additionally, in order to convincingly show a protocol is flawed, it is not sufficient merely to cite a failed proof attempt. Rather, it is necessary to exhibit a sequence of messages leading to some compromise of security. This exchange constitutes a counterexample to the security property. So this would seem to be an area in which an automated counterexample finder for inductive conjectures would be of particular value.

The method for finding counterexamples we investigate in this thesis is based on a technique for inductive proof, called *proof by consistency*. This method was first proposed in the 1980s, [Musser, 1980]. The idea is to show that a conjecture is a theorem by proving consistency with the axioms in the intended semantics. Later versions of the technique had the property of being *refutation complete*, i.e. in addition to be able to show some theorems correct, it can refute any false conjecture in finite time. Recently, Comon and Nieuwenhuis have proposed a new version of the technique allowing any automatic first-order theorem prover to be used to search for proofs and refutations, [Comon and Nieuwenhuis, 2000]. Our work is based on their ideas. We have implemented their strategy in our system called CORAL, employing the first-order theorem prover SPASS, [Weidenbach et al., 1999], and have used it to find attacks on a number of protocols, including 3 previously unknown attacks on a protocol for ad-hoc wireless networks, [Asokan and Ginzboorg, 2000], and 2 new attacks on a multicast key management protocol, [Taghdiri and Jackson, 2003].

1.1 Results

CORAL has rediscovered a number of known attacks on faulty protocols, as described in Chapter 8. CORAL's heuristics were initially developed on a set of four protocols, before the tool was applied to a 'test set' of ten protocols from Clark and Jacob's corpus, [Clark and Jacob, 1997]. CORAL found all ten attacks.

More interestingly, CORAL has discovered several previously undiscovered attacks. Three were found on the Asokan–Ginzboorg protocol for establishing a secure session amongst a set of Bluetooth-enabled laptops, [Asokan and Ginzboorg, 2000]. This is especially significant because the protocol is designed to allow an arbitrary number of participants to take part. Modelling this protocol in a general way, i.e. without deciding on the size of the group in advance, is something that we can do in CORAL because of its inductive model. Most other techniques for finding attacks cannot do this. This enabled us to find attacks on the protocol for groups of size 2 and 3. The details of our case study on the Asokan–Ginzboorg protocol are in Chapter 9.

CORAL has also been used to discover two new attacks on a group multicast key management protocol originally proposed by Tanaka and Sato, [Tanaka and Sato, 2001], and then improved by Taghdiri and Jackson, [Taghdiri and Jackson, 2003]. This protocol consists of a suite of sub-protocols which an agent may take part in an unbounded number of times. Again, our inductive approach meant that we could model the improved protocol without difficulty. Some minor additions to the formalism were required, but the simplicity of our approach meant that this was easily done. This case study is described in Chapter 10.

Additionally, we have used CORAL to refute some incorrect conjectures from other inductive theorem proving case studies. These results are given at the end of Chapter 5.

1.2 Contribution of this Thesis

The contribution of this thesis is an investigation of two hypotheses:

Hypothesis 1 By using the Comon-Nieuwenhuis strategy to refute incorrect inductive conjectures in a first-order version of Paulson's security protocol model, we can effectively find attacks on faulty security protocols.

Hypothesis 2 The use of a simple first-order trace based formalism for analysing protocols allows us to quickly adapt to unusual protocols, such as group key agreement and key management protocols, which approaches optimised for standard 2 and 3 party protocols would struggle with.

Evidence to support the first hypothesis is given by the results of testing on ten protocols from the standard corpus at the end of Chapter 8. Given that there are many other tools that can also find protocol attacks, the second hypothesis is more significant. Evidence to support the second hypothesis is given in the form of two case studies, the first in Chapter 9, where 3 new attacks were discovered, and the second in Chapter 10, where 2 new attacks were found. A comparison with other approaches is made in Chapter 11. Here we will see that in terms of discovering previously unknown attacks on group protocols, CORAL is currently the leading tool.

1.3 Layout of this Thesis

Chapter 2 introduces cryptographic security protocols, gives some examples of how these protocols have been attacked, and describes other attempts to analyse security protocols using formal methods.

Chapter 3 surveys previous work on automated refutation of incorrect conjectures.

Chapter 4 explains the theory of first-order theorem proving and the Comon-Nieuwenhuis method for proof by consistency.

Chapter 5 describes the implementation of the Comon-Nieuwenhuis technique in the CORAL system.

Chapter 6 is a description of our first-order version of Paulson's inductive model for security protocol analysis.

Chapter 7 gives the additional heuristics we used to enable CORAL to find security protocol attacks in reasonable time.

Chapter 8 shows how we used CORAL to rediscover some known attacks on a number of security protocols.

Chapter 9 contains the details of our case study on the Asokan–Ginzboorg protocol for key agreement in an ad-hoc network of Bluetooth enabled laptops. Three new attacks are given.

Chapter 10 documents our case study on the Tanaka–Sato protocol for group multicast key management. Two new attacks are presented.

Chapter 11 compares CORAL to related work, both in the field of security protocol analysis, and refuting incorrect inductive conjectures.

Chapter 12 suggests possible further work.

Chapter 13 summarises the thesis and draws conclusions.

Four appendices complete the thesis:

Appendix A gives the specification files used for two protocols involving a fixed number of parties: the Needham-Schroeder Public Key and Otway-Rees protocols.

Appendix B gives the specification file used for a group protocol, the Asokan–Ginzboorg protocol.

Appendix C contains the specification file used for the Tanaka–Sato multicast key management protocol.

Appendix D is an index of technical terms.

Chapter 2

Cryptographic Security Protocols

Cryptographic protocols are used in distributed systems to allow agents to communicate securely. A protocol specifies an exchange of messages between some honest users (the first two of which are by convention called Alice and Bob) and (possibly) a secure server. Protocols are required to be secure in the presence of a spy, who can see all the traffic in the network and may send malicious messages in order to try and impersonate users and gain access to secrets.

Although security protocols typically describe an exchange of just 2 to 5 messages, they have proven to be extremely difficult to get right. Subtle ¤aws are often found in them after they have been in use for years. In this chapter, we examine the general problem setting, and look at some proposed protocols. We will see how attacks have been found and how these have suggested modifications. We will also survey some attempts at formal protocol analysis.

2.1 General Principles

We assume that Alice and Bob have access to a cryptographic algorithm functioning as described in Figure 2.1. Suppose Alice has a message P she wishes to transmit across the network securely. The algorithm converts P from *plaintext* into *ciphertext*, a form which is unintelligible to anyone monitoring the network. This process is called *encryption*. The exact form of the encrypted text depends on the key K which Alice

uses. In order for Bob to be able to recover the original message from the ciphertext, a process called *decryption*, he must use a second key K^{-1}. So, the secrecy of messages depends on being able to control access to keys.

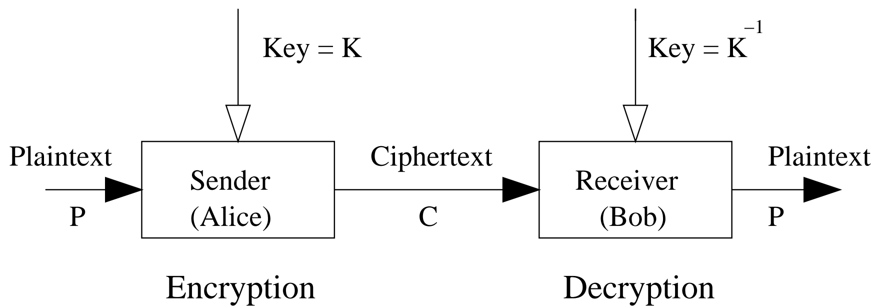

Figure 2.1: Encryption and Decryption, [Clark and Jacob, 1996]

There are two schemes for organising encryption and decryption keys. In *symmetric key cryptography*, the decryption key K^{-1} and the encryption key K are easily obtainable from each other by public techniques. Often, they are identical, and we generally assume this to be the case. The best known algorithm for symmetric key encryption is probably the Digital Encryption Standard (DES), [FIPS, 1977]. In a symmetric key scheme, each pair of principals will have their own key for communicating with each other. We use K_{AB} to denote the key used for communication between principals A and B. To minimize the damage done if a key is lost to a spy, each key has a limited lifespan, after which it is no longer used. This means that principals will occasionally need to communicate to set up a new short term key (also called a *session key*). However, they cannot use their old session keys to set up a new one, as this could allow a spy who has obtained the old key to obtain the new one as well. Setting up a new session key is one application of security protocols.

In *public key cryptography*, each principal A has a public key, $pubK_A$, known to all users of the network, and a private key $priK_A$, known only to herself. Knowledge of a user's public key provides no help in finding her private key. All users of the network can encrypt messages for A, and these messages will then be unintelligible to everyone except A. The best-known public key cryptographic algorithm is the RSA algorithm, [Rivest et al., 1978]. Under many public key schemes, including RSA, public keys

and private keys can be used for both encryption and decryption. So, a principal A can encrypt her transmission with her own private key, $priK_A$. This creates a message readable by everyone (they can decrypt it using A's public key), with the property that it must have been written by someone in possession of A's private key. A disadvantage is that public key encryption and decryption are generally much slower than shared key operations for an equivalent[1] key length.

2.2 A Simple Example: Needham-Schroeder Public Key

Cryptographic security protocols were first proposed by Needham and Schroeder, [Needham and Schroeder, 1978]. Two different protocols for interactive communication were proposed, one using shared key encryption and one using public key encryption. We use the so-called Needham-Schroeder Public Key (NSPK) protocol as our first concrete example of a security protocol. NSPK has become the standard example in the protocol literature. It actually consists of seven messages, but four of these just involve the principals obtaining each other's public keys. Attention is usually restricted to just the three messages we give here. First, we introduce some more notation[2]. We denote Alice sending a message to Bob encrypted with Bob's public key by writing

$$A \rightarrow B: \{\!|\, message \,|\!\}_{pubK_B}$$

The *message* consists of a number of items which may be agent identifiers (like A, B or S for the server), keys, or unique identifying numbers called nonces[3] which we subscript with the agent who generated them (e.g. N_A, N_B etc.). The aim of the NSPK protocol is to establish authenticity – this is pertinent because everyone is assumed to know everyone else's public keys, so an intruder could easily send a message pretending to be someone else. In order to prevent this, Alice and Bob exchange nonces in a manner designed to establish mutual authenticity. The protocol runs like this:

[1] The concept of equivalent key lengths for shared and private key encryption is a slightly informal one, but an equivalence of sorts is generally accepted.

[2] Our protocol notation follows that in [Paulson, 1998], which is based on the notation Needham and Schroeder introduced in [Needham and Schroeder, 1978].

[3] Nonce is short for 'oNly used ONCE' - i.e. a number which once used in a full protocol exchange is never used again.

1. $A \rightarrow B : \{\!|\, N_A, A \,|\!\}_{pubK_B}$

2. $B \rightarrow A : \{\!|\, N_A, N_B \,|\!\}_{pubK_A}$

3. $A \rightarrow B : \{\!|\, N_B \,|\!\}_{pubK_B}$

This can be described informally as follows:

1. Alice initiates a session with Bob by sending him a new nonce, N_A.

2. Bob decrypts the package from Alice, and sends her back both the nonce Alice sent him, N_A, and a new nonce he has generated, N_B.

3. Alice decrypts that package from Bob, and checks that nonce N_A is indeed the nonce she generated to start the run. Only Bob can have returned the same nonce encrypted with her key, as only he could have read message 1. Alice responds by sending Bob back his nonce, N_B. Bob will receive this and assume that it must have come from Alice, as only she could have decrypted message 2.

Alice and Bob can now use nonces N_A and N_B to sign their messages to each other. This is designed not only to establish authenticity, but also *freshness*, i.e. if Bob receives a message signed with nonce N_A, he believes it not only to be from Alice, but also to have been sent after the protocol run establishing the nonce took place. Without this safeguard, a spy may intercept messages and then replay them later, with potentially serious consequences (e.g. if the message is 'enemy will attack in two days time').

This looks simple enough: the idea of accepting a nonce based on the fact that only the intended recipient could have read the original transmission seems to be a sound one. But this protocol is ¤awed, as we demonstrate in §2.3.3

2.3 Protocol Attacks

To understand protocol attacks, we must first understand what properties we would like our protocols to have, and also what we expect an intruder in the network to be able to do.

2.3.1 Protocol Goals

As new communication scenarios arise, the properties we would like our protocols to establish are constantly updated. However, there are a number of common objectives that we often want to achieve:

- *Authenticity*, that a message supposedly from principal A really is from principal A, and often, additionally, the setting up of a nonce that can be used to establish authenticity of future messages.
- *Secrecy*, that certain parts of messages broadcast over the network are only readable by their intended recipients.
- *Non-repudiation*, that a principal may not plausibly deny sending a particular message.

The exact meaning of these high level goals (and in particular authenticity) are open to a certain amount of interpretation. Some authors have worked on translating them into more formal language and have highlighted the importance of being precise about what a protocol is supposed to do, [Lowe, 1997, Gollmann, 2000]. The attacks presented below will illustrate some of these issues.

2.3.2 The Spy

Mention is made in [Needham and Schroeder, 1978] of the assumptions made about the behaviour of the principals and the spy. This is an issue central to work on cryptographic protocols. Needham and Schroeder made the following assumptions, which have been broadly accepted by the computer security community and are used either implicitly or explicitly in the vast majority of the literature:

- Keys used in cryptographic protocols are not readily discoverable by exhaustive search or cryptanalysis.
- The spy can interpose a computer in all communication paths. This means that in addition to being able to see all traffic in the network, the spy can delay messages

or prevent them from ever reaching their intended recipient, and can add fake messages of his own.

- Principals in the network other than the spy are trying to communicate securely. Security protocols do not attempt to force all communication to be carried out in a secure fashion.

In [Dolev and Yao, 1983], Dolev and Yao formalised a model of a spy with these abilities, adding the further assumptions:

- The spy can break down messages he has seen in traffic into component nonces, agent identifiers etc, and build new fake messages out of these parts.
- The spy can forward packages he cannot read (because they are encrypted).

Later work usually also assumes:

- The spy knows the long term or private key of one agent[4]. Usually we just assume that the spy himself is accepted as an honest agent by the other principals, and has long term keys of his own. This is the same thing in practice as the spy 'stealing the identity' of an honest agent, i.e. obtaining an honest agent's private keys.

This model of an intruder became known as 'the Dolev-Yao intruder', and is used in almost all research in cryptographic protocol analysis.

2.3.3 Lowe's attack on Needham-Schroeder Public Key

Here is an example of how such a spy may attack a protocol. This attack was found by Gavin Lowe in 1995, [Lowe, 1995], on the NSPK protocol presented above (§2.2). Here, A attempts to start a run with the dishonest agent C. This is plausible under the above assumption that the spy is accepted as an honest agent by the other principals. Then, C starts a parallel run with B (marked with $'$s on the message numbers), in which

[4] Originally the spy was assumed to hold the long term keys of an arbitrary number of agents, but it has since been shown that a spy holding the keys of just one agent is just as powerful, [Syverson et al., 2000].

he masquerades as A, using A as an oracle to decrypt B's message $2'$ in messages 2 and 3. The attack runs as follows:

$$
\begin{array}{lcccl}
1. & A & \rightarrow & C & : \{\!| N_A, A |\!\}_{pubK_C} \\
1'. & C_A & \rightarrow & B & : \{\!| N_A, A |\!\}_{pubK_B} \\
2'. & B & \rightarrow & C_A & : \{\!| N_A, N_B |\!\}_{pubK_A} \\
2. & C & \rightarrow & A & : \{\!| N_A, N_B |\!\}_{pubK_A} \\
3. & A & \rightarrow & C & : \{\!| N_B |\!\}_{pubK_C} \\
3.' & C_A & \rightarrow & B & : \{\!| N_B |\!\}_{pubK_B}
\end{array}
$$

We use C_A to indicate that the spy C is playing Alice's part in the protocol. At the end of this sequence of events, Bob believes he has carried out a complete run of the protocol with Alice, when in fact he has not. The intruder C can use the nonces N_A and N_B to impersonate A to B. Lowe suggests a scenario where B is a bank and the spy sends the faked message:

$$C_A \rightarrow B : \{\!| N_A, N_B, \text{transfer £1000 from my account to C's} |\!\}_{pubK_B}$$

Like many published protocol ¤aws, this attack caused some controversy. Needham argued that the protocol was never intended to be used in situation where a spy may be accepted as an honest player, and certainly not in the banking scenario suggested by Lowe. However, others argued it was perfectly valid to show the limitations of the protocol, and that the acceptance of the spy as an honest player is a possibility that must always be considered.

2.3.4 Needham-Schroeder Shared Key

The shared key version of the Needham-Schroeder protocol (NSSK) was also found to be subject to attack, [Denning and Sacco, 1982]. This is an example of a freshness or *replay attack*, i.e. an attack in which the spy re-sends a message an honest agent sent earlier. The aim of this protocol is to securely distribute a new session key to Alice and Bob, K_{AB}. Alice and Bob both have keys shared only with the server, K_A[5] and K_B respectively. These keys are only used occasionally, and then just to set up new session keys, and so are assumed to remain secure. The protocol is described here:

[5] Some authors write this as K_{BS}, as it is the key Bob shares only with the server S.

1. $A \to S : A, B, N_A$

2. $S \to A : \{\!| N_A, B, K_{AB}, \{\!| K_{AB}, A |\!\}_{K_B} |\!\}_{K_A}$

3. $A \to B : \{\!| K_{AB}, A |\!\}_{K_B}$

4. $B \to A : \{\!| N_B |\!\}_{K_{AB}}$

5. $A \to B : \{\!| N_B - 1 |\!\}_{K_{AB}}$

This protocol starts with Alice sending a message to the authentication server indicating her intention to start a conversation with Bob, including a nonce N_A generated by her. In message 2, the server replies with Alice's nonce, Bob's identifier and a fresh session term key to use for talking to Bob, K_{AB}. Also included is a package that Alice can't read, encrypted under Bob's long term key K_B. The whole message to Alice is encrypted under her long term key K_A

Message 3 consists of Alice forwarding to Bob the package she received encrypted with his key, containing a copy of the session term key to be used and an indication that it is to be used in conversation with Alice. The next part of the protocol is called the *handshake*, i.e. a pair of messages exchanged by Alice and Bob under the new key. The purpose of this is to convince Bob that Alice's message 3 was *timely*, i.e. it is not the result of a replay attack. Bob replies to Alice with a nonce he has generated encrypted with the new session key. Alice knows that this must have come from Bob, since only he could have decrypted message 3. Alice replies with $N_B - 1$, encrypted with the same key. Bob knows that only Alice could have decrypted his nonce and subtracted 1, so he accepts the use of K_{AB} for talking to Alice.

However, despite this handshake, Denning and Sacco were able to show that the protocol is still susceptible to a replay attack. Suppose a spy C has obtained a session key, K_{AB}. The protocol must allow for this; if session keys are considered to be indefinitely secure, then we would not need a protocol for setting up new ones. The spy can then fool Bob into using that key for a new conversation span, and masquerade as Alice, by the following sequence of messages:

3. $C_A \to B : \{\!| K_{AB}, A |\!\}_{K_B}$

4. $B \rightarrow C_A : \{\!| N_B |\!\}_{K_{AB}}$

5. $C_A \rightarrow B : \{\!| N_B - 1 |\!\}_{K_{AB}}$

C does not need to know Bob's long term key K_B to create message 3 - he just replays it, as Alice must have sent it as a genuine message 3 earlier. At the end of this sequence, Bob believes a correct protocol run has been followed, and accepts K_{AB}. Denning and Sacco suggested a fix for this problem involving the use of timestamps. This requires the assumption that principals have access to at least loosely synchronized clocks, an assumption that Needham and Schroeder were disinclined to make. The protocol now requires no final handshake, and runs as follows, where T_S denotes a timestamp generated by the server S:

1. $A \rightarrow S : A, B$

2. $S \rightarrow A : \{\!| B, K_{AB}, T_S, \{\!| A, K_{AB}, T_S |\!\}_{K_B} |\!\}_{K_A}$

3. $A \rightarrow B : \{\!| A, K_{AB}, T_S |\!\}_{K_B}$

Needham and Schroeder suggested their own fix involving an extra handshake at the beginning of the protocol, and no need for synchronised network clocks [Needham and Schroeder, 1987]. The issue of synchronised network clocks remains a controversial one: Gong exhibited a new risk of relying on network clocks for security ten years after the Denning and Sacco protocol was proposed [Gong, 1992].

2.3.5 The Otway-Rees Protocol

In [Otway and Rees, 1987], Otway and Rees proposed a shared key protocol designed to eliminate the risk of replay attack. The major departure from the Needham-Schroeder shared key protocol was to require Alice to inform Bob of her wish to communicate in message 1, before any interaction with the authentication server. This way, both principals are involved with obtaining the session key. Three nonces are required: one to assure Alice of Bob's identity, N_A, one to assure Bob of Alice's identity, N_B, and one to identify the run, N. The protocol is:

1. $A \rightarrow B : N, A, B, \{\!| N_A, N, A, B |\!\}_{K_A}$

2. $B \rightarrow S : N, A, B, \{\!| N_A, N, A, B |\!\}_{K_A}, \{\!| N_B, N, A, B |\!\}_{K_B}$

3. $S \rightarrow B : N, \{\!| N_A, K_{AB} |\!\}_{K_A}, \{\!| N_B, K_{AB} |\!\}_{K_B}$

4. $B \rightarrow A : N, \{\!| N_A, K_{AB} |\!\}_{K_A}$

In the first message of this protocol, Alice generates nonce N and nonce N_A to identify a new run, and then signals to Bob her desire to establish a fresh shared key (which will be K_{AB}). During the course of the protocol, the common nonce N gets encrypted by both parties - hence it is sent in the clear in message 1. The identifiers for the principals, A and B, also have to be sent in the clear so that the server knows which keys to use to decrypt the contents of message 2 and obtain N_A and N_B. Nonce N_A is sent encrypted with Alice's long term key. Bob receives the encrypted package $\{\!| N_A, N, A, B |\!\}_{K_A}$, but cannot read it. He forwards it along with a package containing a nonce he generates himself (N_B) encrypted with his own long term key to the server S. The server knows all long term keys, so can decrypt both the packages it receives. It generates a fresh shared key, packages it up separately for A and B, and sends the whole lot to B in message 3. Bob decrypts his part of message 3, and checks that the nonce N_B is as he generated. If it is indeed the same, he accepts session key K_{AB}. B then forwards the rest of the message to A, who checks her nonce before accepting K_{AB}.

In later literature, [Burrows et al., 1990], Burrows et al. suggested that the same guarantees could be derived from a protocol using only two nonces. They also suggested, mistakenly, that nonce N_B need not be encrypted in message 2. Attacks were found on this simplified protocol by Mao and Boyd, [Mao and Boyd, 1993], and Paulson, [Paulson, 1998]. Paulson also showed that a version of the protocol that encrypted N_B but had only two nonces was secure with respect to his assumptions. Here is the simplified protocol as suggested by Burrows et al. – it operates in the same manner as the standard Otway-Rees, except that there is no nonce N, N_A is used instead of N in messages 2, 3 and 4, and nonce N_B is sent in the clear in message 2:

1. $A \rightarrow B : N_A, A, B, \{\!| N_A, A, B |\!\}_{K_A}$

2. $B \rightarrow S : N_A, A, B, \{\!| N_A, A, B |\!\}_{K_A}, N_B, \{\!| N_A, A, B |\!\}_{K_B}$

3. $S \rightarrow B : N_A, \{\!|\, N_A, K_{AB} |\!\}_{K_A}, \{\!|\, N_B, K_{AB} |\!\}_{K_B}$

4. $B \rightarrow A : N_A, \{\!|\, N_A, K_{AB} |\!\}_{K_A}$

Paulson discovered the following attack on the simplified version:

1. $A \rightarrow C_B : N_A, A, B, \{\!|\, N_A, A, B |\!\}_{K_A}$

1.' $C \rightarrow A : N_C, C, A, \{\!|\, N_C, C, A |\!\}_{K_C}$

2.' $A \rightarrow C_S : N_C, C, A, \{\!|\, N_C, C, A |\!\}_{K_C}, N_{A'}, \{\!|\, N_C, C, A |\!\}_{K_A}$

2." $C_A \rightarrow S : N_C, C, A, \{\!|\, N_C, C, A |\!\}_{K_C}, N_A, \{\!|\, N_C, C, A |\!\}_{K_A}$

3." $S \rightarrow C_A : N_C, \{\!|\, N_C, K_{CA} |\!\}_{K_C}, \{\!|\, N_A, K_{CA} |\!\}_{K_A}$

4. $C_B \rightarrow A : N_A, \{\!|\, N_A, K_{CA} |\!\}_{K_A}$

Informally, what's going on this attack is:

1. *A* tries to start a session with *B*, but the message is intercepted by the spy *C*, who notes nonce N_A.

1.' *C* starts a new run with *A*.

2.' *A* believes *C* to be an honest agent, and so sends message 2 of the protocol to the server, having generated another new nonce, $N_{A'}$. *C* again intercepts this message.

2." *C* mixes up the message *A* tried to send the server with the nonce N_A he noted earlier to trick the server.

3." The server is fooled and sends *C* back the old nonce N_A encrypted with *A*'s secret key.

4. *C* can now masquerade successfully as *B*.

In terms of authenticity, this is a serious attack: Alice has no way of telling if she is talking to the real Bob or not. We look at how Paulson discovered this attack in §2.4.4.1 below.

2.3.6 The Neuman-Stubblebine Protocol

The Neuman-Stubblebine protocol was designed to improve on previous shared key protocols by using timestamps as nonces, [Neuman and Stubblebine, 1993]. Here is the key establishment part of the protocol:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow S : B, \{\!|\, A, N_A, T_B |\!\}_{K_B}, N_B$
3. $S \rightarrow A : \{\!|\, B, N_A, K_{AB}, T_B |\!\}_{K_A}, \{\!|\, A, K_{AB}, T_B |\!\}_{K_B}, N_B$
4. $A \rightarrow B : \{\!|\, A, K_{AB}, T_B |\!\}_{K_B}, \{\!|\, N_B |\!\}_{K_{AB}}$

In message 1, A generates a nonce N_A and sends it to B to signal that she wants to establish a new key. B then generates both a timestamp, T_B, and a nonce N_B. The timestamp he sends to the server encrypted under his long term key, while the nonce is sent in the clear. In message 3 the server sends A a package containing B's identifier, A's nonce, the fresh session key K_{AB} and the timestamp for the key. A knows the key must be good, because only the server could encrypt her nonce N_A under her long term key, and by the presence of B's identifier, the server has also indicated the key is for use with B, preventing an impersonation attack. In message 4, A forwards to B the rest of message 3, but this time encrypting nonce N_B under the fresh session key K_{AB}. B now receives the package containing A's identifier, the new key K_{AB} and the timestamp he generated. He uses K_{AB} to obtain nonce N_B from the final part of message 4, which proves to B the identity of A, since only A could have got the session key from the server and used it to encrypt N_B.

The attack on this protocol is an example of a *type attack*, and is dependent on the implementation of the protocol using the same length bit strings for nonces as it uses for keys. Here is the attack, due to Hwang et al. [Hwang et al., 1995]:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow S : B, \{\!|\, A, N_A, T_B |\!\}_{K_B}, N_B$
4. $C_A \rightarrow B : \{\!|\, A, N_A, T_B |\!\}_{K_B}, \{\!|\, N_B |\!\}_{N_A}$

In this attack, the spy first notes nonce N_A, sent by A in the clear in message 1. The spy intercepts B's message 2 to the server, and then uses the encrypted package at the beginning of that message and the nonce N_B sent in the clear at the end to construct a fake message 4 to send to B. This fools B into accepting nonce N_A as the session key, because the encrypted package he is expecting has the same structure as that sent in message 2, and the spy can use nonce N_A to encrypt nonce N_B.

Some research has focused on eliminating the risk of type attacks by distinguishing different types of object using tagging techniques, [Heather et al., 2000]. In this work, the authors prove that by using a simple tagging system (which is assumed to be known to the spy) whereby each agent attaches a few bits to each message element indicating its intended interpretation, type attacks can be eliminated. What they in fact show is that if there is a type attack which cannot be prevented by this tagging, then there is also a simple attack where the spy makes up a value of his own. Informally, the intuitive idea here is that type attacks usually involve a spy forwarding an encrypted packet, hoping that it will be misinterpreted by an honest agent. However, he cannot actually get inside the encrypted packet to change the tag, so the tagging prevents the attack. If the type attack involves forwarding an unencrypted value for misinterpretation, then the spy can just as effectively forward a value he has made up. We can conclude from this work that a good implementation can remove the risk of type attacks. Some of the formal approaches below, such as Paulson's §2.4.4.1, assume such an implementation will always be used, and so adopt a strongly-typed formalism in which type confusion cannot occur.

2.4 Formal Methods for Protocol Analysis

The attacks presented above give an indication of just how tricky it is to specify a secure protocol. One of the main thrusts of security research has been to apply formal methods to the problem. Researchers have applied techniques from term rewriting, model checking, theorem proving and modal logics amongst others. We look at some of this work below.

The essential problem faced by designers of methods for protocol analysis is that

the problem is, in general, undecidable, even under quite severe restrictions on protocol design, [Durgin et al., 1999]. There are at least three unbounded factors in the general case: an unbounded number of agents who may be involved, an unbounded number of parallel protocol runs (and so an unbounded number of nonces), and an unbounded set of terms known to the intruder.

2.4.1 State Exploration Approaches

One way to analyse security protocols is to model them as some kind of state-transition system. This idea has been used by a number of researchers.

2.4.1.1 The Dolev and Yao Approach

An early attempt to formalise the security protocol problem was presented by Dolev and Yao, [Dolev and Yao, 1983]. The intruder modelled by their system became the standard model in terms of its ability to intercept all messages and send fake messages based on what has appeared in the traffic. However, their spy did not start with any secret information, such as a private key belonging to one of the agents. The system is modelled as a machine used by the intruder to generate words. The rules of the system are expressed as rewrite rules. The problem of the intruder trying to discover a secret is then expressible as a word problem in a rewrite system. Dolev and Yao presented several algorithms that could be used to decide the problem for certain classes of protocols. However, the model had several weaknesses. It could not model the storing of information by agents, and could not be used to prove authenticity or freshness.

Other researchers have developed tools based on Dolev and Yao's work with augmentations aimed at addressing these shortcomings. These include Millen's Interrogator, [Millen et al., 1987], and Meadows' NPA[6], [Meadows, 1996b]. NPA has been used to analyse many protocols, including group protocols, [Meadows, 2000a], with a certain amount of success. It can also be set to present counterexamples in the case of a ¤awed protocol, yielding an appropriate attack. Proving authenticity guarantees with NPA can be messy though, [Meadows, 2000b].

[6]NPA stands for 'NRL Protocol Analyser'. NRL in turn stands for 'Naval Research Laboratories'. Meadows' tool is sometimes also referred to as 'NRL'.

2.4.1.2 Model checking

Model checking is a state exploration based technique that has been applied particularly successfully to hardware verification and communication problems. The idea of model checking is to first construct a model of the system you are interested in as a finite state machine. A model checker then exhaustively searches the reachable states in the model to determine whether or not they conform to some property. Properties are typically specified in a modal logic of some kind, which means that typically both secrecy and authenticity properties can be checked for.

Lowe used the FDR model checker to find an attack on the Needham-Schroeder public key protocol, presented in §2.2, [Lowe, 1996]. One of the key advantages of model checking is its ability to present a counterexample when asked to verify a faulty system. When analysing the public key Needham-Schroeder protocol, the model checker informed Lowe that the system was able to perform an action committing Bob to a session with Alice even though Alice is not trying to establish a session with Bob. By examining the CSP trace that formed this counterexample, the attack can be found. Lowe's team have continued to use the FDR model checker to analyse protocols, successfully applying the technique to at least 50 protocols, [Donovan et al., 1999].

A disadvantage of model checking for security protocol analysis is that, in order to guarantee termination in a reasonable time (and without exhausting available memory), a small finite abstraction of the system has to be considered, e.g. with only two agents, each agent only able to generate one nonce, only one run of the protocol etc. This could lead to attacks going undiscovered, and faulty protocols being certified as correct. There are two ways to attack this problem: one, adopted by Lowe, is to first check the small finite instance of the protocol, and then to prove on paper that if an attack exists on the larger system, then there must also be an attack on the smaller system, [Lowe, 1999]. Another is to do away with finite instances, and just look for attacks on the unconstrained infinite system with arbitrary numbers of agents, nonces etc. Termination (without a guarantee of correctness) can be achieved by checking all traces of up to 10 messages say, or by using a time limit. This is the approach adopted by Basin [Basin, 1999], who also adds some heuristics to prune and re-order the search space, resulting in attacks being found very quickly. We compare Basin's approach to

ours in more detail in Chapter 11.

2.4.2 The BAN logic

Burrows, Abadi and Needham were the first to propose a formal modal logic analysis of the security protocol problem [Burrows et al., 1990]. They demonstrated the use of a logic based on beliefs, where formulae express what an agent may infer upon receiving a given message. This logic has become known as the BAN logic. It is simple to understand and can be used to prove a variety of properties of protocols. However, in general, it does not result in a counterexample in the case of a ¤awed protocol, though it may suggest how one might be found. BAN logic has been used to detect ¤aws in several protocols, but it has also missed others. Nessett exhibited a protocol that was deeply ¤awed, yet the BAN logic showed it to be correct, [Nessett, 1990]. Burrows et al. responded that this was because the protocol violated one of their assumptions - that no messages should give away secret keys. However, as Nessett pointed out, this assumption should be verified. It may be possible to trick an agent into giving away a key. There are also some quite subtle issues involved in the translation from formal specification to the BAN logic - a common problem in all formal methods work. A good discussion of these issues is given in [Meadows, 1996a].

Various researchers have proposed augmented versions of the BAN logic, designed to verify secrecy properties such as that referred to by Nessett, [Kindred, 1999, Kessler and Wedel, 1994]. However, one of the chief attractions of the original logic was its simplicity. Others have produced automated implementations of BAN-like logics, e.g. [Schumann, 1997], where Schumann used the theorem prover SETHEO, [Letz and Stenz, 2001], to automate proof in the calculus. However, this approach suffered from the same drawbacks as BAN logic described above, and though proofs were often found very quickly, SETHEO's failure to find a proof provided no information on where the ¤aw might be (it would just run forever). One advantage of doing the proofs by hand was that the breakdown of a proof attempt would often lead to the discovery of an attack.

2.4.3 The Spi Calculus

Abadi proposed an improved calculus for reasoning about security protocols called the spi calculus in joint work with Gordon, [Abadi and Gordon, 1997]. This is an augmentation of Milner's pi calculus, [Milner, 1993]. Abadi and Gordon define a notion of equivalence called *observational equivalence*, i.e. that two processes are the same as far as another process (i.e. an attacker) can observe. This allows them to reason about security properties in terms of this equivalence, e.g. item X remains secret if a protocol with X is observationally equivalent to a protocol with X' for any X'. The spy is modelled rather neatly as an arbitrary spi calculus process. The spi calculus has been used to specify and prove both authenticity and security properties of a number of protocols. Gordon and Jeffrey have also produced a tool called Cryptyc that type checks protocols specified in spi calculus, [Gordon and Jeffrey, 2001]. Typing tags are added to the protocol, which means the type checker can show correctness in terms of security and authenticity in the presence of a Dolev-Yao intruder. Properties can be checked very quickly, though the tool is quite rigid in terms of what properties can be checked for, what elements are allowed in protocols and what kinds of encryption can be used. There is also the slight chance of a perfectly good protocol failing to type check, though if it does type check it is at least guaranteed to be secure. No counterexamples are given in the case of a ¤awed protocol.

2.4.4 Theorem Proving

Theorem proving of some kind has been used by a number of researchers to tackle the protocol analysis problem. A variety of different models have been used.

2.4.4.1 Paulson's Inductive Method

Paulson has used an inductive approach to verify properties of protocols, [Paulson, 1998]. Protocols are formalised in typed higher-order logic as the set of all possible traces, a trace being a list of events like 'A sends message X to B'. Intruder knowledge is specified in terms of what has been seen in the trace so far, using the *synth* and *analz* operators. For a trace T, $analz(T)$ specifies every individual term a

spy may extract from the trace, and $synth(analz(T))$ specifies everything he can build from those terms. This formalism is mechanised in the Isabelle/HOL theorem prover. Properties of the security protocol can be proved by induction on traces. Paulson has proved theorems both of authenticity and secrecy by specifying them in terms of properties of traces, e.g. 'If *A* receives message 3 with nonce *N*, and he sent message 1 with nonce *N* to *B*, then message 3 came from *B*.'

Sometimes a theorem cannot be proved, and in these situations an attack may be suggested, as in the Otway-Rees variant shown above (§2.3.5). However, Paulson notes that it can be hard to interpret a failed proof attempt. It may be that some additional lemmas need to be proved or a generalisation made. So, while Paulson's formalism is expressive enough to capture a good model of the system with arbitrary numbers of agents and nonces, agents with compromised keys etc., it does not provide much assistance in finding attacks on faulty protocols.

Paulson's formalism has been used to model the Needham-Schroeder protocols, the Otway-Rees protocol, smart card protocols and the SET online payment protocol amongst others [Paulson, 1998, Bella et al., 2002, Bella, 2003].

2.4.4.2 First-Order Formalisms

Weidenbach has developed a first-order formalism of the security protocol problem, [Weidenbach, 1999]. His aim was to combine the benefits of Paulson's inductive method and finite state analysis methods. By formalising the problem in a fragment of monadic Horn logic, Weidenbach is able to specify an inductive model where the intruder's knowledge is, in general, unbounded, but still to carry out proofs automatically using the SPASS saturation-based first-order theorem prover, [Weidenbach et al., 1999]. A ¤avour of the formalism can be gained from the following example of a translation from our protocol notation (introduced in §2.2) to Weidenbach's formalism. Weidenbach used the Neuman-Stubblebine protocol for his case study (§2.3.6). Weidenbach formalises the first message in this protocol with these three formulae:

(1) $Ak(key(as,t))$

(2) $P(a)$

(3) $M(sent(a,b,pair(a,na))) \wedge Sa(pair(b,na))$

Formula (1) expresses that Alice (a) shares key as with the server t (Weidenbach uses t to stand for 'trusted server'). Formula 2 tells us that a is one of the principals involved in the protocol. Formula (3) tells us firstly that a sent and stored message 1 of the protocol, and secondly that she has stored the nonce she generated for verification later on. Weidenbach formalises the rest of the protocol in a similar manner. He also formalises a spy able to synthesize messages based on previous traffic, and is then able to find the type attack on the protocol first discovered by Hwang et al. (§2.3.6) by proving a conjecture that there exists some x that B thinks is a secure key for A, but is in fact known to the spy, allowing him to impersonate A.

The main advantages of Weidenbach's work are that proofs are fully automatic and attacks are quite easy to construct from the output provided by SPASS. It is a matter of inspecting the proof and working backwards through it to see what messages have been exchanged. This process could probably be automated. However, Weidenbach's formalism is not as expressive as Paulson's, and the proofs presented are not quite as strong, for the following reasons:

1. The abilities of the spy are weaker. He does not control the private keys of any legitimate agents, and cannot generate fresh nonces of his own. This severely limits the scope for impersonation attacks that break authenticity properties.

2. There are only two agents in Weidenbach's model. This means that to show that the whole system is secure, we would have to prove theorems such as are proved in model checking work, showing that if there is an attack on the large system (i.e. with any number of agents and simultaneous protocol runs) there must be an attack on the small system (with just two agents and one protocol run) as well.

3. Each agent modelled only knows one half of the protocol. An attack may require one agent to be performing both halves of the protocol at once to two different agents, e.g. Paulson's attack on Otway-Rees [Paulson, 1998], Clark's parallel session attack [Clark and Jacob, 1996].

We compare our work to Weidenbach's in detail in §11.

Blanchet has developed another Horn-clause model, in which the spy has full Dolev-Yao capabilities, [Blanchet, 2002]. A specification in a spi calculus variant is translated automatically into Horn clauses, and then a customised prover is used to determine whether a certain term is derivable. If it is derivable, then an attack can be found by stepping through the proof to see what messages were exchanged, in a similar way to Weidenbach's work. The method can verify protocols assuming an unbounded number of nonces and parallel runs, though not an unbounded number of agents. It is also not complete, in the sense that it may not terminate, or it may terminate with a false attack. These false attacks are due to the way Blanchet represents freshness to avoid having to use an explicit trace-based inductive formalism. False attacks seem not to occur in practice though, and often secrecy and authenticity properties can be verified in under a second for typical protocols.

Other first-order formalisms use rewriting as the basis for searching for attacks. For example Jacquemard et al. have proposed a rewrite rule formalism suitable for the theorem prover `daTac`, [Jacquemard et al., 2000]. A key feature of the `daTac` is that it enables associative-commutative (AC) unification over the arguments of AC operators. This is used in the modelling of sets of messages that have been sent and sets of terms the intruder has learnt. They also proposed a high-level specification language for protocols, which can then be automatically compiled to the rewrite formalism. The system produced is called Casrul. It has been tested on a number of flawed protocols from the Clark-Jacob corpus, [Clark and Jacob, 1997], with good results, [Chevalier and Vigneron, 2002], though it does require the user to decide a *scenario* in advance, i.e. what agents will be involved and what role they will play in the protocol. We compare Casrul to CORAL in more detail in §11.3.

Ernie Cohen has produced a first-order formalism for verification based on invariants, called TAPS [Cohen, 2000]. TAPS models protocols as a transition system. A state is represented by the set of steps that have been executed and the set of messages that have been sent. The TAPS system constructs first-order invariants for the system that capture desirable security properties. Sometimes, particularly in the case of simple protocols, these protocols are constructed automatically, but sometimes hints from the

user are required. Once the invariant has been found, it is sent to a first-order theorem prover and typically solved extremely fast (when the protocol is secure). If the protocol has a ¤aw however, TAPS does not give a counterexample. TAPS has been used to verify a large number of protocols, and Cohen has published plans to add bitwise operations and modelling of algebraic properties of cryptographic functions to the system, [Cohen, 2003]. However, these plans look unlikely to become reality, as Cohen has moved into a different area.

2.4.5 The Strand Space Model

A lot of recent interest has been sparked by Fábrega, Herzog and Guttman's *strand space* model for analysing protocols, [Fábrega et al., 1999]. A strand space is a collection of *strands*, with a graph structure which expresses causal relations. A strand is a sequence of *events* that a single party may engage in. An event is the sending or reception of a message, represented by a *node* in the graph. Nodes have a sign: a positive sign indicates a message was sent, and a negative sign indicates it has been received.

There are two kinds of strands in the model, those for honest participants and intruder strands. A strand belonging to an honest agent contains that agent's actions in one particular run of the protocol. If an agent is involved in several runs, each of these will have its own strand. Nodes in separate strands are adjacent when they represent the sending and reception of the same message. As an example, the Needham-Schroeder public key protocol as represented by a strand space model is presented in Figure 2.2.

The intruder strands are sequences of nodes representing the intruder doing anything that he is able to do, e.g. intercepting a message, breaking it down into component parts, building new messages and sending them. Useful intruder actions may be modelled by connecting many strands. For example, in Figure 2.3 we show Lowe's attack on the NSPK protocol (§2.3.3) represented in a strand space model. There are two short intruder strands, I_1 and I_2.

A *bundle* is a finite acyclic subgraph of the strand space that is in a certain sense backwards-closed: all received messages occurring in strands in the bundle must have come from nodes also in the bundle, and if an event on a strand is in the bundle, then all preceding events on that strand must also be in the bundle. Secrecy and authenticity

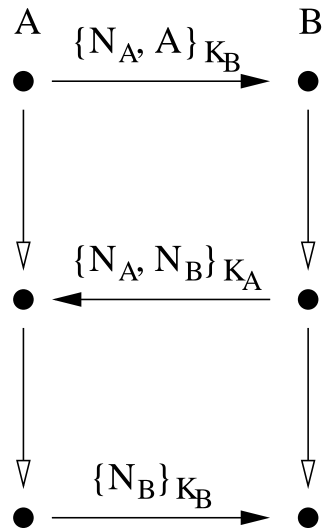

Figure 2.2: NSPK modelled in a strand space

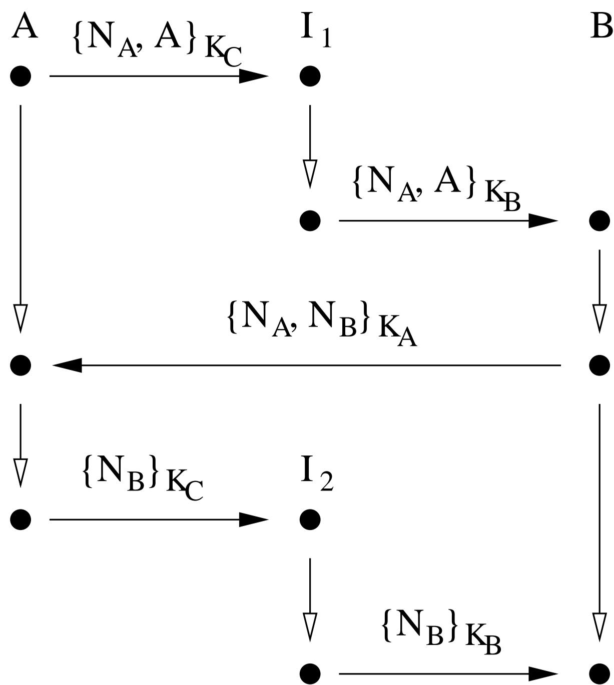

Figure 2.3: NSPK attacked

properties of the protocol can be expressed in terms of the connections between different kinds of strands and bundles. For example, the property of the NSPK protocol that is violated in Figure 2.3 would be roughly stated as:

Proposition 1 *If:*
1. $\mathcal{C}$ is a bundle in an NSPK strand space
2. $\mathcal{C}$ contains a responder strand for B using nonces N_A and N_B, in apparent communication with A
3. The spy does not have A's private key, and
4. N_B is a fresh nonce
Then $\mathcal{C}$ also contains an initiator strand for A using nonce N_A and N_B

In the original work, [Fábrega et al., 1999], proofs of these properties were carried out by hand. For example, the authors showed that the property in Proposition 1 is true for Lowe's revised version of the NSPK protocol. More recently, an automated tool called Athena which uses the strand space model has appeared, [Song et al., 2001]. Athena can also present counterexamples when a protocol is found faulty.

The advantages of the strand space model seem to be that, once understood, it provides a simple and succinct way of expressing the causal relationship between different parts of a protocol, and together with some techniques for cutting down the search space, it can be used to efficiently obtain automated proofs. However, it is not as ¤exible as, for example, Paulson's model. Athena, the automated strand space prover, can verify many protocols in a fraction of a second, though it does require a limit on number of concurrent runs and length of messages to achieve termination sometimes. We compare the strand space model work with our own in detail in Chapter 11.

2.5 Outlook

The need for secure cryptographic protocols increases daily. The higher uptake of the Internet and electronic commerce makes fraud potentially more profitable, increasing the risk of attack and hence the need for security. Proposed new applications often require the development of a new kind of protocol, e.g. Internet auction houses

[Stajano and Anderson, 1999] and mobile phones incorporating Internet access. However, protocol designers continue to make simple design mistakes. For example, the "secure" mail protocol proposed by GCHQ has been criticised for ignoring many known protocol ¤aws, [Anderson and Roe, 1997].

As formal protocol analysis methods develop, researchers are looking to take it in new directions. For example, some work has involved developing more fine-grained models of the encryption algorithms used, taking into account properties such as associativity and commutativity, [Millen and Shmatikov, 2003]. Others have looked at verifying properties of protocols which assume some underlying level of secrecy has already been established, [Bella et al., 2003].

While the field of formal protocol analysis is a very active one, with a huge variety of techniques and tools now available, formal methods are not widely used by protocol designers. In fact, almost all published formal protocol analysis has been carried out by the designers of the particular techniques used. This is probably because often the formalisation of the protocol into the syntax required by the theorem prover or model checker, and the subsequent use of the tool, demands a great deal of specialist knowledge. Formal methods will only be used more widely when these tasks can be made easier. Formal protocol analysis has other limitations. For example, it relies on the abilities of an attacker being captured by a small number of assumptions. It may be possible for an intruder to perform some action outside of this model and thus effect an attack. So, a formal guarantee of protocol security is only as strong as the spy model used.

However, formal methods have been responsible for the uncovering of many published protocol attacks. It is possible that without these techniques, these attacks would not have been discovered until a malicious user made use of them to commit fraud. The model checking techniques seem to be particularly good at detecting simple errors in protocols, while theorem proving methods can provide much stronger guarantees of security than informal reasoning. So, the field of formal methods for protocol analysis remains a promising one.

A large amount of security protocol literature is not concerned with formal methods, but rather with good engineering practices, [Abadi and Needham, 1996,

Gong and Syverson, 1998]. Also, authors such as Anderson have pointed out that many security compromises arise as a result of implementation level errors [Anderson and Needham, 1995]. Good system security can only arise as a consequence of well designed and verified protocols, prudent engineering and sound implementation.

In summary, the field of cryptographic security protocol design and analysis is one of increasing importance. Formal techniques for protocol analysis form a key part of research in the field, but cannot guarantee security on their own. Formal analysis methods must be made simpler to use and understand in order for them to gain wider acceptance by those responsible for implementing secure computer systems.

Chapter 3

Refuting Incorrect Conjectures

We saw in the previous chapter that Paulson's model for the cryptographic protocol analysis problem provides a very flexible and expressive setting for proving security properties. However, some considerable expertise is required to complete the proofs. If a protocol is flawed, a user may waste a lot of time trying to prove a conjecture about the protocol's security which is in fact false. An automated tool that can not only detect these incorrect conjectures but also present a counterexample, i.e. an attack on the protocol, would clearly be very useful. In this chapter we survey various different approaches to the refutation of incorrect conjectures, and evaluate how suitable they would be for discovering protocol attacks in a Paulson-style inductive model.

3.1 First-Order Finite Domain Enumerators

Several tools have been developed to find counterexamples to first order formulae over finite domains. These include Finder [Slaney, 1995], and MACE [McCune, 1994]. These programs search a finite domain, evaluating a given conjecture at each point to see if a counterexample has been found. Finder is a generic counterexample finder that can be used with any search algorithm, whereas MACE has been designed specifically to use the Davis Putnam procedure. As automated first-order theorem provers become more powerful and can be applied to more complex problems, it is likely that the development of tools like MACE and Finder will continue in a complementary fashion.

However, they are not suitable for applying to security protocols, since security protocols in general have infinite models.

3.2 Finding Counterexamples by Instantiating with Constructors

Inductive models are often specified in terms of a *constructor* theory – indeed, Paulson's model for the cryptographic protocol problem is such a theory. A set of constructors for a datatype is a set of symbols such that any term in the datatype can be built from those symbols. A set of constructors is *free* when every element in the datatype has a unique representation in terms of these constructors, and any two non-identical constructor terms are unequal in the theory. At least two authors have suggested ways of finding counterexamples in these kinds of theories. We describe their approaches below, and evaluate their suitability for discovering protocol attacks in §3.2.3.

3.2.1 Protzen's Calculus for Refutation

In [Protzen, 1992], Protzen presents a calculus which is sound and complete for the refutation of faulty universally quantified conjectures in inductive theories specified by universally quantified axioms and using only free datatypes. Given a universally quantified and (possibly) incorrect conjecture, the method of refutation is as follows: first, formulate an existentially quantified negation of the conjecture. Then, attempt to find an instantiation of the existentially quantified variables such that the counterexample statement is satisfied. So it is a search to satisfy a Σ_1 conjecture in a Π_1 theory. The search for the counterexample is guided by the recursive definitions of the function symbols in the counterexample formula. For example, suppose we are trying to refute the false conjecture:

$$\forall n, m : nat . plus(n+1, m) \neq 1 \tag{3.1}$$

with definitions

$$\begin{aligned}
&\forall x, y : nat.\ x = 0 \Rightarrow plus(x, y) = y \\
&\forall x, y : nat.\ x \neq 0 \Rightarrow plus(x, y) = plus(x-1, y) + 1
\end{aligned}$$

A counterexample must be a witness to the formula:

$$\exists n, m : nat.\ plus(n+1, m) = 1$$

The method proceeds by propagating a restriction on the value of $plus(n+1, m)$ to its subterms. The definition above of *plus* has two cases, and the first one gives us the formula

$$\exists n, m : nat.\ n+1 = 0 \bigwedge m = 1$$

which is unsatisfiable. The second case of the definition, after some simplification and propagation of restrictions, gives us the formula:

$$\exists n, m : nat.\ n = 0 \bigwedge m = 0$$

This gives us values for the variables n and m which satisfy the original (negated) formula.

The method will not always terminate, as a semi-decision procedure is as good as can be obtained for arithmetic formulae. So, for the method to be useful in practice, a depth limit is specified. Protzen has implemented the counterexample finder as part of the INKA inductive theorem prover, [Hutter and Sengler, 1996]. Its chief application in this context is to refute incorrect generalisations. The method is quite suitable for this, since the over generalisations that occur in this context are often quite trivially false, and a counterexample can be found very close to the base case values.

3.2.2 Reif's Counterexample Finder

Reif et al., [Reif et al., 2001], have implemented a method for counterexample construction for infinite datatypes in first-order constructor specifications. It is integrated with the interactive theorem prover KIV, [Reif, 1995]. Their method incrementally instantiates the variables in a formula with constructor terms and evaluates the formulae produced using the simplifier rules made available to the system during proof attempts. A heuristic strategy guides the search through the resulting subgoals for one that can be reduced to *false*. If such a subgoal is not found, the search terminates when all variables have been instantiated to constructor terms. In this case the user is left with

a *model condition*, which must be used to decide whether the instantiation found is a valid counterexample.

For example, take the conjecture:

$$sorted(l_1) \wedge sorted(l_2) \quad \rightarrow \quad sorted(append(l_1, l_2))$$

with the usual definitions of list append and the predicate *sorted*. This is not true in all models of the specification, and Reif's system finds the counterexample

$$l_1 = cons(e_1, nil) \qquad l_2 = cons(e_2, nil)$$

with the model condition

$$\neg(e_1 \leq e_2)$$

The user sees that this model condition can be satisfied, e.g. with $e_1 = 1$ and $e_2 = 0$, and so accepts the counterexample.

3.2.3 Evaluation of Protzen's and Reif's Approaches

Reif's system is attractive in that it is tightly integrated into the theorem prover, allowing it to be easily called during a proof attempt. It can also deal with non-free datatypes, which Protzen's method cannot. However, although the examples tackled by Reif are larger than those in [Protzen, 1992], they all involve domains where the datatypes can be easily enumerated. For example, Reif refutes a number of false graph theory conjectures by using a non-free datatype for graphs, in which a vertex may be explicitly added to the graph, or may be be implicitly added in an edge. This means that any combination of vertices and edges constitutes a valid graph, making it easy to construct candidate counterexamples.

Reif's method does not seem be suitable for inductive datatypes, such as are used in protocol analysis. The relatively simple technique of combining constructors would tend to generate many non-valid traces, for example, traces in which an honest agent sends message 3 in a protocol without having received message 2. This could be accounted for in the specification by using a predicate to specify valid traces, but it would seem too inefficient to keep generating invalid traces only to later reject them.

The vast majority of traces generated by constructor symbols would indeed be invalid. Protzen's method generates its instantiations by a slightly different method, but has the same drawback.

3.3 Monroy's Non-theorem Work

In 1993, Raul Monroy produced an MSc thesis, [Monroy, 1993], on correcting incorrect inductive conjectures in first-order logic. The non-theorem detection was carried out by a simple 'formula tester', borrowed from the *Clam* proof planner, [Bundy et al., 1990]. This system tests conjectures by evaluating their truth at a few pre-chosen values, typically 0, $s(0)$ and $s(s(0))$ for conjectures over the natural numbers. Monroy used information based on the nature of the failure of the proof as a basis for choosing a patch attempt. The idea is to synthesize a *corrective predicate* step by step as the proof is attempted. At the end of the process, the corrective predicate gives conditions under which the conjecture is a theorem.

Monroy's latest work, [Monroy, 2000], proposes a more powerful technique for correcting faulty conjectures. This new mechanism performs non-theorem detection by referring to a user-defined finite set of obvious contradictions, e.g. $x \neq x, s(x) = x, s(x) = p(x)$. If, when proving a conjecture, the proof of a particular subgoal fails, and that subgoal is **not** one of the known contradictions, then the subgoal is abducted into the patch predicate as a condition for the original conjecture to be true.

For example, given the non-theorem

$$\forall N : nat.double(half(N)) = N \tag{3.2}$$

with definitions

$$\begin{aligned} double(0) &= 0 & half(0) &= 0 \\ double(s(N)) &= s(s(double(N)) & half(s(0)) &= 0 \\ & & half(s(s(N)) &= s(half(N)) \end{aligned}$$

Monroy's mechanism synthesizes a corrective predicate with definition

$$
\begin{aligned}
c(0) &= true \\
c(s(0)) &= false \\
c(s(s(N)) &= c(N)
\end{aligned}
$$

which specifies that the conjecture (3.2) is true for even numbers.

Note that if a subgoal is a non-theorem that is not included in the user-defined contradiction set, then the process does not identify it as being incorrect. So, the output from the process is of the form, 'I can't prove conjecture X as you presented it to me, it may or may not be incorrect, but given conditions Y I can prove it to be true.' However, the same 'folk knowledge' that leads us to look for simple counterexamples around the base cases of recursive definitions suggests that a small but well chosen set of contradictions may well trap a large proportion of the non-theorems, at least in simple applications.

Monroy is currently working on extending the idea to more complex problems and to higher-order theories, [Monroy, 2003]. He also observes that the corrective predicates synthesized are sometimes 'unnatural', and is working on trying to characterise families of corrective predicates to improve the quality of the patches.

An attraction of Monroy's work is that we would like be able to say how to fix a protocol that is found to be faulty, and Monroy's corrective predicates may provide a means of doing this. They might also allow us to specify requirements for the secure implementation of an otherwise insecure protocol (this would be useful in the context of legacy protocols which cannot be changed for backwards compatibility reasons). However, the system for actually detecting incorrect conjectures is currently much too basic for our requirements.

3.4 Proof by Consistency

Proof by consistency is a technique for automating inductive proof. It has also been called *inductionless induction*, *inductive completion*, and *implicit induction*, as the actual induction rule used is described implicitly inside a proof of the con-

jecture's consistency with the set of hypotheses. It was first proposed by Musser, [Musser, 1980], and developed by Huet and Hullot, [Huet and Hullot, 1982], Jouannaud and Kounalis, [Jouannaud and Kounalis, 1989], Bachmair, [Bachmair, 1991], Ganzinger and Stuber, [Ganzinger and Stuber, 1992], and Bouhoula and Rusinowitch, [Bouhoula and Rusinowitch, 1995], amongst others. Recent versions, from Bachmair's work onwards, have been shown to be *refutation complete*, i.e. are guaranteed to detect non-theorems in finite time.[1] This is in contrast to classical 'explicit induction' techniques, [Boyer and Moore, 1979], that generally offer no such support.

Proof by consistency has generally fallen out of favour as a method for automating inductive proof, although it has recently been used to verify some properties of the JavaCard bytecode verifier, [Barthe and Stratulat, 2003], and new hybrid versions incorporating some techniques from more conventional explicit induction are still being proposed, e.g. [Deplagne and Kirchner, 2001, Avenhaus et al., 2003]. The loss of interest in the technique was mainly due to problems encountered when scaling up the method to larger problems, and because it is not very similar to the way humans do induction, which makes combining it with an interactive approach quite difficult. However, the suitability of the technique for refuting non-theorems of significant size, e.g. a conjecture about properties of a security protocol, had not previously been tested.

Recently, Comon and Nieuwenhuis have proposed a setting of the proof by consistency technique that encompasses and extends previous versions, [Comon and Nieuwenhuis, 2000]. It allows proof by consistency to be carried out using any saturation-based first-order theorem prover. There is no need for any specialised completion procedure. This allows powerful simplification and redundancy rules developed for automatic first-order theorem proving to be utilised. Much of the work in this thesis is based on this setting, and it will be explained in detail in the next chapter.

[1] Such a technique must necessarily be incomplete with respect to proving theorems correct, by Gödel's incompleteness theorem.

3.5 Evaluation

In order to detect and refute incorrect security conjectures, and so obtain protocol attacks, we require a refutation technique suitable for infinite, inductive models. Various techniques have been proposed to refute small non-theorems that arise in automated reasoning systems, as we have seen (§3.2.1, §3.2.2, §3.3). However, little attention has been paid to refutation of larger non-theorems where enumerating the domain is a non-trivial task. We decided the proof by consistency technique was the most promising for this purpose, since it allows us to use the full power of a modern first-order theorem prover, and it will be interesting to see how useful this aspect of the proof by consistency technique is in a practical example. Also, it gives us at least a theoretical possibility of working towards a proof of the correctness of a secure protocol at the same time as we search for an attack. We describe the theory of the proof by consistency technique in more detail in the next chapter.

Chapter 4

The Theory of the Comon-Nieuwenhuis Method

In this chapter we outline the theory that lies behind the Comon-Nieuwenhuis method for proof by consistency. The chapter is organised as follows: first we give the background required in terms of notation, definitions etc. Then we give a description of the theory and terminology of automatic first-order theorem proving. This is followed by an overview of how the Comon-Nieuwenhuis method works. We then describe the method for Horn clause problems in detail, and describe different methods for obtaining *I-Axiomatisations* (defined in Definition 4 below). Sections 4.3 to 4.6 draw heavily on [Comon and Nieuwenhuis, 1998] and [Comon and Nieuwenhuis, 2000]. In particular, all proofs are from [Comon and Nieuwenhuis, 2000], except for Corollary 1, which we prove here, and Theorem 3, which is proved in [Bachmair and Ganzinger, 1994].

4.1 Background

We use the standard notations of [Dershowitz and Jouannaud, 1990]. $\mathcal{T}(\mathcal{F},\mathcal{X})$ is the set of terms over a set of symbols $\mathcal{F}$ and a denumerable set of variables $\mathcal{X}$. $\mathcal{T}(\mathcal{F})$ is the set of ground terms over $\mathcal{F}$. The subterm of t at *position* p is denoted $t|_p$, and the result of replacing $t|_p$ with s in t is written as $t[s]_p$.

A *multiset* over a set S is a function $M : S \to \mathbb{N}$. The union of multisets is defined

as $M_1 \cup M_2(x) = M_1(x) + M_2(x)$, sometimes written as M_1, M_2.

If $\rightarrow$ is a binary relation, then $\leftarrow$ is its inverse, $\leftrightarrow$ is its symmetric closure, $\rightarrow^+$ is its transitive closure and $\rightarrow^*$ is its re¤exive-transitive closure. We write $s \rightarrow^{\prime} t$ if $s \rightarrow^* t$ and there is no t' such that $t \rightarrow t'$. Then t is called *irreducible* and a normal form of s w.r.t. $\rightarrow$. The relation $\rightarrow$ is *well-founded* or *terminating* if there exists no infinite sequence $s_1 \rightarrow s_2 \rightarrow \ldots$. We define the join of two relations, $\rightarrow_1 \circ \rightarrow_2$, to be the following binary relation: $s \rightarrow_1 \circ \rightarrow_2 t$ iff $\exists u. s \rightarrow_1 u \rightarrow_2 t$. A relation $\rightarrow$ is *con¤uent* if the relation $\leftarrow^* \circ \rightarrow^*$ is contained in $\rightarrow^* \circ \leftarrow^*$. A relation $\rightarrow$ on terms is *monotonic* is $s \rightarrow t$ implies $u[s]_p \rightarrow u[t]_p$ for all terms s, t and u and positions p. A *congruence* is a re¤exive, symmetric, transitive and monotonic relation on terms.

An equation is a multiset $\{s, t\}$ of terms, denoted $s = t$ or equivalently $t = s$. A *£rst-order clause* is a pair of finite multisets of equations Γ (the antecedent) and Δ (the succedent), written as $\Gamma \longrightarrow \Delta$. It is a *Horn clause* if Δ contains at most one equation, and a *de£nite Horn clause* if Δ contains exactly one equation. The empty clause $\Box$ is one with both Γ and Δ empty. We will adopt the notational convention of sometimes writing the multisets $\Gamma \longrightarrow \Delta$ as disjunctions, i.e. we will write $\neg P_1 \vee \ldots \vee \neg P_n \vee Q_1 \vee \ldots \vee Q_m$ to denote $\{P_1, \ldots, P_n\} \longrightarrow \{Q_1, \ldots, Q_n\}$.

A *substitution* σ is a function $\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$. $s\sigma$ is the term obtained by applying substitution σ to s.

A *rewrite rule* is an ordered pair of terms (s, t), written $s \rightarrow t$, and a set of rewrite rules R constitutes a *term rewrite system* (TRS). The rewrite relation with R on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, denoted $\rightarrow_R$, is the smallest monotonic relation such that $l\sigma \rightarrow_R r\sigma$ for all $l \rightarrow r \in R$ and all substitutions σ. If $s \rightarrow_R t$ then we say that *s rewrites into t with R*. A rewrite system R is *convergent* if it is con¤uent and terminating; then every term t has a unique normal form w.r.t. $\rightarrow_R$, denoted by $nf_R(t)$, and $s = t$ is a logical consequence of R iff $nf_R(s) = nf_R(t)$.

A *Herbrand model* H of a set of clauses E is a set of congruence classes containing only ground terms referred to in E such that $H \models E$. A minimal Herbrand model H for a set of clauses E is a set of congruence classes such that $H \models E$, with the number of congruence classes in H less than or equal to the number in any other Herbrand model for E. An equality Herbrand interpretation is a Herbrand model constructed from a set

of ground terms and a congruence on those terms.

Let R be a set of ground equations or rewrite rules. Then the congruence $\leftrightarrow_R^*$ defines an equality Herbrand interpretation denoted by R^*, where the only predicate $=$ is interpreted by $s = t$ iff $s \leftrightarrow_R^* t$. We write $s = t \in R^*$ if $s \leftrightarrow_R^* t$. R^* satisfies (or is a model of) a ground clause $\Gamma \longrightarrow \Delta$, denoted by $R^* \models \Gamma \longrightarrow \Delta$, if $R^* \not\subseteq \Gamma$, or $R^* \cap \Delta \neq \emptyset$. The empty clause $\Box$ is hence satisfied by no interpretation. R^* satisfies a set of clauses S, denoted by $R^* \models S$, if it satisfies every clause in S.

A (strict partial) *ordering* on $\mathcal{T}(\mathcal{F},\mathcal{X})$ is an irre¤exive, transitive relation $\succ$. It is a *reduction ordering* if it is well-founded and monotonic, and stable under substitutions: $s \succ t$ implies $s\sigma \succ t\sigma$ for all substitutions σ.

An example of such an ordering is the *recursive path ordering* (RPO). For this ordering, we first set a strict precedence on the symbols in our signature $\mathcal{F}$, $\succ_{\mathcal{F}}$. We extend this to a suitable ordering $\succ$ on $\mathcal{T}(\mathcal{F},\mathcal{X})$, by the following scheme:

$$\begin{array}{rcll} f(s_1,\ldots,s_n) & \succ & s_i & \forall i, 1 \leq i \leq n \\ f(s_1,\ldots,s_n) & \succ & g(t_1,\ldots,t_m) & iff \succ_{\mathcal{F}} g, f(s_1,\ldots,s_n) \succ t_1,\ldots,t_m \\ f(s_1,\ldots,s_i,\ldots,s_n) & \succ & g(s_1,\ldots,s_{i-1}, & \\ & & t_1,\ldots,t_k,s_{i+1},\ldots,s_n) & iff = g \text{ or } f \succ_{\mathcal{F}}, \\ & & & s_i \succ t_1,\ldots,s_i \succ t_k, k \geq 0 \end{array}$$

4.2 First-Order Theorem Proving

Automated first-order theorem proving really took off with the advent of Robinson's *resolution inference rule* and *£rst-order uni£cation algorithm*, [Robinson, 1965]. We give the resolution inference rule below. Unification is the process of finding substitutions σ such that $s\sigma = t\sigma$ for some terms s,t. The crucial point is that the resolution inference rule is in some sense complete (as explained below), and the first-order unification algorithm is decidable. This means that first-order theorem proving can be attempted completely automatically using Robinson's methods.

Although many refinements have been made to Robinson's technique, the basic idea remains the same. Suppose we have a set of *axioms* A. Axioms are the formulae we know to be true or assume to be true without proof in our intended semantics. Additionally, suppose we have a *conjecture* C, that is a formula we would like to prove

to be a logical consequence of our axioms. We must first convert the problem into a suitable form for the resolution rule. To accomplish this, quantifiers are stripped from the formulae by moving universal quantifiers to the outside, and by Skolemising existentially quantified variables. Skolemising is the process of replacing existentially quantified variables with fresh *Skolem functions* that indicate the dependence of the existential variable on universally quantified variables. For example,

$$\exists Y P(X,Y)$$

is transformed to

$$P(X, f(X))$$

with f our new Skolem function. Then we stratify the logical connectives $\neg, \wedge, \vee$, putting the axioms and conjectures into *conjunctive normal form*, i.e. clauses of the form we described in §4.1:

$$\neg P_1 \vee \ldots \vee \neg P_n \vee Q_1 \vee \ldots \vee Q_m \tag{4.1}$$

We call the individual $\neg P_i$ and Q_j *literals*, and the number i or j identifying the literal is known as the *index* of the literal. We will sometimes use the $\Gamma \longrightarrow \Delta$ form for denoting clauses in the rest of this section when it makes the inference rules clearer.

The conjecture C is negated, and the idea then is to prove C by deriving a contradiction from $\neg C \cup A$. The resolution inference rule is defined like this:

Resolution:

$$\frac{P, \Gamma_1 \longrightarrow \Delta_1 \qquad \Gamma_2 \longrightarrow \Delta_2, Q}{(\Gamma_1, \Gamma_2 \longrightarrow \Delta_1, \Delta_2)\sigma} \quad \text{with } \sigma \text{ the most general unifier of } P \text{ and } Q$$

We call the conclusion of the resolution inference rule the *resolvent*. The version of the resolution rule given here is known as *binary resolution*, as there are only two resolving literals, P and Q. In order for this form of resolution to be *refutation complete*, i.e. guaranteed to find the empty clause from an inconsistent set of clauses in finite time, we require a rule known as *factoring*:

Factoring:

$$\frac{Q_1 \vee \ldots \vee Q_i \vee \ldots \vee Q_j \vee \ldots \vee Q_n}{(Q_1 \vee \ldots \vee Q_i \vee \ldots \vee Q_{j-1} \vee Q_{j+1} \vee \ldots \vee Q_n)\sigma} \quad \text{with} \quad \sigma \text{ the most general unifier of } Q_i \text{ and } Q_j$$

The process of searching for the empty clause is typically organised as follows: clauses are stored in two sets, the *usable* set and the *worked off* set. In general, we start with all clauses in the usable set, though if we know our axioms are consistent we may use the so-called *set of support* strategy, in which case only the conjecture clauses are placed in the usable set, and the axioms in the worked off set.

Proof search then proceeds by picking a clause from the usable set, typically by a simple weight heuristic, e.g. the clause with the fewest variables and function symbols is chosen first. This clause is then called the *given clause*. The given clause is considered for resolution against all the other clauses in the worked off and usable sets, unless we are using the set of support strategy in which case given clauses are only considered for resolution against worked off clauses. Any resolvents produced from these inferences are placed in the usable set. The given clause is then placed in the worked off set. This continues until the given clause has weight 0, i.e. it is the empty clause.

4.2.1 Modern Provers

In the years since Robinson first proposed the resolution calculus, a number of important refinements have been made which have significantly increased the power of automated first-order provers. One refinement has been to introduce slight variants to the inference rules in order to better handle theories with equality. The *paramodulation* calculus was introduced by Robinson and Wos, [Robinson and Wos, 1969], and later the *superposition* calculus was developed by Bachmair and Ganzinger amongst others, [Bachmair and Ganzinger, 1990, Bachmair and Ganzinger, 1994]. In the latter, techniques from rewriting theory are used to restrict the application of the inference rule while still preserving completeness. This requires us to choose well-founded ordering on the terms in our theory, $\succ$. Informally, the idea is that if our set of clauses

is inconsistent, then we can get to the empty clause by deriving new clauses which are smaller (w.r.t. $\succ$) than those it was derived from. The superposition calculus for an equational problem requires the following four inference rules[1]. The informal idea of the superposition rules is to use a positive equation from the left premise as a rewrite rule to simplify the rightmost clause.

Superposition left:

$$\frac{\Gamma_1 \longrightarrow l = r, \Delta_1 \quad \Gamma_2, s = t \longrightarrow \Delta_2}{(\Gamma_1, \Gamma_2, s[r]_p = t \longrightarrow \Delta_1, \Delta_2)\sigma}$$

if $\sigma = mgu(l, s|_p)$ and s is not a variable, and $l\sigma \succ r\sigma$, and $l\sigma = r\sigma$ is maximal w.r.t. $\succ$ in $(\Gamma_1 \longrightarrow l = r, \Delta_1)\sigma$, and $s\sigma \succ t\sigma$ and $s\sigma = t\sigma$ is maximal in $(\Gamma_2, s = t \longrightarrow \Delta_2)\sigma$

Superposition right:

$$\frac{\Gamma_1 \longrightarrow l = r, \Delta_1 \quad \Gamma_2 \longrightarrow \Delta_2, s = t}{(\Gamma_1, \Gamma_2 \longrightarrow s[r]_p = t, \Delta_1, \Delta_2)\sigma}$$

if $\sigma = mgu(l, s|_p)$ and s is not a variable, and $l\sigma \succ r\sigma$ and $l\sigma = r\sigma$ is maximal in $(\Gamma_1 \longrightarrow \Delta_1, l = r)\sigma$, and $s\sigma \succ t\sigma$ and $s\sigma = t\sigma$ is maximal in $(\Gamma_2 \longrightarrow \Delta_2, s = t)\sigma$ and $(s = t)\sigma \not\preceq (l = r)\sigma$.

Equality factoring:

$$\frac{\Gamma \longrightarrow \Delta, l = r, s = t}{(\Gamma, r = t \longrightarrow \Delta, s = t)\sigma}$$

if $\sigma = mgu(l, s)$ and $r\sigma \not\succeq l\sigma$ and $l\sigma = r\sigma$ is maximal in $(\Gamma \longrightarrow \Delta, l = r, s = t)\sigma$

Equality resolution:

$$\frac{\Gamma, l = r \longrightarrow \Delta}{(\Gamma \longrightarrow \Delta)\sigma}$$

if $\sigma = mgu(l, r)$, and $l\sigma = r\sigma$ is maximal in $(\Gamma, l = r \longrightarrow \Delta)\sigma$

Additionally, we can use information from our ordering $\succ$ to check for *redundancy*. If we derive a new clause that can itself be derived from smaller clauses we already

[1]For superposition left and right, the given clause is the rightmost premise

have, the new clause is redundant and so is pruned from the search space. More formally, we write $S^{\prec c}$ to denote the set of all ground instances of clauses S that are smaller than a ground clause c w.r.t. $\succ$, and then define:

De£nition 1 *1. A ground clause c is* redundant *in a set of clauses S if* $S^{\prec c} \models c$. *Similarly, a non-ground clause is redundant if all its ground instances are.*

De£nition 2 *A ground inference with rightmost premise c and conclusion* c' *is* redundant *in a set of clauses S if* $S^{\prec c} \models c'$. *Similarly, a non-ground inference is redundant if all its ground instances are.*

De£nition 3 *A set of clauses S is* saturated *if all inferences with premises in S are redundant in S.*

If we derive a set of saturated clauses S, then we know that it is consistent, so there is no chance of deriving the empty clause. Techniques exist for constructing a model from a saturated S (see Definition 11).

In order to determine whether a newly derived clause is redundant in practice, a number of techniques have been proposed. One is to check for *subsumption*, i.e. if a clause we have newly derived is a more specific instance of one we already have, then it is redundant. In general, checking for subsumption is an NP-complete problem, but various algorithms have been proposed that produce acceptable performance on a good number of cases, [Gottlob and Leitsch, 1985].

A simpler test for redundancy is to look for *tautologies*, for example clauses of the form $\Gamma, E \longrightarrow \Delta, E$ or $\Gamma \longrightarrow \Delta, t = t$. Most modern provers employ rules to detect these kinds of clauses and a number of other such rules.

As well as rules for testing redundancy, modern provers also employ a number of rules which simplify the derived clause. These are known as *reduction rules*. For example, provers often employ some kind of rewriting based reductions, based on the rewrite rules inferred from other clauses. The theorem prover SPASS, [Weidenbach et al., 1999], employs the following rules for *local rewriting*:

$$\frac{\Gamma_1 \longrightarrow \Delta_1, s = t \quad \Gamma_2, E \longrightarrow \Delta_2}{\begin{array}{c}\Gamma_1 \longrightarrow \Delta_1, s = t \\ \Gamma_2, E[t\sigma]_p \longrightarrow \Delta_2\end{array}} \quad \text{if} \quad \begin{array}{l} E|_p = s\sigma,\ s = t \text{ is strictly maximal} \\ \text{in } \Gamma_1 \longrightarrow \Delta_1, s = t, \\ \text{and } s \succ t \text{ and } \Gamma_1\sigma \subseteq \Gamma_2, \Delta_1\sigma \subseteq \Delta_2 \end{array}$$

$$\frac{\Gamma_1 \longrightarrow \Delta_1, s = t \quad \Gamma_2 \longrightarrow \Delta_2, E}{\begin{array}{c}\Gamma_1 \longrightarrow \Delta_1, s = t \\ \Gamma_2 \longrightarrow \Delta_2, E[t\sigma]_p\end{array}} \quad \text{if} \quad \begin{array}{l} E|_p = s\sigma,\ s = t \text{ is strictly maximal} \\ \text{in } \Gamma_1 \longrightarrow \Delta_1, s = t, \\ \text{and } s \succ t \text{ and } \Gamma_1\sigma \subseteq \Gamma_2, \Delta_1\sigma \subseteq \Delta_2 \end{array}$$

Another refinement to the method is *literal selection*, the rough idea being that, since we need to eliminate all literals in a clause to obtain a refutation, it cuts down the search space if we decide to work on eliminating one literal at a time. For this we define a function that selects a (possibly arbitrary) negative literal from a clause, and then only that literal is considered as an equation $s = t$ in a superposition left inference. Only maximal selected literals (w.r.t. our ordering $\succ$) are considered for the other equations in superposition inferences. Superposition remains complete under selection. A common strategy is to select a literal in a clause only if that clause has more than one maximal literal.

4.2.2 Memory Allocation

A major improvement to the efficiency of theorem prover implementation has resulted from the use of *term indexing*. This is used both to save memory, and to speed up operations such as searching for unifying literals for use in inference and reduction rules. The principle of the method is to try to store every subterm that occurs in the worked off or usable set only once, together with a set of links to superterms containing that subterm. These superterms are linked back to their own superterms, and hence back to literals and then to clauses. Normalisation techniques are used to allow non-ground terms to be shared more efficiently. This saves memory, and therefore also allows the program to run faster.

To see how term indexing also allows inferences and reductions to be calculated faster, suppose we have a given clause $\Gamma, s = t \longrightarrow \Delta$ and we want to consider it for

inferences by superposition left on our selected literal $s = t$. We can retrieve possible right premises for the inference rule application by looking at the subterms of s, extracting unifying subterms from the index, and then tracing back their superterm links. This avoids the potentially very laborious process of looking through all the clauses we have, looking through each literal in each clause and then looking in each literal for a unifying term. In terms of efficiency improvement, term indexing is probably the biggest implementation based advance to have been made in theorem proving in the last 20 years.

4.2.3 Answer Extraction

Some applications of theorem proving require that we extract an answer, i.e. we prove an existential goal $\exists x.P(x)$ and extract a witness t satisfying $P(t)$. This can be done in resolution style theorem proving by the use of *answer literals*, [Green, 1969]. These literals contain an occurrence of all the universally quantified variables[2] that we are interested in an instantiation of. As unifications are found and applied to the literals in a clause, they are also applied to the answer literals. However, the answer literals are hidden in the sense that they don't count towards the weighting of a clause, and they are never used for inferences. When the empty clause is derived, we simply look in the answer literal and read off the value of the variable, yielding our answer t.

Answer extraction extends naturally from the resolution calculus to equational calculi like the superposition calculus. We just need to accumulate the unifiers in the same way. Some recent work has shown completeness of the method for equational calculi (i.e. if we find all the proofs of a $\exists x.P(x) = y$ conjecture, we will have obtained all valid answers t such that $P(t) = y$), [Lynch, 1997].

[2]Recall that we negate a conjecture in order to try and prove it, so our existentially quanti£ed variables in the original conjecture will now be universally quanti£ed.

4.3 Overview of Comon-Nieuwenhuis Method

The proof by consistency technique was developed to solve problems in equational theories, involving a set of equations defining the *initial model*[3], E. We will refer to these as the *axioms*. The problem then is to establish the truth (or otherwise) of a conjecture C with respect to this equational system, where C is not an equational theorem of the system (i.e. there may be models of E in which C is not true), but C may be an *inductive* theorem of E, i.e. it is true in the initial model.

The approach is to produce a first-order axiomatisation $\mathcal{A}$ of the minimal Herbrand model such that $C \cup \mathcal{A} \cup E$ is consistent if and only if C is an inductive consequence of E. As we saw in §4.2.1, modern first-order theorem provers have two modes of termination: one in which a contradiction is derived and the (negated) conjecture is refuted, and one where the prover ascertains that a refutation cannot be found, and so the conjecture is consistent with the axioms. If we can show consistency with our axiomatisation $\mathcal{A}$ and the axioms E, then we have shown that C is a theorem of the initial model of E. Note that this is the other way round to ordinary resolution theorem proving, where we check the consistency of $E \cup \neg C$, and if the consistency check fails (i.e. we derive the empty clause), we have proved C. In proof by consistency, if our consistency check of $C \cup \mathcal{A} \cup E$ fails, then we have shown that C is **not** an inductive theorem of E.

4.4 The Comon-Nieuwenhuis Method for Horn Clauses

Although the Comon-Nieuwenhuis method can be applied to non-Horn specifications, the method for Horn clauses is easier to understand, as we have a unique minimal model of the axioms. All the security protocol problems in this thesis are specified in terms of Horn clauses.

We assume the axioms contain only equations, i.e. that equality is the only predicate symbol. Other specifications could be transformed to meet this requirement by converting literals A into $A = true$, with *true* a new, minimal function symbol. We will

[3]The initial or standard model is the minimal Herbrand model. This is unique in the case of a Horn or purely equational speci£cation.

also firstly assume that our set of axioms is saturated under superposition and equality resolution (i.e. they could be written as a convergent rewrite systems). Later we will show how to drop this requirement.

Suppose we have a saturated set of Horn clauses E, a set of conjectures C, and a total ordering $\succ$ on the terms in E. Then there is a unique minimal Herbrand model of E, which we will call I. In order to reduce the problem of proving or disproving C to first-order consistency, we require an *I-Axiomatisation*:

De£nition 4 *A set of £rst-order formulae $\mathcal{A}$ is an* I-Axiomatisation *of I if*

1. *$\mathcal{A}$ is a set of purely universally quanti£ed formulae*
2. *I is the only model of $E \cup \mathcal{A}$ up to isomorphism.*

$\mathcal{A}$ must therefore contain enough negative information to rule out all non-minimal Herbrand models. Assuming a total ordering on terms, $\succ$, we define a *normal I-Axiomatisation*.

De£nition 5 *A ground term t is* normal *if it is the minimum (w.r.t. $\succ$) representative of its congruence class in I. A ground clause c is normal if all terms in c are normal. A substitution σ is normal if $x\sigma$ is normal for all x in the domain of σ.*

De£nition 6 *Let $\mathcal{A}$ be a set of £rst-order clauses s.t. $I \models \mathcal{A}$ and $\mathcal{A} \models s \neq t$ for all pairs of distinct normal terms s and t. Then $\mathcal{A}$ is a* normal I-Axiomatisation.

A proof that a normal I-Axiomatisation is indeed an I-Axiomatisation can be found in [Comon and Nieuwenhuis, 2000]. Not all I-Axiomatisations are normal, however we need our I-Axiomatisations to be normal for Theorem 2 below. Normal I-Axiomatisations are also convenient to construct. For example, suppose E is the following specification of natural numbers and the plus symbol:

$$\begin{aligned} 0 + x &= x \\ s(x) + y &= s(x + y) \end{aligned}$$

Then the following two formulae constitute a normal I-Axiomatisation, with $\succ$ defined such that $+$ is bigger than s or 0:

$$\begin{array}{rcl} 0 & \neq & s(x) \\ s(x) = s(y) & \longrightarrow & x = y \end{array}$$

This theorem about I-Axiomatisations states a property central to the Comon-Nieuwenhuis technique:

Theorem 1 *If $\mathcal{A}$ is an I-Axiomatisation, then $\mathcal{A} \cup E \cup C$ is consistent $\iff I \models C$*

Proof: If $I \models C$, then $I \models \mathcal{A} \cup E \cup C$, and hence $\mathcal{A} \cup E \cup C$ is consistent. Conversely, if the set $\mathcal{A} \cup E \cup C$ is consistent, then it has a Herbrand model, as it is a set of purely universal formulas. Now, this model must be I, as I is the only Herbrand model of $E \cup \mathcal{A}$. $\Box$

Comon and Nieuwenhuis discuss techniques for constructing I-Axiomatisations for various classes of problem. We will look at this issue in §4.6 below.

We will now move on to the method of showing the consistency of $\mathcal{A} \cup E \cup C$. Informally, an attempt to show consistency involves two parts. In one, we pursue a *fair induction derivation*. This is a restricted kind of saturation, where we need only consider overlaps between axioms and conjectures, and produce inferences from an adapted superposition rule. In the second part, every clause in the induction derivation is checked for consistency against the I-Axiomatisation. If any consistency check fails, then the conjecture is incorrect. If they all succeed, and the induction derivation procedure terminates, the theorem is proved.

We now describe the first part of the method more formally. The only inference rule required by the prover is *conjecture superposition*. This is a superposition rule restricted to simplification of conjectures by axioms. The left premise of the rule is a (definite) Horn clause from E and the right premise is a conjecture c from C:

Conjecture superposition:

$$\frac{D \vee l = r \quad c}{(D \vee c[r]_p)\sigma} \quad \text{if} \quad \begin{array}{l} \sigma = mgu(c|_p, l) \text{ and } c|_p \text{ is not a variable, and} \\ \text{for some ground } \theta\text{:} \\ l\sigma\theta \succ (r\sigma\theta, D\sigma\theta)\text{, and, if } p \text{ is inside } s \text{ in a} \\ \text{negative literal } s = t \text{ of } c \text{ then } s\sigma\theta \succ t\sigma\theta \end{array}$$

The notion of redundancy is slightly different to that in conventional saturation-based theorem proving, in that it includes the use, without ordering restrictions, of any formula known to be true in I. This includes *lemmas*, i.e. arbitrary first-order clauses L such that $I \models L$. We might have proved these in previous runs of the Comon-Nieuwenhuis method or by other means. Redundancy is also defined with respect to unproved conjectures which are smaller with respect to $\succ$ than our possibly redundant conjecture. This is where the essence of induction is found - the use of smaller conjectures in this manner is similar to applying an induction hypothesis. Let $C^{\prec c}$ be the set of all formulae in C smaller than conjecture c with respect to our ordering $\succ$. We then define the following:

De£nition 7 *A ground conjecture c is* redundant *in a set of conjectures C if $E \cup \mathcal{A} \cup L \cup C^{\prec c} \models c$. A non-ground conjecture c is redundant if all its ground instances are.*

De£nition 8 *A ground inference by superposition with rightmost premise c and conclusion c' is* redundant *in a set of conjectures C if $E \cup \mathcal{A} \cup L \cup C^{\prec c} \models c'$. Similarly, a non-ground inference is redundant if all its ground instances are.*

De£nition 9 *A set of conjectures C is* saturated *if all inferences by conjecture superposition with rightmost premises in C are redundant in C.*

We can now define a *fair induction derivation*. Informally, it is a sequence of sets of conjectures in which all conjectures are, at some finite point in the sequence, shown to be redundant, or considered for conjecture superposition.

De£nition 10

1. An *induction derivation* is a sequence of sets of conjectures $C_0, C_1, \ldots$ such that each C_{i+1} is obtained from C_i either by adding to C_i a logical consequence of $E, \mathcal{A}, L, C_i$ or by removing from C_i some conjecture that is redundant in C_i.

2. A conjecture is *persistent* in the derivation if for some j it belongs to all C_k with $k \geq j$.

3. A derivation is fair if every conjecture superposition inference with a persistent rightmost premise is redundant in C_j for some j.

The second key theorem of the technique is this:

Theorem 2 *Let $\mathcal{A}$ be a normal I-Axiomatisation, and $C_0, C_1, \ldots$ be a fair induction derivation. Then $I \models C_0$ iff $\mathcal{A} \cup \{c\}$ is consistent for all clauses c in $\cup_i C_i$.*

To prove this theorem, we need a lemma, which states the fact that our inference system only need reduce non-normal clauses:

Lemma 1 *Suppose $\mathcal{A}$ is a normal I-Axiomatisation and c is a clause. Let $c\sigma$ be a normal clause such that $I \not\models c\sigma$. Then $\mathcal{A} \cup \{c\}$ is inconsistent.*

Proof: (from [Comon and Nieuwenhuis, 2000]) $\mathcal{A} \cup \{c\sigma\}$ is inconsistent if (i) every model of $\mathcal{A}$ satisfies the negative literals of $c\sigma$, and, (ii) no model of $\mathcal{A}$ satisfies any of the positive ones. If $I \not\models c\sigma$ then $I \models s = t$ for all negative equations $s = t$ in $c\sigma$. Since s and t are both normal, it must be the case that $s \equiv t$, which implies (i). For (ii), let $u = v$ be a positive equation in $c\sigma$. Since $I \not\models c\sigma$, we have $u \not\equiv v$, and hence, by the normality of $\mathcal{A}$, $\mathcal{A} \models u \neq v$, and hence no model of $\mathcal{A}$ satisfies $u = v$. This implies the inconsistency of $\mathcal{A} \cup \{c\sigma\}$, and hence of $\mathcal{A} \cup \{c\}$. □

We also require a definition of how to construct a model R^* for a saturated set of (Horn) clauses. The idea is to define a rewrite system R based on the rules generated by the clauses, and then to use this to define a congruence for constructing a model.

Definition 11 *An instance G of the form, $\Gamma \longrightarrow l = r$, of a clause in E* generates *the rule $l \rightarrow r$ if*
1. $R_G^* \not\models G$,
2. $l \succ r$, and $(l = r) \succ_e e$ for all equations e in Γ,
3. l is irreducible by R_G. where R_G is the set of rules generated by all instances D of clauses in E such that $G \succ_G D$. We denote by R the set of rules generated by all ground instances of E.

Theorem 3 *(from [Bachmair and Ganzinger, 1994]) The ground TRS R is convergent. Furthermore, if E is saturated under superposition and equality resolution then either $\square \in E$ or else R^* is a model for E, i.e. E is consistent. More precisely, I is isomorphic to R^*.*

Equipped with these definitions and results, we are now in a position to prove Theorem 2.

Proof of Theorem 2: For the forward implication, suppose $\exists$ some $c \in$ some C_j s.t. $\mathcal{A} \cup \{c\}$ is inconsistent, then $I \not\models C_0$, since all such c are logical consequences from $E, \mathcal{A}, L, C_0$ and $I \models E, \mathcal{A}, L$.

For the reverse implication, assume $\mathcal{A} \cup \{c\}$ is consistent $\forall c \in \cup_i C_i$. Suppose for contradiction that $I \not\models C_0$. Then we derive a contradiction from the existence of a minimal (w.r.t. $\succ$) ground instance $c\sigma$ of a clause c in $\cup_i C_i$ s.t. $I \not\models c\sigma$.

If c is redundant in some C_j then from the definition of redundancy of conjectures, it follows that there is some false instance of a conjecture in C_j that is smaller than $c\sigma$, contradicting the minimality of $c\sigma$.

Otherwise c must be persistent. By lemma 1, $c\sigma$ is not a normal clause, otherwise $\mathcal{A} \cup \{c\}$ would be inconsistent. Furthermore, σ must be normal, since otherwise σ must be reducible by R (see definition 11) into some σ' such that $I \not\models c\sigma'$ and $c\sigma \succ c\sigma'$, contradicting the minimality assumption on $c\sigma$. Since $c\sigma$ is therefore not a normal clause, $c\sigma$ is reducible at some position p by a rule $l\theta \to r\theta$ in R, generated by some clause $D \vee l = r$ in E. Then $(c\sigma)|_p \equiv l\theta$ and hence $c|_p$ and l are unifiable by some mgu α.

Furthermore, if $c\sigma$ is reducible by some rule in R only by some negative literal $s\sigma = t\sigma$, then, since $I \models s\sigma = t\sigma$, the normal forms of $s\sigma$ and $t\sigma$ w.r.t. R coincide. Hence, if $s\sigma \succ t\sigma$, the maximal side $s\sigma$ is reducible by R. Therefore, there exists some inference by conjecture superposition

$$\frac{D \vee l = r \quad c}{(D \vee c[r]_p)\alpha}$$

whose conclusion has an instance $c\sigma[r\theta]_p$ such that $I \not\models c\sigma[r\theta]_p$ and moreover $c\sigma \succ c\sigma[r\theta]$. By fairness of the induction derivation, this conclusion is redundant in some

C_j. But then from the definition of redundancy of inferences, it follows that there is some false instance of a conjecture in C_j that is smaller than $c\sigma$, contradicting the minimality of $c\sigma$. □

A key point about Theorem 2 is that it also gives us refutation completeness.

Corollary 1 *The Comon-Nieuwenhuis system is refutation complete, i.e. will refute any conjecture c inconsistent with $E \cup \mathcal{A}$ in £nite time.*

Proof: Suppose c is our conjecture such that $I \not\models c$. Then the second half of the proof of Theorem 2 shows that $A \not\models \{c'\}$ for some c' in $\cup_i C_i$. By fairness of the induction derivation, we will eventually consider this c'. All that is required is that we do indeed detect the first-order inconsistency between c' and the I-Axiomatisation. If we use a standard resolution-style theorem prover, for example, then as observed in §4.2, we have the completeness required. □

Having refuted a non-theorem in this way, we now have an overall proof of the falsehood of our original universally quantified inductive conjecture C. This consist of two parts, first a derivation of some c' via conjecture superposition inferences, and then a derivation of the empty clause from $c' \cup \mathcal{A}$ by standard first-order resolution type methods. This refutation constitutes a resolution style proof of the theorem $\neg C$. Hence, we can extract the counterexample, t, using Green's answer literals, as described in §4.2.3.

Note that we used the normality of $\mathcal{A}$ in the proof of Theorem 2. In fact, this is required not just for this particular proof but for the theorem itself to hold. For a counterexample, suppose we have a set of axioms $E = a = b, c = d$ for constants a, b, c, d, ordering $a \succ b \succ c \succ d$, I-Axiomatisation $\mathcal{A} = a \neq c$, and C the false conjecture $b = d$. We now have no available conjecture superposition inferences that can lead us to $a = c$, and so detection of the inconsistency is not possible. In §6.9, we show that the I-Axiomatisation we use in our security protocol model is normal, so preserving refutation completeness.

4.5 Non-saturated Sets of Axioms

A key advance on previous proof by consistency techniques made by the Comon-Nieuwenhuis technique is its ability to handle non-saturated sets of axioms E, i.e. specifications which cannot be expressed as a convergent rewrite system. Instead, we require that E is a *reductive de£nition*, which is defined in terms of constructors. We explain this part of the method here.

We defined constructors informally in §3.2. More formally, let $\mathcal{F}$ be the union of $\mathcal{F}_0$, a non-empty set of *constructor symbols*, and $\mathcal{D}$, a set of *de£ned symbols*. Assume E_0 is a saturated subset of E build over $\mathcal{T}(\mathcal{F}_{\prime},\mathcal{X})$. Terms in $\mathcal{T}(\mathcal{F}_{\prime},\mathcal{X})$ are called *constructor terms*. The constructors are called *free* if E_0 is empty. We additionally assume E is sufficiently complete, i.e. for every ground term s there is some ground constructor term t such that $E \models s = t$.

As an example of a simple constructor specification, the two equations we gave after Definition 6 in §4.4 specify $+$ as a defined symbol in terms of constructors s and 0. As an example of a function that cannot be defined in a convergent rewrite system, we give the *gcd* function, as shown in Example 1. Our constructors here are s and 0, and $+$ and *gcd* are the defined symbols i.e. $+$ is defined in terms of s and 0, and *gcd* is defined in terms of $s, 0$ and $+$. Our term ordering $\succ$ should ensure that all defined symbols are larger than all constructor symbols, e.g. by using the RPO with precedence $gcd \succ + \succ s \succ 0$.

Example 1

$$E = \left\{ \begin{array}{rcl} x+0 & = & x \\ s(x)+y & = & s(x+y) \\ gcd(0,x) & = & x \\ gcd(x,0) & = & x \\ gcd(x,x+y) & = & gcd(x,y) \\ gcd(x+y,y) & = & gcd(x,y) \end{array} \right.$$

This is not a saturated specification, but it is what Comon and Nieuwenhuis call a *reductive de£nition*:

De£nition 12 *Let E be a possibly non-saturated constructor-based speci£cation where $E = E_0 \cup E_1$ is a set of Horn clauses. E_0 contains clauses axiomatising constructor terms, and E_1 contains clauses de£ning the de£ned symbols.*

Furthermore, assume that for every ground term u of the form $f(t_1,...,t_n)$ where $t_j \in \mathcal{T}(\mathcal{F}_0)$ for $j = 1..n$ and $f \in \mathcal{D}$, there is some clause in E_1 with an instance $\Gamma \longrightarrow l = r$ such that $E \models \Gamma, l = u$, l is headed by f, and $u \succ (\Gamma, r)$. Then E is called a reductive definition.

To see that our *gcd* definition in Example 1 is reductive, take any term u of the form $gcd(s^n(0), s^m(0))$. If n (respectively m) is 0, then u is equivalent to the smaller term s^n (respectively s^m). Otherwise, without loss of generality, let $m = n + n'$. Then $E \models gcd(s^n(0), s^{n+n'}(0)) = gcd(s^n(0), s^n(0) + s^{n'}(0))$, which can be reduced by the fifth rule to $gcd(s^n(0), s^{n'}(0))$, which is smaller w.r.t. $\succ$ than u. Hence Example 1 is a reductive definition.
In the proof of Theorem 2, sautratedness was used for showing that if $c\sigma$ is not normal, then $c\sigma$ is reducible by R. The purpose of this is to show that for non-normal $c\sigma$ such that $I \not\models c\sigma$, there is some conjecture superposition inference yielding a smaller false conjecture. We can also show this in the case of a reductive definition, as long as our conjecture has a *de£nition pattern*:

De£nition 13 *A* definition pattern *is a term of the form $f(x_1, \ldots, x_n)$ where f is a de£ned symbol and x_i and x_j are distinct variables for $1 \leq i, j, \leq n$, $i \neq j$.*

Lemma 2 *Let E be a reductive de£nition and let c be a conjecture such that $c|_p$, with p the position of the innermost de£ned symbol, is a de£nition pattern.*

Then for every ground instance $c\sigma$ where σ is normal, there exists some inference by conjecture superposition at position p with a conclusion c', and a normal substitution σ' such that $I \not\models c\sigma$ implies $I \not\models c'\sigma'$, and furthermore, $c\sigma \succ c'\sigma'$.

Proof: Let $c|_p$ be $f(x_1, \ldots, x_n)\sigma$. By Definition 12, there exists an instance $c_2 \vee l = r$ of a clause c_1 in E_1 such that $E \models l = s$, l is headed with f, and $s \succ (c_2, r)$. Furthermore, since σ is normal, $l \succeq s \succ r$. Hence there exists an inference of c_1 on c, whose conclusion has an instance of the form $c_2 \vee c\sigma[r]_p$ with the desired properties. $\square$.

We can appeal to this lemma instead of saturatedness to complete the proof of Theorem 2 in the case where E is not saturated, but is a reductive definition. However, we must be sure we have a definition pattern. In [Comon and Nieuwenhuis, 2000, p.21-22], Comon and Nieuwenhuis give details of how this can be achieved using *variable abstractions*.

Our formalism for the security protocol problem is a reductive definition rather than a saturated one. However, it has a very simple equational part of the theory, and defined symbols are always the outermost function symbols in a literal, allowing us to recover refutation completeness without recourse to the variable abstraction mechanism. The details of this are given in §6.8, after we have presented the formalisation itself. The idea is to prove a stronger form of Lemma 2 without requiring definition patterns, and thence to prove Theorem 2 for our formalism.

4.6 Finding I-Axiomatisations

There are a number of ways of obtaining a suitable set of clauses to be an I-Axiomatisation $\mathcal{A}$. In Musser's original setting for proof by consistency, [Musser, 1980], he assumed the presence of a *completely de£ned equality function*, i.e. a function *eq* with the property that two terms are equal under the congruence relation given by the equation set E if and only if $eq(s,t) = true$, and unequal if and only if $eq(s,t) = false$. *true* and *false* are assumed to be irreducible.

Lemma 3 *In this situation, a suf£cient I-Axiomatisation is the single equation true* $\neq$ *false.*

Proof: $I \models \mathcal{A}$ as $true \neq_E false$ as soon as the initial algebra is not trivial. If $M \models E \cup \mathcal{A}$, then for any two ground terms s,t, if $I \not\models s = t$, then the normal forms of s and t are different. It follows that $E \models eq(s,t) = false$, hence $M \models eq(s,t) = false$. Since $M \models true \neq false$, $M \not\models eq(s,t) = true$, hence $M \not\models s = t$.
For any two ground terms, we have $M \models s = t$ iff $I \models s = t$. If M is a Herbrand model, it is isomorphic to I. □

For problems for which we have a set of free constructors, we can use Huet and Hullot's technique, [Huet and Hullot, 1982].

De£nition 14 *A set of constructors* C *is* free *if:*
$E \cup \{s \neq t | s,t \in T(C), s \not\equiv t\}$ *is consistent.*

If we have this condition for our theory E, then we simply define $\mathcal{A}$ to express the inequality of any two non-identical constructors, like this:

For every constructor symbol c:
$\forall x_1,\ldots,x_n,y_1,\ldots,y_n.c(x_1,\ldots,x_n) = c(y_1,\ldots,y_n) \longrightarrow x_1 = y_1 \wedge \ldots \wedge x_n = y_n$
For every pair of distinct constructors c, c':
$\forall x_1,\ldots,x_n,y_1,\ldots,y_n.c(x_1,\ldots,x_n) \neq c'(y_1,\ldots,y_n)$

Huet and Hullot also assume that E is a convergent rewrite system and, quite reasonably, that constructor terms are considered smaller than defined symbols.

Lemma 4 *Under Huet and Hullot's assumptions, $\mathcal{A}$ as speci£ed above is a normal I-Axiomatisation.*

Proof: Let s and t be distinct ground terms such that $s \succ t$ and s is minimal in its equivalence class. Then s is irreducible, by convergence of the rewrite system, hence s is a constructor term. Since $s \succ t$, t must also be a constructor term. Since s and t are distinct, then we must have $\mathcal{A} \models s \neq t$.

For problems with non-free constructors, we may be able to use an *inductive reducibility* predicate, as described by Jouannaud and Kounalis, [Jouannaud and Kounalis, 1989]. This applies when the problem can be specified in terms of a convergent rewrite system R. A term is defined to be inductively reducible w.r.t. R if all its ground instances are reducible. Our I-Axiomatisation $\mathcal{A}$ is the rewrite system R, and $\mathcal{A} \cup \{c\}$ is inconsistent iff c is an equation $l = r$, $l \not\equiv r$, and l is not inductively reducible. Alternatively, Comon and Nieuwenhuis describe a method for lifting the rewrite system to a finite set of clauses $\mathcal{A}$ so that consistency checking can be carried out with a standard prover[Comon and Nieuwenhuis, 2000, p. 32].

Comon and Nieuwenhuis also give a general procedure for determining I-Axiomatisations for Horn clause examples based on disunification, [Comon and Nieuwenhuis, 2000, p. 29]. However, the details are quite complex and it is not really relevant to our work. Our protocol model uses free constructors, as does Paulson's. We describe the I-Axiomatisation we use in §6.9.

4.7 Summary

In this chapter, we have defined the notation we will use in the thesis, and explained the theory of first-order theorem proving. We have given an overview of the Comon-Nieuwenhuis method for proof by consistency, and explained in detail how it works for the Horn clause case. The most important result is Theorem 2, which we have shown gives us refutation completeness. Following the proof of this theorem, we made the important observation that the answer extraction mechanism explained in §4.2.3 can be simple adapted for extracting counterexamples from our refutation.

We have also discussed the theory of construction I-Axiomatisations. For free constructor theories, such as the security protocol models developed in this thesis, it is very straightforward.

Chapter 5

System Description

This chapter is divided into three sections. First, we describe the adaptation of a first-order prover to pursue a fair induction derivation. Then we describe the parallel architecture of our system (which we decided to call CORAL) and explain how this facilitates the I-Axiomatisation check. Finally we give some results from testing CORAL.

5.1 Adapting a First-Order Prover

The aim of the system was an implementation of the Comon-Nieuwenhuis method for proof by consistency in a fast, modern theorem prover. The first stage of this task was to adapt a prover to construct fair induction derivations (Definition 10, p. 53). This included the 'conjecture superposition' inference rule (see §4.4), and the use of lemmas to show redundancy (also §4.4).

5.1.1 Choosing a First-Order Prover

The first issue addressed by our implementation work was the choice of first-order theorem prover to adapt for proof by consistency. We chose SPASS, because it had shown good performance in recent CASC first-order theorem prover competitions, [Pelletier et al., 2002], and because the source code is freely available from the SPASS website. The code is written in C and is well commented (in English), which also contributed to the decision to use it.

SPASS is described in [Weidenbach et al., 1999] and, more extensively, in [Weidenbach, 2001]. It features term indexing, an extensive set of reduction rules, and a variety of inference rules for different syntactic classes of problem.

5.1.2 Separation of Axioms and Conjectures

The inference rule described in §4.4 is similar to the normal superposition rule, except that it requires that only overlaps between conjectures and axioms be considered, i.e. the left premise is always an axiom and the right premise a conjecture. This means that our system has to keep derived clauses and axioms separate, rather than putting them together in a set of 'worked-off clauses' as is the standard practice in first-order theorem proving.

In common with all modern first-order theorem provers, SPASS uses term indexing to allow fast checking for clauses containing literals that unify. This is used both when looking for a clause to partner the given clause in an application of an inference rule, and also when performing redundancy checks on derived clauses. We only need to check for unifications between axioms and conjectures (or usable clauses) when considering inference rules, but when checking for redundancy we will need to check lemmas and previously derived (i.e. worked off) clauses as well. In this process, the worked off clauses and lemmas have the same status, so we can save some memory and time by putting them in a shared index together. This is indeed what CORAL does, producing a theorem prover with 3 sharing indexes, instead of the normal 2:

Axioms - used for conjecture superposition rule and redundancy checking

Worked Off - contains lemmas as well as worked off clauses, used for redundancy checking

Usable - contains derived clauses yet to be considered as given clauses

Input to SPASS is via a file in DFG syntax, [Hähnle et al., 1996]. This is a common syntax used in several provers, and translators exist to convert from and to this format to and from other commonly used standards. DFG syntax only contains provision for the input of axioms and conjectures. It would be a shame to deviate from this

format. So, in the CORAL system, instead of changing the format to add in a section for lemmas, we just use the existing label mechanism. Any axiom or conjecture in a DFG syntax file can have a label attached, which is then used when referring to the formula in the output of a proof. If no labels are given, SPASS creates its own labels, like 'axiom2' or 'conjecture1'. So, CORAL treats any axiom the user has given a label matching 'lemmaN' as a lemma, and adds it to the worked off index instead of the axiom index.

5.1.3 The Use of Lemmas in Reduction Rules

By our definition in §4.4, we are permitted to use both axioms, lemmas and worked off clauses in our proofs of redundancy. Lemmas in this context are taken to be theorems added by the user to the input set that have been proved already. Our implementation fully supports this. Some reduction rules required no modification to work with our newly organised shared indexes. These are the ones that only compare the clause being reduced to one other clause, such as subsumption checking (see §4.2)[1]. For these, we just need to repeat the subsumption check firstly with the axiom index, and then with the worked off index. This is precisely what CORAL does.

Only slightly more complicated is the implementation of rules such as rewriting, where several previously derived rules may be used. In the standard implementation of SPASS, this is done in one of two modes, depending on whether we have specified complete inter-reduction or lazy reduction at the command line. In complete inter-reduction mode, we may use rewrite rules inferred from either the worked off set or the usable set. In lazy reduction mode, we only reduce with respect to the worked off set. In CORAL, we made changes to allow the axiom set to be used for generating rewrite rules in both these modes, although for our protocol experiments we used only complete inter-reduction.

CORAL must exhaustively reduce a clause by rewriting, and in order to do this, it is not sufficient to exhaustively rewrite using rules from one index, and then again with the other indexes. It may be that by applying a rewrite rule from one index, we reduce

[1] The rules requiring no modi£cation were: trivial literal elimination (in SPASS notation `Obv`), unit con¤ict (`UnC`), tautology elimination (`Taut`), subsumption deletion (`Sub`), removal of redundant equations (`AED`), and condensation (`Con`)

a literal such that a rule from another index may apply. So where previously SPASS would have reduced a clause using rules from the worked off set, CORAL's rewriting loop checks first for applications of rules from the axiom index, and then from the worked off index. If either of these produces a reduction, then we go back and look again for a rule to apply to the newly reduced clause. When complete inter-reduction is specified, we combine our new mechanism for reducing with respect to rules from the axiom and worked off indexes with the existing mechanism for additionally reducing with respect to the usable set, allowing exhaustive reduction via rules from all three sets of clauses.

5.1.4 Recovery of the Counterexample

A system designed to detect incorrect conjectures is only of practical use if it can also exhibit counterexamples. For our proposed application to cryptographic protocols, a counterexample to a security property will constitute an attack on the protocol, the vital piece of evidence required to show that the protocol is indeed flawed.

To present counterexamples found, we used the same mechanism involving answer literals explained in §4.2. In order to implement this in CORAL, it was necessary to keep the answer literals 'unshared', i.e. not to place them into the term index. This stops them from being picked up as potential partners for an application of the inference or reduction rules.

When using CORAL for a problem where the counterexample is of interest, the `Ans` command line flag must be set. In this case, CORAL will move any answer literals in the conjecture to the end of the clause, and keep them unshared. To see the final answer, the standard SPASS flag `PEmp` should also be set. This means that when an empty clause is derived, it is printed out, revealing the answer.

5.2 I-Axiomatisation Checking

Theorem 2 requires that we check every clause in the fair induction derivation against an I-Axiomatisation. If an inconsistency is found here, this indicates an incorrect conjecture. The fair induction derivation process is not guaranteed to terminate. In order

to retain refutation completeness, we have to do our I-Axiomatisation checking at the same time as the fair induction derivation is being computed.

In implementing the I-Axiomatisation check in CORAL, we were faced with two choices: first, to combine I-Axiomatisation checking and inductive completion in a single super prover, whereby we would keep a separate I-Axiomatisation set of clauses in memory, and check each derived clause against it before adding it to the worked off set. Second, we could feed derived clauses off to a parallel I-Axiomatisation checker. We decided to opt for the latter option, for several reasons. One is that it allows us to carry out the check at the same time as the induction derivation, as required in the specification, but without resulting in a large, messy and unwieldy program. Another advantage is that we are able to easily devote a machine solely to inductive completion in the case of harder problems. It is also very convenient when testing a new model to be able to just look at the deduction process before adding the consistency check later on, and we preserve the attractive separation in the theory between the deduction and the consistency checking processes.

Figure 5.1 illustrates the operation of CORAL. The induction derivation, as described in §4.4, is pursued by the modified SPASS prover on the right of the diagram. Every non-redundant clause derived is passed to the refutation control client on the left, which launches a standard SPASS prover to do the check against the I-Axiomatisation. Crucially, these spawned SPASS s are not given the original axioms – only the I-Axioms are required (by Theorem 1, p. 52). Communication between the processes is via sockets.

If, at any time, a refutation is found by a spawned prover, the proof is written to a file and the completion process and all the other spawned SPASS processes are killed. If completion is reached by the induction prover, this is communicated to the refutation control program, which will then wait for the results from the spawned processes. If they all terminate with saturation, then there are no inconsistencies, and so the theorem has been proved (by Theorem 2, page 54).

One final issue was the passing on of answer literals. The I-Axiomatisation check may frequently make further instantiations of variables in the answer literals, and we would like the final answer presented to re¤ect these. To achieve this, we simply pass

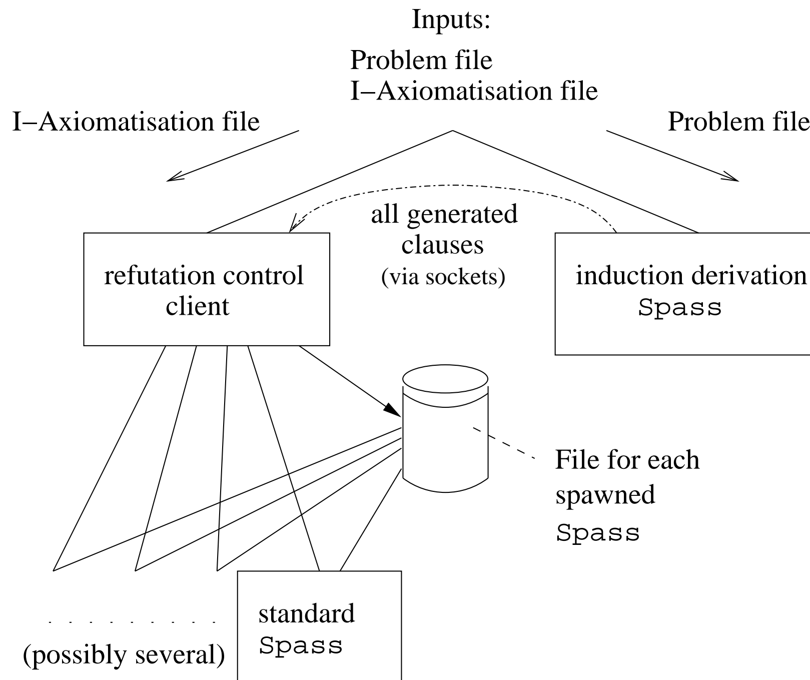

Figure 5.1: CORAL system operation

the answer literals on to the spawned SPASS processes in their partially instantiated state, and any further substitutions are accrued in the usual way, resulting in the correct final answer being presented. It may well be that different spawned processes will return different answers, but this is expected, and all the answers will be valid counterexamples. This is proved in Green's original paper on the use of answer literals, [Green, 1969].

5.3 Testing

Before proceeding to develop the formalism for the cryptographic security protocol verification problem, we tested the system. The separation of inductive completion and I-Axiomatisation checking means that almost all of the search for an inconsistency is done by the prover designed for inductive problems, and the spawned SPASS processes are just used to check for inconsistencies between the new clauses and the I-Axiomatisation. This should lead to a false conjecture being refuted after fewer in-

ference steps have been attempted than if the conjecture had been given to a standard first-order prover together with all the axioms and I-Axioms. Additionally, we have refutation completeness for our method, whereas a standard prover will not be able to deal with commutativity conjectures. We tested this on a number of small examples from the non-theorem refutation literature.

The results of this are in Table 5.1. The first three examples are from Protzen's work, [Protzen, 1992], and the next three from Reif et al.'s, [Reif et al., 2000]. As explained in §3.2.1, §3.2.2, these are two well-known techniques for refuting incorrect inductive conjectures. Their examples are taken from verification case studies in inductive theorem proving. The first example is the one given as a worked example in §3.2.1. The next two are slightly tougher examples from inductive theorem proving case studies. In particular, the third example shows how the splitting up of the inference process into the fair inductive derivation and the I-Axiomatisation checking allows us to tackle commutative type conjectures.

The fourth example is a lists conjecture from one of Reif's case studies. The fifth and sixth are from a case study in graph theory. In Reif's graph theory work, a graph is represented by a list containing entries $v(x)$, which indicates the graph contains a vertex x, and $e(x,y)$, which indicates the graph contains a directed edge from x to y. Referring to a vertex in an edge constructor implies it is also in the graph. So, the counterexample to the conjecture 'all graphs are acyclic' is a graph with one vertex, a, and a loop edge from a back to a. The counterexample to the next conjecture, 'all loopless graphs are acyclic', is a graph with two nodes, a and $s(a)$, with edges running from a to $s(a)$, and from $s(a)$ to a, producing the cycle. Reif remarked that that this one of the toughest conjectures he tested his system on, and it took several seconds to find the counterexample. CORAL takes just under one second on a Pentium III machine.

The final example involving the greatest common divisor function, *gcd*, is included because previous methods of proof by consistency could not refute this conjecture. This was because the *gcd* function could not be specified by a convergent rewrite system. Comon and Nieuwenhuis showed in theory how it could be tackled, [Comon and Nieuwenhuis, 2000], and here we confirm that our implementation of their method works.

In addition to the results in the table, we also present the comparison between the number of clauses derived by CORAL and the number derived by a standard prover in a graph, Figure 5.2. Here, the clauses required by a standard prover are plotted on the x-axis and the number of clauses required by CORAL on the y-axis. Both scales are logarithmic. We only have a small number of data points, but we can see that CORAL appears to save more time over the cruder approach as the examples get larger. This is encouraging. We discuss our results on these examples in §11.7.

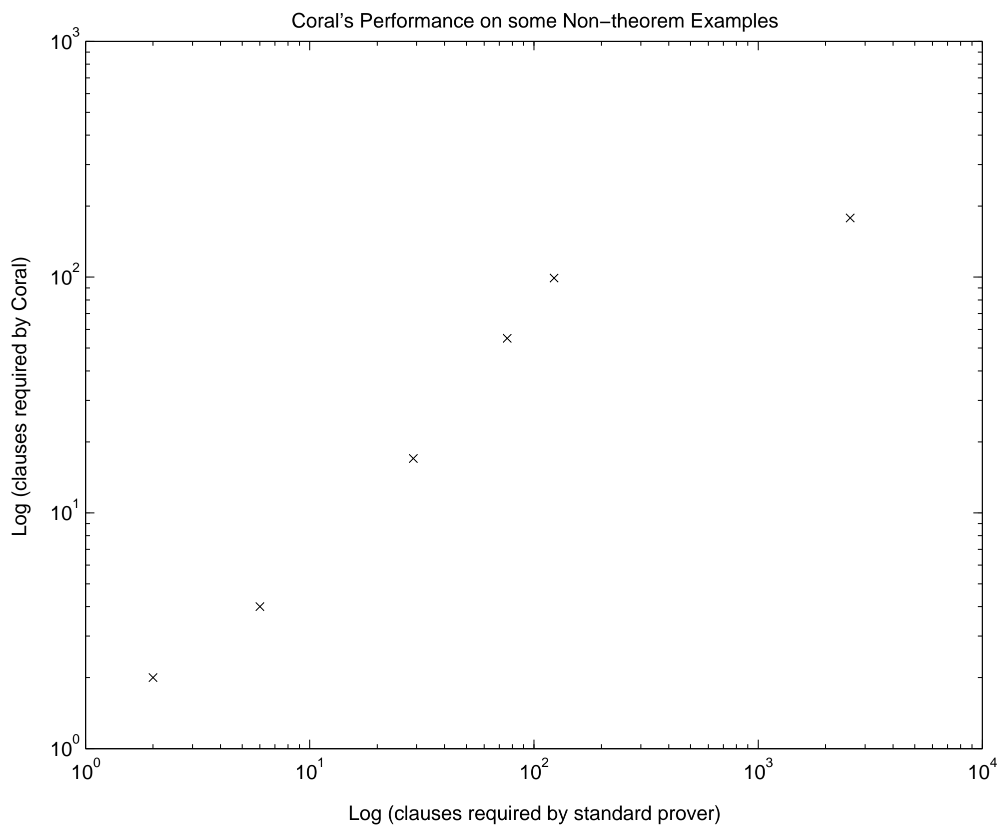

Figure 5.2: CORAL's run-times in terms of clauses derived compared with that for a standard prover

5.4 Summary

We have presented in this chapter our implementation of the Comon-Nieuwenhuis method for proof by consistency. It involved the adaptation of a state of the art first-order prover SPASS, to allow it to carry out the restricted inference steps to compute a fair induction derivation, and to allow it to carry out consistency checking. The key features are the separate indexes for axioms and lemmas, and the parallel I-Axiomatisation checker utilising sockets.

The results achieved on some examples from the non-theorem literature were enough to convince us that CORAL was working properly. Our work continued to the task of formalising the cryptographic security protocol problem.

Table 5.1: Sample of Results on Non-theorems from the Literature. *In the third column, the £rst number shows the number of clauses derived by the inductive completion prover, and the number in brackets indicates the number of clauses derived by the parallel checker to spot the inconsistency. The fourth column shows the number of clauses derived by an unmodi£ed £rst-order prover when given the conjecture, axioms and I-Axioms all together.*

Problem	Counterexample found	No. of clauses derived to find refutation	No. of clauses derived by a standard prover
$\forall N, M. \neg(s(N) + M = s(0))$	$N = 0, M = 0$	2(+0)	2
$X \neq Y \wedge X \neq 0 \wedge Y \neq 0$ $\Rightarrow (X \geq Y \wedge Y \neq 0) \vee$ $(X \neq 0 \wedge Y = 0)$	$X = s(0)$, $Y = s(s(X))$	4 (+3)	6
$app(K, L) = app(L, K)$	$K = [0], L = [s(X)]$	9(+11)	stuck in loop
$sort(l_1) \wedge l_2 = app(l_1, [head(l_3)])$ $\wedge length(l_3) \geq 2 * length(l_1)$ $\wedge l_3 \neq nil \wedge$ $member(head(l_1), tail(l_3))$ $\Rightarrow sort(l_2)$	$l_1 = [s(X)]$, $l_2 = [s(X), 0]$ $l_3 = [0, s(X)\|Y]$	55(+1)	76
All graphs are acyclic	$[e(a, a)]$	99	123
All loopless graphs are acyclic	$[e(s(a), a), e(a, s(a))]$	178	2577
$gcd(X, X) = 0$	$X = s(0)$	17(+2)	29

Chapter 6

Formalisation of the Protocol Veri£cation Problem

In this chapter we describe our formal model for cryptographic protocol analysis. It is based on Paulson's higher-order inductive model, [Paulson, 1998], which he formalised in Isabelle/HOL. Comon and Nieuwenhuis' 'Proof by Consistency' is a first-order technique, so we required a first-order model for the problem. Since we want to provide a complementary counterexample finding tool for the Isabelle/HOL approach, we tried to produce a model that is as close as possible to being a first-order version of Paulson's. We will show why it is possible to construct such a model in a satisfactory way, and then give details of our solution.

6.1 The Nature of the Paulson Model

A key feature of most cryptographic protocols is the typed nature of the information exchanged. Items sent in a message can be intended as keys, nonces, timestamps etc. Some protocols have been found to be susceptible to type attacks, where a principal accepts a nonce as a key for example (see §2.3.6). These kind of attacks assume that such type confusion is possible, e.g. that the same number of bits are used to specify a nonce and a key. Good engineering practice can easily prevent these kind of attacks, [Heather et al., 2000]. Paulson's model assumes that such an implementation

is used, i.e. that items of one type cannot be confused with items of another type. Consequently, he uses typed higher-order logic to formalise protocols. Each item has its type, and the type cannot change. However, no other higher-order features are used, such as quantification over sets. This means we can produce a first-order version of the model using some notion of first-order types or sorts.

6.2 Example - Needham-Schroeder Public Key

To explain how our model works, we will use the example of the Needham-Schroeder public key protocol (see §2.2). This protocol was one of the first proposed in the original paper on cryptographic protocols, [Needham and Schroeder, 1978]. The discovery of a ¤aw in the protocol 17 years later was a major event, [Lowe, 1995]. Since then, it has been used as the standard example for almost all tools for protocol analysis. Explaining our formalism for this protocol facilitates comparison with other approaches.

6.3 Free Constructors, Deciding Equivalence

At the base of our model is a free constructor theory. We have two basic sorts, numbers and agents.

Numbers: $0, s(0), s(s(0)), \ldots$
Agents: $spy, a, s(a), s(s(a)), \ldots$

Note that spy is an agent. For trusted third party protocols like Otway–Rees (see §2.3.5), we will have a key server as well, but this is not accepted as an agent. Instead the server is a special 0 arity function symbol.

A free constructor theory allows us to easily produce axioms to decide (in)equality of any pair of symbols. This allows us to restrict the overall theory to Horn clauses (i.e. clauses with at most one positive literal). This restriction is achieved by using an equational theory where everything is equated to two symbols, *true*, or *false* (this follows an example given by Comon and Nieuwenhuis, [Comon and Nieuwenhuis, 2000]). Suppose we would like to add a rule

$A \land \neg B \land C \rightarrow D$

(this matches the pattern of the rules for principals sending new messages in §6.5). In more usual disjunctive form, the clause would be written

$$\neg A \vee \neg C \vee B \vee D$$

We can see from the two positive literals, B and D, that it is not Horn. However, when we convert to the equational theory described above, the original rule looks like

$$A = true \wedge B = false \wedge C = true \rightarrow D = true$$

which is a Horn clause. We require suitable axioms for deciding the falsity of literals equated to *false*. For example, we have a set of axioms for deciding if two agents are the same person:

$eqagent(U,V)=false \rightarrow eqagent(s(U),s(V))=false$
$\rightarrow eqagent(spy,s(U))=false$
$\rightarrow eqagent(s(U),spy)=false$
$\rightarrow eqagent(s(U),a)=false$
$\rightarrow eqagent(a,s(U))=false$
$\rightarrow eqagent(a,spy)=false$
$\rightarrow eqagent(spy,a)=false$
$\rightarrow eqagent(U,U)=true$

Note that we only need the one clause to decide if the agents are the same person, since in a free constructor theory, two terms are equal only if they are syntactically identical.

We also have functions *agent* and *number* for checking that symbols belong to the sorts defined above. The slightly unusual construction for testing if a symbol is an agent is to prevent things like $s(spy), s(s(spy)), \ldots$ qualifying as agents.

$\rightarrow agent(spy)=true$
$\rightarrow agent(a)=true$
$\rightarrow agent(s(a))=true$
$agent(s(U))=true \rightarrow agent(s(s(U)))=true$
$\rightarrow number(0)=true$
$number(U)=true \rightarrow number(s(U))=true$

Protocols for key distribution often involve a secure server. In these cases, we represent this principal as a special symbol, *server*.

6.4 Principals, Keys and Nonces

There are a number of different types of object that may form part of a message sent in a protocol. However, to keep our formalism general and easily extensible, we don't want to have to define a set of symbols for each one and a set of axioms for deciding equivalence of their constructor symbols. So instead we base them all on numbers, and use a single arity function symbol as a type specifier. So we have message objects like this:

Nonces: $nonce(0), nonce(s(0)), \ldots$
Keys: $key(0), key(s(0)), \ldots$

Note that these keys are short term keys generated by a server for shared key protocols. Long term keys and public keys are based on the names of the principals:

Public keys: $pubk(a), pubk(s(a)), \ldots$
Long term keys: $longtermkey(a), longtermkey(s(a)), \ldots$

The names of agents, when they appear in the contents of a message, are represented in a similar way:

Agent identifiers: $principal(a), principal(s(a)), \ldots$

We usually want to put groups of symbols together to send in a message. We decided to do this using fixed sized tuple symbols ($pair, triple, quad$ etc.) rather than ¤exible length lists, as this will enable CORAL to tell the difference between two unequal length messages quickly without having to dig down the list. Some protocols have a ¤exible amount of data in certain messages, but we can model this by including a list inside the appropriate tuple symbol (for an example of this, see the modelling of message 3 of the Asokan–Ginzboorg protocol, Chapter 9).

A trace of messages exchanged by the principals is represented by a list, with the usual *cons* and *nil* symbols being employed. The list consists of elements of the form $sent(A,B,M)$, with *sent* a special arity 3 symbol. The first two arguments store the identities of the agent who sent the message and the intended recipient. The third holds the body of the message. The reason for using this special symbol (rather than

just *triple*) is to increase readability, but to also enable a shortcut in the definition of the *eq* symbol, since we know that the first two elements of the triple will be agent identifiers (see below).

As a final message element, we need to consider encrypted information. This is represented by an arity 2 symbol *encr*, with the first argument being the data and the second being the key. We can use the tuple symbols inside the first argument to encrypt a package of data. Note that we don't specify whether this is public or symmetric key encryption – this is handled for a particular protocol by the rules governing what each agent (and the spy) is able to do. As in most protocol models, we assume 'perfect cryptography', i.e. that the only way someone may decrypt $encr(X,Y)$ is by knowing the appropriate symmetric key Y or private key Y^{-1}, depending on the encryption scheme used.

These identifiers allow us to define an arity 2 function *eq* which can be used to decide equivalence of any two symbols in the theory. It is defined by the following axioms:

$\rightarrow$ *eq(U,U)=true*
eq(MSG1,MSG2)=false $\rightarrow$ *eq(sent(U,W,MSG1),sent(V,Y,MSG2))=false*
eqagent(A,B)=false $\rightarrow$ *eq(sent(A,W,X),sent(B,Y,Z))=false*
eqagent(A,B)=false $\rightarrow$ *eq(sent(W,A,X),sent(Y,B,Z))=false*
eq(H1,H2)=false $\rightarrow$ *eq(cons(H1,T1),cons(H2,T2))=false*
eq(T1,T2)=false $\rightarrow$ *eq(cons(H1,T1),cons(H2,T2))=false*
eq(MSG1,MSG2)=false $\rightarrow$ *eq(encr(MSG1,KEY1),encr(MSG2,KEY2))=false*
eq(KEY1,KEY2)=false $\rightarrow$ *eq(encr(MSG1,KEY1),encr(MSG2,KEY2))=false*
eq(M1,M2)=false $\rightarrow$ *eq(pair(M1,X),pair(M2,Y))=false*
eq(M1,M2)=false $\rightarrow$ *eq(pair(X,M1),pair(X,M2))=false*
eq(U,V)=false $\rightarrow$ *eq(nonce(U),nonce(V))=false*
eq(U,V)=false $\rightarrow$ *eq(s(U),s(V))=false*
$\rightarrow$ *eq(nonce(U),principal(V))=false*
$\rightarrow$ *eq(principal(U),nonce(V))=false.*
$\rightarrow$ *eq(a,s(U))=false*
$\rightarrow$ *eq(spy,s(U))=false*

$\rightarrow eq(0,s(U))=false$

$\rightarrow eq(s(U),a)=false$

$\rightarrow eq(s(U),spy)=false$

$\rightarrow eq(s(U),0)=false$

$\rightarrow eq(a,spy)=false$

$\rightarrow eq(spy,a)=false$

Note that the clauses for defining *eq* on terms with *sent* as their outermost function symbol pass the check on the first two arguments on to the special *eqagent* function. This speeds up the check and prevents nonsense values for agent identifiers being found when the check is performed on uninstantiated variables (as will often be the case). These would be detected later, but might waste a lot of proving time.

6.5 Protocol Messages

Our message trace model follows Paulson's very closely. We define a unary function *m* which is true just when its argument is a valid trace (with respect to the protocol being modelled). It is defined recursively. The empty trace *nil* is a valid trace, and a valid trace may be extended by a protocol message from an agent, or by a faked message from the spy. To decide whether an agent can extend a trace with a particular message, we need to be able to decide things about the messages that have already been sent. For this, we first need a standard member function, defined like this:

$$eq(H1,H2)=false \wedge member(H1,L)=false$$
$$\rightarrow member(H1,cons(H2,L))=false$$

$\rightarrow member(U,nil)=false$

$member(H,L)=true \rightarrow member(H,cons(W,L))=true$

$\rightarrow member(H,cons(H,T))=true$

$member(U,nil)=true \rightarrow$

This allows an agent to decide, for example, only to send a message 2 when a valid message 1 has been received. We also need agents to be able to generate fresh nonces. We assume that fresh nonces are 'perfectly fresh', i.e. no freshly generated nonce

ever coincides with a previously used one. In a practical implementation of a protocol, agents would probably just choose random numbers from a suitably enormous range so as to make coincidentally equal nonces almost impossible. To get a perfectly fresh nonce, we have to define a '*parts*' operator, similar to the one used by Paulson, [Paulson, 1998, p. 12]. If H is a list of messages, then $parts(H)$ is the least set containing the messages in H closed under projection and decryption. We are only concerned with determining when something is not in $parts(H)$. So we define the operator and a set membership function *in* like this:

$$in(U,parts(V)) = false \land in(U,parts(W)) = false \\ \rightarrow in(U,parts(cons(sent(X,Y,W),V))) = false$$

$$in(U,parts(V)) = false \land in(U,parts(W)) = false \\ \rightarrow in(U,parts(encr(V,W))) = false$$

$$in(U,parts(V)) = false \land in(U,parts(W)) = false \\ \rightarrow in(U,parts(pair(V,W))) = false$$

$$eq(U,V)=false \rightarrow in(nonce(U),parts(nonce(V)))=false$$

$$\rightarrow in(nonce(U),parts(principal(V)))=false$$

$$\rightarrow in(nonce(U),parts(key(V)))=false$$

$$\rightarrow in(U,parts(nil))=false$$

Having defined *member* and *parts*, we can then model the agents participating in a protocol. For an explanation of this, we return to our example: the Needham-Schroeder public key protocol. The protocol, explained in §2.2, looks like this:

1. $A \rightarrow B : \{\!| N_A, A |\!\}_{pubK_B}$
2. $B \rightarrow A : \{\!| N_A, N_B |\!\}_{pubK_A}$
3. $A \rightarrow B : \{\!| N_B |\!\}_{pubK_B}$

These three messages are modelled in our formalism like this:

$$agent(A)=true \land agent(B)=true \land number(NA)=true \land m(Trace)=true \land \\ in(nonce(NA),parts(Trace))=false \\ \rightarrow m(cons(sent(A,B,encr(pair(nonce(N),principal(A)),pubk(B))),Trace))=true$$

$$number(NB)=true \land m(Trace)=true \land in(nonce(NB),parts(Trace))=false \\ member(sent(X,B,encr(pair(nonce(NA),principal(A)),pubk(B))),Trace)=true \\ \rightarrow m(cons(sent(B,A,encr(pair(nonce(NA),nonce(NB)),pubk(A))),Trace))=true$$

m(Trace)=true$\wedge$
member(sent(X,A,encr(pair(nonce(NA),nonce(NB)),pubk(A))),Trace)=true$\wedge$
member(sent(A,B,encr(pair(nonce(NA),principal(A)),pubk(B))),Trace)=true
$\rightarrow$ *m(cons(sent(A,B,encr(nonce(NB),pubk(B))),Trace))=true*.

The first clause specifies that any agent can add a message 1 to the trace provided the types for the nonce number and the agent identifiers are correct, and provided that the nonce is fresh. Recall that we use the $pubk(A)$ symbol to mean the public key belonging to A. This makes it easy to decide who can decrypt the package (only agent A). The second clause says that any trace with a message one in it may be extended with a message 2 from the recipient of the message 1. Note that the agent responding to message 1 cannot know for certain who the message 1 was from. We model this with the new variable X in the member literal as the first argument of the *sent* function. Again we require the nonce to be fresh.

The third clause specifies that a trace containing a message 1 from agent A, and a message 2 apparently in response to that message, may be extended by a message 3 from A. The reader may have noticed that we allow principals to respond any number of times to a message – this is the same as in Paulson's formalism.

Finally, we need a clause specifying that the empty list is a valid trace, again following Paulson's model:

$\rightarrow$ *m(nil)=true*

A difference between our model and Paulson's model is that the Paulson model explicitly specifies that an agent should not send a message to himself. We could quite easily add this specification to our model, using the *eqagent* symbol, but we omit it. Instead we specify in our conjectures that if we are interested in some property for party A in a run with party B, then they must be distinct agents. In practice, this speeds up the discovery of counterexamples, since there are slightly fewer literals to consider.

6.6 Modelling Intruder Knowledge

To define what messages an intruder or spy may send, we need to define two more operators. These are almost identical to those proposed by Paulson, [Paulson, 1998,

p. 12]. The first is $analz(H)$, defined to be the least set containing H closed under projection and decryption by known keys. The second is *synth*, which is the least set including agent names closed under pairing and encryption by known keys. Our definition differs slightly from Paulson's: since we will only ever be concerned with the contents of $synth(analz(X))$ rather than just $synth(X)$, we combine the definitions together. The final content of $synth(analz(X))$ is identical to that in Paulson's model though. Here are the relevant clauses:

$member(sent(U,V,W),X)=true \rightarrow in(W,analz(X))=true$

$in(encr(U,pubk(spy)),analz(V))=true \rightarrow in(U,analz(V))=true$

$agent(U)=true \wedge in(V,synth(analz(W)))=true$
$\quad \rightarrow in(encr(V,pubk(U)),synth(analz(W)))=true$

$in(pair(U,V),analz(W))=true \rightarrow in(U,analz(W))=true$

$in(pair(U,V),analz(W))=true \rightarrow in(V,analz(W))=true$

$in(U,synth(V))=true \wedge in(W,synth(V))=true$
$\quad \rightarrow in(pair(U,W),synth(V))=true$

$in(U,analz(V))=true \rightarrow in(U,synth(analz(V)))=true$

$agent(U)=true \rightarrow in(principal(U),analz(V))=true$

$\rightarrow in(U,synth(analz(nil)))=false$

$\rightarrow in(U,analz(nil))=false$

Finally, we have a rule allowing the spy to add to a trace any message that he knows how to make.

$m(Trace) = true \wedge eq(spy, AGENT) = false \wedge$
$in(MSG, synth(analz(Trace))) = true \wedge$
$\quad \rightarrow m(cons(sent(spy,AGENT,MSG),Trace))=true \; (\dagger)$

6.7 Term Ordering

Superposition theorem proving requires a term ordering, as described in §4.2. For our protocol formalism, we use a recursive path ordering (RPO), as defined in §4.1, with one minor modification. In order for the *analz* operator to work as desired, we need the ordering to work the opposite way when comparing two $in(X, analz(Y))$ terms.

Normally under the recursive path ordering, if one term has a function symbol as its top functor, and the other term has a variable which is an argument of the first term's function symbol, then the first term is considered largest. An example makes this clearer. We have rules which look like:

in(pair(U,V),analz(W))=true→in(U,analz(W))=true

The idea of this rule is that it allows the spy to decompose pairs of terms into component parts. Under the RPO, the left hand side will be considered larger, since it contains the variable *U* inside the function *pair*. However, we want the right hand side to be maximal, as otherwise we won't be able to use it for inferences by conjecture superposition on negative literals (informally, we expect to encounter conjecture clauses with the interpretation 'the spy can send a particular fake message if he can get term *T*', which corresponds to a clause with a negative *in(T,analz(Trace))* literal). In order to make the rule applicable to these clauses, we reverse the ordering for these kinds of rules. This modification to the ordering requires a very simple addition to CORAL's implementation. When the `SPOrd` ¤ag is set, the result of a comparison between two $in(x, analz(Trace))$ literals is reversed.

Note that our new ordering is in general now not well-founded. To see this, consider a sequence

$$in(U, analz(Y)) \succ in(pair(U,V), analz(Y)) \succ in(pair(pair(U,V),W))\ldots$$

However, we prevent a sequence of terms like this from ever arising in our theory by using ¤at constructors for messages of length 2,3 etc. as outlined in §6.4 above. A heuristic reduction rule (described in §7.1) prevents terms with nested constructors from arising. Note that our list constructor *cons*, which of course does have to occur nested in order to make a list, does not arise inside the X in $in(X, analz(y))$ terms. This is because the *cons* part of the trace is stripped away by the rule

member(sent(U,V,W),X)=true → in(W,analz(X))=true

giving us just the message contents. Messages do not contain lists in fixed 2 and 3 party protocols. In the case of group protocols however, like the one studied in Chapter 9, we do indeed have lists inside messages. We discuss this problem in §9.2.2.

In general, choosing the precedence for an ordering to be used for superposition theorem proving is very much a matter of trial and error. One obvious restriction is that defined symbols should be considered larger than the things they are defined in terms of. With this in mind, we chose the ordering:

m≻in≻≻agent≻number≻member≻eq≻nonce≻synth≻analz≻cons
≻encr≻s≻key≻pair≻sent≻nil≻0≻a≻spy≻false≻true

6.8 Refutation Completeness

As we saw in §4.5 (page 57), Comon and Nieuwenhuis recover refutation completeness in the case of a non-saturated theory by means of *reductive de£nitions* (Definition 12) and *de£nition patterns* (Definition 13). They use these two concepts in Lemma 2:

Lemma 2 *Let E be a reductive de£nition and let c be a conjecture such that $c|_p$, with p the position of the innermost de£ned symbol, is a de£nition pattern.*

Then for every ground instance $c\sigma$ where σ is normal, there exists some inference by conjecture superposition at position p with a conclusion c', and a normal substitution σ' such that $I \not\models c\sigma$ implies $I \not\models c'\sigma'$, and furthermore, $c\sigma \succ c'\sigma'$.

This lemma is then used to complete the proof of refutation completeness for the Comon-Nieuwenhuis method for reductive definitions. However, although our formalism is reductive, we do not in general have definition patterns. Comon and Nieuwenhuis give a method for producing definition patterns in general formulae, but this adds to the complexity of the inference process and is something we would like to avoid. Fortunately, we can prove a version of Lemma 2 specific to our formalism, but with the same conclusion. This is then sufficient to complete the proof of refutation completeness (see [Comon and Nieuwenhuis, 2000, p. 21]). Our form of the lemma is:

Lemma 3 *Suppose E is the formalism for security protocols given above. Then for every non-normal ground instance $c\sigma$ where σ is normal, there exists some inference by conjecture superposition with a conclusion c', and a normal substitution σ' such that $I \not\models c\sigma$ implies $I \not\models c'\sigma'$, and furthermore, $c\sigma \succ c'\sigma'$.*

Proof: First note that in our theory, all literals take the form of $f(t_1,\ldots,t_n) = \{true/false\}$, with f a defined symbol from $m, in, eq, agent, number, member$, and no further defined symbols occurring in the t_i. This means that all superposition inferences happen at the first position. Recall also that our formalism is a free constructor theory, so equality in the formalism corresponds to syntactic identity.

To simplify the proof, we restrict the conjectures we will pose to Prolog-style queries, i.e. with only negative literals. One literal specifies that we have some valid trace *Trace*, and the other literals specifying properties of the trace using *member*, *eq* and $in(.,analz(.))$, or specifying types using *number* or *agent*. This is sufficient to cover secrecy, authenticity, resistance to disruption, and other properties. All the conjectures used to rediscover known attacks in Chapter 8, and also to carry out the case studies described in Chapters 9 and 10, fit into this category.

Now, take a ground instance of a conjecture $c\sigma$, with σ a normal substitution. We will now show that if $I \not\models c\sigma$, then a literal in the conjecture clause is reducible, resulting in a smaller clause c'. Since $c\sigma$ is non-normal, there must exist a literal $l = f(t_1,\ldots,t_n) = \{true/false\}$. Now we treat the case where f is each defined symbol in turn:

$m(t_1) = true$: if t_1 is *nil*, then l is immediately reducible to the tautology $true = true$. Otherwise, t_1 is equal to some other ground term. Since $I \not\models c\sigma$, then a counterexample exists to our security conjecture, in which case t_1 must be a sublist of a valid trace. Then t_1 is of the form *cons(sent(.,.,),..)*. For a valid trace, this is always reducible by rule † (above, §6.6, page 81) if the first message was sent by the spy, or one of the rules in §6.5 if it was sent by an honest agent. Note we will not pose conjectures of the form *m(X)=false*, since this is rather pointless.

$member(t_1,t_2) = \{true/false\}$: From our definition in §6.5, l is reducible for any ground term provided t_2 is a list. But since t_2 always will be the trace, and in the case of $I \not\models c\sigma$, as above, t_2 must be a sublist of a valid trace, so it must be a valid list and hence l is reducible.

$eq(t_1,t_2) = \{true/false\}$: In §6.4 we defined clauses specifying *eq* over all ground terms. One of these will always apply to l in this case reducing it to $\{true/false\} = \{true/false\}$ as appropriate.

$in(t_1, analz(t_2)) = true$: Always reducible by the rules in §6.6.
$in(t_1, synth(analz((t_2))) = true$: Always reducible to $in(t_1, analz(t_2))$, by the rule in §6.6.
$in(t_1, parts(t_2)) = false$: The rules in §6.5 reduce l for any ground term t_2 provided t_1 is a nonce. But the only occurrences of *parts* in our theory refer to nonces, so l is indeed reducible.
$number(t_1) = true$: Since $I \not\models c\sigma$, t_1 is a valid number, so l is reducible by the rules in §6.3.
$agent(t_1) = true$: As for *number* above.
□

6.9 I-Axiomatisation

As we have a free constructor theory, and a completely defined equality function *eq*, the job of constructing an I-Axiomatisation is an easy one. We can simply use Musser's technique (see §4.6). Hence the only equation we need in our I-Axiomatisation is

$$true \neq false$$

I-Axiomatisations constructed by Musser's method are not necessarily normal. However, ours is normal:

Theorem 4 *The I-Axiomatisation as de£ned above is normal for our theory.*

Proof: Suppose $s \succ t$ and s is minimal w.r.t. its congruence class. The *eq* function in our theory reduces any two ground terms arising in the theory to *true* or *false*. Hence if $s \succ t$, then $s = t \equiv false = true$, hence $\mathcal{A} \models s \neq t$. □

6.10 Comparison between Paulson's Formalism and CORAL's Formalism

Paulson's inductive model for security protocol analysis is well established, and accepted as being a suitable model for the problem. Our model retains the key features

of Paulson's model that give it its expressivity: we model an unbounded number of agents, nonces, and messages in the trace. The formulae for modelling the honest agents behaviour are almost identical. The formalism of the *parts*, *synth* and *analz* operators is the same modulo the fact that we combine their definitions with the definition of the *in* set membership operator. However, we cannot prove an equivalence in the sense that any protocol and security property that can be formalised in Paulson's model can also be formalised in ours. The problem is that Paulson's model is in higher-order logic, allowing quantifications to be made that we cannot make. Perhaps someone might propose a protocol that establishes properties of some arbitrary set of different types of object, and we will want to prove properties that quantify over all possible types.

More differences between our formalism and Paulson's are introduced in the next chapter, where we introduce optimisations to make the search for refutations more efficient. These optimisation are based principally on the results of Syverson et al., [Syverson et al., 2000], where it is shown that it is not necessary to model an arbitrary number of bad agents, that is agents with compromised keys, nor is it necessary to model a spy who may send any term. It is sufficient to model a spy who has a single long term key (or a single malicious insider) who only sends messages which look like messages from the protocol, i.e. will be accepted as valid protocol messages by some hones agent. We show how we incorporate these results into our formalism in §7.2.1. The salient point is that thanks to the results of Syverson et al., we have not reduced the expressivity of our model in terms of finding protocol attacks.

Key advantages of the Paulson model are its naturalness and simplicity, which we believe we have retained. In particular, these features have allowed the Paulson model to be easily adapted to new non-standard protocols and security properties, for example where the timeliness of messages must be modelled, [Bella and Paulson, 1997], or an arbitrary number of principals may be involved, [Paulson, 1997], or properties such as non-repudiation must be considered, [Bella and Paulson, 2001]. We believe that our model has similar properties, as is demonstrated by its adaptation to two very different kinds of group protocol in Chapter 9 and Chapter 10. We will return to this point in our evaluation in §11.5.

6.11 Formulating Conjectures in the Formalism

The conjectures we have disproved in CORAL are very similar to those Paulson attempts to prove. For example, here is the 'possibility property' for the Needham-Schroeder public key protocol. A possibility property basically states that there are valid traces that reach the end of the protocol. In our formalism, we effectively refute an impossibility conjecture, i.e. that there are no valid traces that reach the end. The counterexample is a valid trace. With the conjecture

% A and B are distinct honest agents
eqagent(A,B)=false $\wedge$
eqagent(A,spy)=false $\wedge$ *eqagent(B,spy)=false*$\wedge$
% Trace is a valid Needham-Schroeder trace
m(Trace)=true$\wedge$
% Trace contains a message 3
member(sent(A,B,encr(pair(nonce(NA),principal(A)),pubk(B))),Trace)=true
$\rightarrow$

CORAL gives the counterexample

```
answer(cons(sent(s(a),a,encr(nonce(0),pubk(a))),
       cons(sent(a,s(a),encr(pair(nonce(s(0)),nonce(0)),pubk(s(a)))),
       cons(sent(s(a),a,encr(pair(nonce(s(0)),principal(s(a))),pubk(a)))
      ,nil)))),tru)
```

which is just a straight run of the protocol (note that message 3 appears first and message 1 last, because of the way the lists are built in CORAL). We wrote a simple pretty-printing script in Perl reverse the message order and give the output in a something close to standard notation. for example, for the counterexample above, it produces the output:

```
s(a) --> a        :  { N(1) , s(a) }_PK(a)

   a --> s(a)     :  { N(1) , N(0) }_PK(s(a))

 s(a) --> a        :  { N(0) }_PK(a)
```

which corresponds to a straight run of the protocol:

1. $s(a) \rightarrow a : \{\!| N_1, s(a) |\!\}_{pubK_a}$

2. $a \rightarrow s(a) : \{\!| N_1, N_0 |\!\}_{pubK_{s(a)}}$

3. $s(a) \rightarrow a : \{\!| N_0 |\!\}_{pubK_a}$

Attacks on protocols are found by refuting security properties. In Chapter 8, we show how we posed conjectures to find known attacks on four security protocols. In Chapter 9, we show how we posed conjectures which allowed CORAL to discover three new attacks on a protocol. However, before this could be done, we had to add some heuristics to CORAL to make the search problem tractable. These are described in Chapter 7.

6.12 Summary

In this chapter we have shown how we formulated a first-order version of Paulson's inductive model, giving the formalisation of the Needham-Schroeder public Key protocol as a worked example. Our formalism consists only of Horn clauses, and uses simple first-order types. We have shown why the Comon-Nieuwenhuis method is refutation complete with respect to our formalism. We have also demonstrated the kind of output CORAL gives when a counterexample is found, and how our pretty printer renders it easy to understand. To refute security conjectures in a reasonable amount of time, we had to add some domain specific reduction rules to the basic CORAL system outlined in Chapter 5, and make some additions to the basic formalism. These heuristics are described in the next chapter. In Chapter 8, we show how some known attacks were found using CORAL and this formalism. In Chapters 9 and 10, we show how we used the formalism to model group protocols, and find five new attacks.

Chapter 7

Optimisations

To make CORAL effective in finding attacks on faulty protocols, we had to make some optimisations, both to the formalism described in the previous chapter and to the implementation of CORAL itself. We describe those optimisations and the reasoning behind them here. The development of these heuristics was carried out during the rediscovery of the attacks on the Needham-Schroeder, Otway-Rees, Neuman-Stubblebine and Clark-Jacob protocols described in the next chapter. However, no further heuristics were needed to carry out the case study described in Chapter 9, and for the case study in Chapter 10, we simply had to re-implement the heuristic described in §7.3 to suit the adapted formalism.

Throughout this chapter, it will help to bear in mind that the nature of CORAL is to construct traces backwards, from the final message towards the first.

7.1 Elimination of Invalid Terms

A key heuristic was to eliminate clauses containing malformed terms, i.e. terms containing nested *pair*, *triple* or *quad* type constructors. The exception is when the nested constructor occurs inside an *encr* constructor – the correct way to represent a message like

$$N_A, \{\!|\, N_B, B \,|\!\}_{K_B}$$

in our formalism is

pair(nonce(NA),encr(pair(nonce(NB),principal(B)),key(B)))

CORAL contains a function that checks a term for nested *pair* type constructors taking into account the *encr* constructor. Clauses containing malformed terms are discarded as redundant.

7.2 Restricting the Spy's Messages

One large source of combinatorial blow-up is the specification of the spy. He can send anything he can build out of the preceding traffic in the network. This gives him the potential to send a lot of garbage that has no chance of fooling the honest principals. We use several heuristics to combat this problem. Heuristics of this type are used in many protocol analysis tools, and are sometimes referred to as *step compression*, since they combine steps taken by the spy to intercept and send messages with the steps taken by honest agents to send messages in the protocol.

7.2.1 Spy Only Sends Protocol Messages

The spy has nothing to gain from sending messages which honest players will not recognise as part of the protocol (this is shown formally in [Syverson et al., 2000]). Therefore, we can cut down the search space by allowing the spy only to send messages which fit the protocol pattern. For example, in our formalism for the Needham-Schroeder public key protocol (see §2.2), we turn the single rule:

m(XT)=true $\wedge$ *eq(spy,AGENT)=false* $\wedge$
in(MSG,synth(analz(XT)))=true
$\rightarrow$ *m(cons(sent(spy,AGENT,MSG),XT))=true*

into the set of three rules (one for each protocol message):

% spy faking a message 1
m(Trace)=true $\wedge$
in(encr(pair(nonce(NA),principal(A)),pubk(B)),synth(analz(Trace)))=true
$\wedge$ *eq(spy,B)=false* $\rightarrow$
m(cons(sent(spy,B,encr(pair(nonce(NA),principal(A)),pubk(B))),Trace))=true

% spy faking a message 2
m(Trace)=true∧
in(encr(pair(nonce(NA),nonce(NB)),pubk(A)),synth(analz(Trace)))=true
∧ eq(spy,A)=false →
m(cons(sent(spy,A,encr(pair(nonce(NA),nonce(NB)),pubk(A))),Trace))=true

% faked message 3
m(Trace)=true∧
in(encr(nonce(NB),pubk(B)),synth(analz(Trace)))=true
∧ eq(spy,B)=false →
m(cons(sent(spy,B,encr(nonce(NB),pubk(B))),Trace))=true

7.2.2 Spy Only Expects Protocol Messages

A similar idea is to adapt the rules which model the spy learning new information to the protocol being considered. CORAL builds traces in reverse, so what we are aiming to do is remove clauses that involve the spy sending a message containing terms he cannot possibly obtain from previous messages in the trace. For example, we don't want the spy to send messages based on the assumption that an honest agent will have sent his long term key in clear text. To this end, in the case of the Needham-Schroeder public key protocol, we exchange the single rule:

member(sent(A,B,MSG),Trace)=true→in(MSG,analz(Trace))=true

for the set of rules:

member(sent(X,Y,encr(pair(nonce(NA),principal(A)),pubk(B))),Trace)=true
→in(encr(pair(nonce(NA),principal(A)),pubk(B)),Trace)=true

member(sent(X,Y,encr(pair(nonce(NA),nonce(NB)),pubk(A))),Trace)=true
→in(encr(pair(nonce(NA),nonce(NB)),pubk(B)),Trace)=true

member(sent(X,Y,encr(nonce(NB),pubk(B))),Trace)=true
→in(encr(nonce(NB),pubk(B)),Trace)=true

Now the spy can only expect to gain information from messages that conform to the protocol. To save the user some effort, we wrote a short Perl script to take the messages

required to specify a protocol and produce both the rules required for eavesdropping on messages like the ones above, and the rules required for the spy faking messages like the ones in §7.2.1.

7.2.3 Spy Only Expects Subterms from Protocol Messages

As an enhancement to the above heuristic, we prevent the spy from expecting not just invalid messages, but also terms which are not a subterm of a valid protocol message. We can eliminate such terms very quickly by taking advantage of the term indexing mechanism used in SPASS and similar modern provers (see §4.2). In order to speed up the search for candidate literals for the next inference step, the prover indexes terms by storing only one copy of every subterm, along with pointers to all of its superterms, in the worked off clause index. In CORAL, where we have a separation between the axioms and the worked off clauses in order to implement the proof by consistency strategy (see §5.1.2), we also have such an index for the axiom clauses. The result of this is that, for any given subterm, we can quickly extract a list of all literals in the axiom set that contain candidate unifiers for that subterm[1]. We can then quickly check to see if any of these are a succedent $m()$ literal. These only occur in the description of the protocol for honest users (see §6.5), and in the optimised version of the rules for the spy's faked messages (above, §7.2.1). So, a subterm found in such a situation must be a subterm of a valid protocol message.

A clause containing a literal of the form $in(X, analz(Trace))$ indicates that the spy has sent a message containing the subterm X, and expects to find it in an earlier message in the trace. If this subterm X, fails the test described above, i.e. it does not occur in any succedent $m()$ literal, then such a subterm does not exist in any valid message, so the clause is pruned away as redundant.

7.2.4 No Two Spy Messages in a Row

In a normal two or three party protocol, where no honest party sends two messages in a row, it can be seen that the spy gains nothing from sending two messages in a row. If

[1] The statistics in Table 8.2, on page 106, give an indication of how fast these operations are compared to those required for subsumption checking and rewriting

there is an attack on such a protocol involving the spy sending two messages in a row, then there must also be one where he sends the first, waits for a response, and then sends the second. We don't want to have both of these in our search space, as such duplication wastes search time.[2]

In CORAL, this simplification is implemented as a redundancy rule, turned on by the `SPRed` command line ¤ag (this also turns on the redundancy rule described in §7.2.3). Clauses containing an answer literal which represents a trace with two consecutive messages from the spy are marked as redundant, and hence pruned out.

7.3 Eager Elimination of Unsatis£able *parts* Literals

As explained in §6.5, we follow Paulson in using a parts operator to model freshness. For a trace T, the set $parts(T)$ contains all the keys, nonces and other message elements that appear in T. A nonce N is only fresh with respect to T if $in(nonce(N), parts(T)) = false$. This suggests another pruning heuristic. If a clause contains an antecedent literal $in(nonce(N), parts(T)) = false$, and the term $nonce(N)$ occurs in term T, then the clause cannot possibly be reduced to the empty clause, so it is pruned from the search space. These literals would eventually be removed anyway by the unpacking of the definition of *parts*, but by eagerly pruning them away, we save time.

7.4 Literal Selection

Standard first-order theorem proving using an ordered superposition rule usually employs literal selection, as explained in §4.2. The intention is to exert a certain amount of control over the theorem proving strategy, which will eliminate some duplication of effort, i.e. in our case prevent different sequences of inferences that lead to the same trace being examined. The standard literal selection function in SPASS chooses the literal of maximal weight. However, after some experimentation, we found that a slightly customised selection strategy resulted in a significant performance increase. With the

[2]Note however that this is not necessarily useful for group protocols, where the spy may have to imitate an agent sending out messages to several different members of the group. There is an example of this is the Asokan–Ginzboorg protocol, examined in the next chapter.

command line ¤ag `SPSel` set, the strategy first chooses $in(x, analz(trace))$ literals, and if none of these can be found it chooses $m()$ literals. If the clause contains neither of these then the standard selection strategy is used.

We look for $in()$ literals first because we don't want to proceed further in examining a clause with a spy's message in it without determining what he needs to see earlier in the trace. The second priority of the $m()$ literals is for the same reason. Processing each of these will generate further $member()$ literals, which will in term lead to further $in()$ and $m()$ literals.

7.5 Summary

In this chapter we have described the 5 heuristics developed to allow CORAL to rediscover security protocol attacks. These included new reduction rules, and some modifications to the way we formalise the protocols. These modifications are carried out by an automatic pre-processor. In the next chapter, we show how CORAL was used to rediscover 10 known attacks on security protocols, and evaluate the part these heuristics played in this. In the subsequent chapters, we see that no further heuristics were required to complete two successful case studies on new protocols, yielding 5 new attacks.

Chapter 8

Rediscovering Known Attacks

In this chapter, we explain how CORAL was used to rediscover known attacks on some standard security protocols. The protocols in this chapter are 'standard' in the sense that they were all proposed in the academic literature, and involve just two parties who wish to communicate with each other and, in some cases, a secure key server. We follow standard AI methodology in that the heuristics described in the previous chapter were refined on a small 'development set' of four protocols described in the next section. We show how security properties were conjectured for these examples, and the counterexamples CORAL found. We give timing information and evaluate the effectiveness of the heuristics we developed, as described in the previous chapter. For the Needham-Schroeder public key protocol, we also give a close analysis of the search pattern. After completing the development, we applied CORAL to a 'test set' of ten protocols from a standard corpus, [Clark and Jacob, 1997]. The ten protocols were chosen to reflect the five different kinds of attacks in the corpus. We give timing results for the test set, and then evaluate the performance of CORAL on these examples.

8.1 The Development Set of Protocols

The four protocols in the development set were chosen to reflect different kinds of protocol and different kinds of attacks. Three involve shared key encryption, and one a public key system. They were also chosen to vary from the simplest to the most

complex 'standard' protocol, in the hope that this would allow for smooth development of heuristics. We present here first the results on the Needham-Schroeder public key protocol, since this was used as example to explain the formalism in Chapter 6. The next three attacks are presented in approximate order of complexity: first a simple parallel session attack on a didactic Clark-Jacob protocol, then an attack on the Neuman-Stubblebine protocol, and finally a quite complex attack on the BAN Otway-Rees protocol.

8.2 Attacking Needham-Schroeder Public Key

We used this protocol to demonstrate our formalism in Chapter 6. To discover the well known attack on the NSPK protocol, we refute a conjecture which would be B's guarantee of the authenticity of A. Our conjectures are all in terms of a trace, and are very similar to those used by Paulson in his work, [Paulson, 1998]. In this case, the conjecture states that if B receives a message 3 apparently from A, and has sent out a message 2 to A, then there is no valid trace in which that message 3 didn't really come from A. We formulate the conjecture like this:

% B has sent a message 2
member(sent(B,A,encr(pair(nonce(NA),nonce(NB)),pubk(A))),Trace)=true$\wedge$

% message 3 has been received
member(sent(X,B,encr(nonce(NB),pubk(B))),Trace)=true$\wedge$

% but A hasn't sent it to B
member(sent(A,B,encr(nonce(NB),pubk(B))),Trace)=false
$\rightarrow$

The counterexample is found in 38 seconds, and having been passed through CORAL's pretty printer, looks like this:

```
    a --> spy      :  { N(1) , a }_PK(spy)


  spy --> s(a)     :  { N(1) , a }_PK(s(a))


 s(a) --> a        :  { N(1) , N(0) }_PK(a)
```

```
spy --> a        :  { N(1) , N(0) }_PK(a)

  a --> spy      :  { N(0) }_PK(spy)

spy --> s(a)     :  { N(0) }_PK(s(a))
```

The counterexample CORAL has found is the well known attack discovered by Lowe:

$$
\begin{array}{lcclcl}
1. & A & \rightarrow & C & : & \{\!|N_A, A|\!\}_{pubK_C} \\
1'. & C_A & \rightarrow & B & : & \{\!|N_A, A|\!\}_{pubK_B} \\
2' & B & \rightarrow & A & : & \{\!|N_A, N_B|\!\}_{pubK_A} \\
2 & C_B & \rightarrow & A & : & \{\!|N_A, N_B|\!\}_{pubK_A} \\
3. & A & \rightarrow & C & : & \{\!|N_B|\!\}_{pubK_C} \\
3.' & C_A & \rightarrow & B & : & \{\!|N_B|\!\}_{pubK_B}
\end{array}
$$

8.2.1 How CORAL Finds the NSPK Attack

We wrote a short script to analyse the output from CORAL's rediscovery of the NSPK attack. We used this to look at the path through the search space that lead to the attack, the point in time at which each node on this path was considered, and what CORAL was doing in between considering those nodes. The raw data for the attack path is in the form of pairs consisting of a clause number and a number showing when this was considered as the given clause. This is shown in Table 8.2.1.

Broadly speaking, there were three significant gaps in the search, between clauses 151 and 154 being considered, between clauses 154 and 520, and between clauses 902 and 912. Inspecting the output with our pretty printing tool gives some insight as to what is going on:

Clause 151 represents a trace ending with the spy sending message 3. Between clauses 151 and 154, CORAL considers other traces where the final message is sent by an honest agent. These traces only involve honest agents and/or the spy behaving as an honest agent, and so contain no literals requiring the spy to break down previous traffic and replay messages. This makes them smaller and so more attractive to CORAL's

Clause	Step	Clause	Step	Clause	Step
93	1	916	350	1346	366
97	3	1295	351	1350	367
104	5	1298	352	1352	370
109	7	1301	353	1364	371
119	8	1303	356	1366	372
122	13	1314	357	1370	373
143	15	1318	358	1371	375
151	16	1320	360	1377	376
154	122	1326	361	1384	377
520	231	1333	363	1395	380
898	232	1385	378	1435	381
902	235	1340	365	1438	382

Table 8.1: The pattern of search for the NSPK attack.

standard lightest-first heuristic. Once these possibilities have been exhausted, the spy comes back to clause 154, originally derived from clause 151.

Clause 154 represents the fact that the spy has replayed message 3 rather than sending it as an honest participant in the protocol. Between clause 154 and clause 520 being considered, CORAL considers traces where the spy is involved in an honest session with the agent who sends a message 3 for him to replay. Having exhausted all these possible combinations, CORAL considers clause 520, where someone else has sent message 2 to the agent whose message 3 he will eventually replay.

CORAL runs quickly through to clause 902. From here, it considers various ways the sender of message 2 could be involved, until coming to clause 916, where the spy sees a message 2 in the trace and replays it. From here, once clause 916 in considered, CORAL runs straight through to the finish.

An interesting feature of CORAL highlighted by this analysis is the way that subsumption checking prunes away redundant states in the search space. For example, CORAL derives a clause corresponding to the trace fragment:

$$\begin{array}{rcccl} spy & \rightarrow & Y & : & \{\!|N_2|\!\}_{PubK_Y} \\ Y & \rightarrow & X & : & \{\!|N_1, N_2|\!\}_{PubK_X} \end{array}$$

Later, CORAL derives a clause representing this trace:

$$\begin{array}{rcccl} Y & \rightarrow & X & : & \{\!|N_1, N_2|\!\}_{PubK_X} \\ spy & \rightarrow & Y & : & \{\!|N_2|\!\}_{PubK_Y} \end{array}$$

This second clause is subsumed by the first, and so deleted as redundant. This is because the second clause is identical except that it contains an additional literal, so it must be a more specific instance of the original clause. This kind of subsumption is used to prune 219 clauses from the search space during the rediscovery of the NSPK attack.

This search pattern illustrates the way CORAL genuinely rediscovers the attack, without any prior knowledge about who should be involved in the attack and what role they should play in the protocol. However, this openness also explains why CORAL finds the attack comparatively slowly. In the first gap in the search, it considers many different honest principals being involved in a run, all represented by uninstantiated variables, and how different bindings of the variables could possibly lead to an attack. Of course, this turns out not to be possible - a dishonest player must involved. Although subsumption checking rules out the checking of a lot of equivalent clauses, this still takes up a lot of time. Other tools that are constrained to use one of two possible ground terms to represent honest agents would not suffer this problem, and tools that predetermine what roles the agents will play even less so (see §8.8).

8.3 Attacking a Clark-Jacob Protocol

On page 26 of their security protocol survey, [Clark and Jacob, 1997], Clark and Jacob give a protocol intended to demonstrate the principle of parallel session attacks. The protocol and the attack are very simple indeed, but the attack does demand that an honest principal plays both roles in the protocol, i.e. he should be involved in two parallel sessions, one as an initiator playing the part of A, and one as a responder playing the part of B. This is something that some protocol models do not allow, e.g.

Weidenbach's, presented in §2.4.4.2. The protocol assumes two parties already share a key K_{AB}, and wish to establish a fresh shared nonce N_A to protect against replayed messages. Here is the protocol.

1. $A \rightarrow B : \{\!|\, N_A |\!\}_{K_{AB}}$
2. $B \rightarrow A : \{\!|\, s(N_A) |\!\}_{K_{AB}}$

The only issue in modelling this protocol was the symmetry of key K_{AB}. Normally, we represent symmetric keys in our formalism as $key(n)$ with n a number, assigned by the key server. However, for this simple handshake, we are assuming the key has already been given out. So we model the key as $key(A,B)$, with A and B principal names. The problem is now that syntactically $key(A,B) \neq key(B,A)$, but in fact they are the same key. So, we must add a commutativity axiom to our model for such keys, i.e:

key(pair(X,Y))=key(pair(Y,X))

Being able to add these kinds of axioms is one of the advantages of using a theorem prover that supports equational reasoning. All the clauses for the model are given in Appendix A.

The attack arises when CORAL is given a simple authenticity guarantee for A. The guarantee for A says that if A has initiated a protocol run with nonce N_A, and receives a response containing $s(N_A)$, then B must at some point have sent a response containing $s(N_A)$. The conjecture in CORAL's formalism looks like this:

% *Trace* is a valid trace
m(Trace)=true∧

% *A* and *B* are two honest agents
eqagent(A,spy)=false∧*eqagent(B,spy)=false*∧

% *A* has started a run
member(sent(A,B,encr(nonce(NA),key(pair(A,B)))),Trace)=true∧

% someone (*X*) has sent a response
member(sent(X,A,encr(s(nonce(NA)),key(pair(A,B)))),Trace)=true∧

% B has not responded
member(sent(B,A,encr(s(nonce(NA)),key(pair(A,B)))),Trace)=false
$\rightarrow$

Note that this conjecture states that there is no trace in which A has started a run, A has received a response apparently from B, and B has not responded. A counterexample to this conjecture will be a trace in which these things *have* occurred in conjunction. CORAL finds the following counterexample in 13 seconds:

```
  a --> s(a)      :  { N(0) }_K((a,s(a))

spy --> a         :  { N(0) }_K((s(a),a)

  a --> s(a)      :  { s(N(0)) }_K((s(a),a)

spy --> a         :  { s(N(0)) }_K((a,s(a))
```

We can see that CORAL has found the same attack that Clark and Jacob used the protocol to demonstrate, viz.:

1. $A \rightarrow C_B : \{\!| N_A |\!\}_{K_{AB}}$

1'. $C_B \rightarrow A : \{\!| N_A |\!\}_{K_{AB}}$

2'. $A \rightarrow C_B : \{\!| s(N_A) |\!\}_{K_{AB}}$

2. $C_B \rightarrow A : \{\!| s(N_A) |\!\}_{K_{AB}}$

8.4 Attacking Neuman-Stubblebine

The Neuman-Stubblebine protocol was used as an example by Weidenbach is his first-order protocol model, [Weidenbach, 1999]. In order to show that CORAL extends this work, we wanted to be sure that CORAL can also find this attack. The attack is a type attack, but our formalism, like Paulson's, was explicitly designed to be strongly typed, taking into account the fact that is has been shown that type attacks can be eliminated from protocols, [Heather et al., 2000]. However, we can allow type confusion between nonces and keys (which is what is required to effect this attack) to occur by relaxing our formalism a little. The vital change is to specify in our conjecture only that the spy

has a term which has been sent to B as a key, and not to specify it is a term $nonce(X)$ or $key(X)$.

The Neuman-Stubblebine protocol was explained in §2.3.6. Here again is the key-establishment part of the protocol:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow S : B, \{\!|A, N_A, T_B|\!\}_{K_B}, N_B$
3. $S \rightarrow A : \{\!|B, N_A, K_{AB}, T_B|\!\}_{K_A}, \{\!|A, K_{AB}, T_B|\!\}_{K_B}, N_B$
4. $A \rightarrow B : \{\!|A, K_{AB}, T_B|\!\}_{K_B}, \{\!|N_B|\!\}_{K_{AB}}$

The clauses we used to model this protocol are given in Appendix A. The attack is found when we give CORAL the following conjecture, which is a secrecy guarantee for B:

% B has sent a message 2
member(sent(B,server,triple(principal(B),encr(triple(principal(A),nonce(NA), nonce(T)),longtermkey(B)),nonce(NB))),Trace)=true$\wedge$

% B has been sent key *Key* (note - untyped)
member(sent(X,B,pair(encr(triple(principal(A),Key,nonce(T)),longtermkey(B)), encr(nonce(NB),Key))),Trace)=true$\wedge$

% and that key is in the spy's knowledge
in(Key,analz(Trace))=true
$\rightarrow$

CORAL refutes this conjecture in 10 seconds giving the following counterexample:

```
   a --> s(a)     :  a , N(s(0))

s(a) --> server   :  s(a) , { a , N(s(U)),time(t) }_longtermK(s(a)) , N(0)

 spy --> s(a)     :  { a , N(s(0)),time(t) }_longtermK(s(a)) , { N(0)}_,K(N(s(0)))
```

This corresponds to the same attack Weidenbach found, originally discovered by Hwang et al., [Hwang et al., 1995]:

1. $A \rightarrow B : A, N_A$

2. $B \rightarrow S : B, \{\!|\, A, N_A, T_B \,|\!\}_{K_B}, N_B$

4. $C_A \rightarrow B : \{\!|\, A, N_A, T_B \,|\!\}_{K_B}, \{\!|\, N_B \,|\!\}_{N_A}$

8.5 Attacking BAN Otway-Rees

An important example of an attack which is not in the standard corpus, [Clark and Jacob, 1997], is the attack on the simplified Otway-Rees protocol proposed by Burrows, Abadi and Needham, [Burrows et al., 1990]. The Otway-Rees attack, discovered by Paulson, [Paulson, 1998], is a significant example in that it requires one agent to play both roles in the protocol, and also requires her to take part in two interleaving runs at the same time. Formal approaches relying on small abstractions of the protocol problem, e.g. Weidenbach's, §2.4.4.2, are not able to find these kinds of attacks. It also requires quite a lot of disassembling and reassembling of messages by the intruder. As such it seemed a good example to complete the development set. Here is the protocol, which we presented and explained in §2.3.5:

1. $A \rightarrow B : N_A, A, B, \{\!|\, N_A, A, B \,|\!\}_{K_A}$

2. $B \rightarrow S : N_A, A, B, \{\!|\, N_A, A, B \,|\!\}_{K_A}, N_B, \{\!|\, N_A, A, B \,|\!\}_{K_B}$

3. $S \rightarrow B : N_A, \{\!|\, N_A, K_{AB} \,|\!\}_{K_A}, \{\!|\, N_B, K_{AB} \,|\!\}_{K_B}$

4. $B \rightarrow A : N_A, \{\!|\, N_A, K_{AB} \,|\!\}_{K_A}$

The details of the clauses used to model the Otway-Rees protocol are given in Appendix A. When Paulson originally analysed the protocol, [Paulson, 1998], he found he could not prove a conjecture about the security of the key received by Alice. Eventually he was able to discover the reason for the failed proof – there is an attack on the protocol. Finding this attack automatically from such a conjecture is precisely what we want CORAL to be able to do. The conjecture Paulson could not prove stated that if Alice starts a run with a message 1 using nonce N_A, and then receives a key from the

server under her long term key K_A tagged with the same nonce N_A, then the key must be secret, i.e. not be known to the spy. We state the conjecture the same way:

> % A started a run with nonce NA
> *member(sent(A,B,quad(nonce(NA),principal(A),principal(B),*
> *encr(triple(nonce(NA),principal(A),principal(B)),*
> *longtermkey(A)))),Trace)=true*
>
> % message 4 received by A with same nonce NA
> *member(sent(X,A,pair(nonce(NA),encr(pair(nonce(NA),key(K)),*
> *longtermkey(A)))),Trace)=true*
>
> % the spy has that key
> *not(equal(in(key(K),analz(Trace))=true*
> $\rightarrow$

After 7 minutes 12 seconds, CORAL gives the counterexample:

```
  a --> s(a)      :  N(2) , a , s(a) , { N(2) , a , s(a) }_longtermK(a)

 spy --> a        :  N(0) , spy , a , { N(0) , spy , a }_longtermK(spy)

  a --> server    :  N(0) , spy , a , { N(0) , spy ,a}_longtermK(spy),
                     N(1) , { N(0) , spy , a }_longtermK(a)

 spy --> server   :  N(0) , spy , a , { N(0) , spy , a}_longtermK(spy) ,
                     N(2) , { N(0) , spy , a }_longtermK(a)

server --> a      :  N(0) , { N(0) , K(0) }_longtermK(spy) ,
                     { N(2) , K(0) }_longtermK(a)

  spy --> a       :  N(2) , { N(2) , K(0) }_longtermK(a)
```

which represents the same attack that Paulson found:

1. $A \rightarrow C_B : N_A, A, B, \{\!| N_A, A, B |\!\}_{K_A}$

1.' $C \rightarrow A : N_C, C, A, \{\!| N_C, C, A |\!\}_{K_C}$

2.' $A \rightarrow C_S : N_C, C, A, \{\!| N_C, C, A |\!\}_{K_C}, N_{A'}, \{\!| N_C, C, A |\!\}_{K_A}$

2.'' $C_A \rightarrow S : N_C, C, A, \{\!| N_C, C, A |\!\}_{K_C}, N_A, \{\!| N_C, C, A |\!\}_{K_A}$

3.'' $S \rightarrow C_A : N_C, \{\!| N_C, K_{CA} |\!\}_{K_C}, \{\!| N_A, K_{CA} |\!\}_{K_A}$

4. $C_B \rightarrow A : N_A, \{\!| N_A, K_{CA} |\!\}_{K_A}$

8.6 Development of Heuristics

In order the find the very simple attack on the didactic Clark-Jacob protocol, [Clark and Jacob, 1997, p.26], only the invalid term elimination rule (§7.1) is necessary, though without the others the attack takes longer to find. For the NSPK protocol and the Neuman-Stubblebine protocol, [Needham and Schroeder, 1978, Neuman and Stubblebine, 1993], the literal selection heuristic (§7.4), the faked messages heuristic (§7.2.1), the rule preventing the spy sending two messages in a row (§7.2.4), and the invalid term elimination heuristic (§7.1) were required to find the attacks, though again, without the other optimisations CORAL needs much more time. For the attack on the BAN Otway-Rees protocol, [Burrows et al., 1990], the heuristics governing what terms the spy could expect to see, described in §7.2.2 and §7.2.3, were needed to find the attack. The Otway-Rees protocol has much longer messages that the others, giving the spy many more possibilities for combining terms that might eventually make up a valid message. It was this example which lead us to develop the heuristics described in §7.2.2 and §7.2.3. The heuristic that prunes away unsatisfiable *parts* literals, described in §7.3, was developed after all the development examples had been attempted, but before testing on the test set. It lead to much better times on all protocols, up to a factor of four on the Otway-Rees protocol.

We added code to CORAL to allow us to profile the amount of time used to apply these heuristics. In Table 8.2, we show the results of this profiling on the NSPK protocol and the Otway-Rees-BAN protocol. Thanks to our use of the term indexing for implementing the most complex heuristic (§7.2.3), the total time used is for domain-specific reductions is only a very small fraction of the total time taken.

NSPK	
Total problem time	37.30 sec
Time spent on reductions	34.85 sec
Of this:	
Rewriting	20.53 sec
Subsumption checking	10.01 sec
Security protocol heuristics	0.29 sec
Other reductions	4.02 sec
Otway-Rees-BAN	
Total problem time	7 min 10.95 sec
Time spent on reductions	6 min 56.02 sec
Of this:	
Rewriting	4 min 52.86 sec
Subsumption checking	1 min 34.74sec
Security protocol heuristics	1.49 sec
Other reductions	26.93 sec sec

Table 8.2: Pro£le of CORAL's reduction time

8.7 The Test Set of Protocols

The test protocols were chosen from the Clark-Jacob corpus, [Clark and Jacob, 1997], the standard corpus in the field. There are five different kinds of attack demonstrated in the corpus: replay, man in the middle (MITM), type confusion, parallel session, and attacks based on the compromise of a short-term secret (STS). We chose two of each kind, giving a test set of ten protocols. There are about 33 examples in the corpus in total[1]. Our main reason for not devoting more time to testing CORAL on the whole corpus was that we did not expect CORAL to produce ground–breaking performance in terms of attack finding speeds. Other tools have been built from the ground up to analyse these kinds of standard protocols, strictly in a Dolev-Yao intruder scenario, with tailor made representations and pre-setting of the roles to be played by particular agents in order to find the attacks in the minimum possible time, e.g. [Basin et al., 2003]. As explained in Chapter 1, the aim of CORAL was to build a ¤exible tool with the advantages of Paulson's approach in that it can be easily adapted to slight changes in the protocol scenario, such as a weaker intruder, or a protocol with an unbounded number of different roles and agents, something that would be very difficult with tools tailored to Clark-Jacob protocols. We investigate the ¤exibility of CORAL in the case studies in the next two chapters, and discuss the issue further in our evaluation of CORAL with respect to related work in Chapter 11. Notwithstanding these provisos, we felt that a test on some standard protocol was necessary, to demonstrate CORAL's coverage of known results.

8.8 Results

The results of CORAL on the ten examples from the corpus are given in Table 8.3. All timings are taken on a Pentium IV Linux box. CORAL found all 10 of the attacks we searched for, 5 of them in under 10 seconds, 8 of them in under a minute, 9 in under 15 minutes and all ten in under half an hour. The total run time for all ten is a little

[1] The exact number is a matter of a interpretation. For example, some protocol analysis tool designers count only one attack for each protocol, and others include type attacks that other authors consider dubious.

under 43 minutes and 30 seconds. The parallel session attack on the Andrew-RPC protocol took a long time to find because it is 8 steps long, and requires the symmetry of the key to be exploited 4 times. In fact this attack, though mentioned in the Clark-Jacob survey, has often been left out in the testing of other protocol analysis tools, e.g. [Basin et al., 2003], where only the simpler replay and type ¤aw attacks are found.

CORAL's times are comparable with those achieved on the same examples by some competing tools, e.g. [Chevalier and Vigneron, 2002]. However, other tools can achieve much more impressive run times - a fraction of a second for most of the attacks in the corpus, even on significantly slower hardware, [Basin et al., 2003, Song et al., 2001]. There are two key reasons for this. One is that these tools were designed and optimised for attacking these standard protocols. They model the protocol and a system entirely under the control of the intruder, where each principal is considered to be moved between states by the intruder under the rules of the protocol. This means they do not have to re-infer information from the trace about what messages a principal is expecting to receive when considering whether he should respond to a message, as we have to in CORAL. Instead, it is encoded in the state information for that principal. We could not practically do this in CORAL without pre-determining the number of principals, since we cannot have rules applying to arbitrary predicates in a first-order model. This leads on to the second reason for these tool's better performance - they restrict the number of agents involved in a run in advance. Indeed, the tools described in [Basin et al., 2003] and [Chevalier and Vigneron, 2002] go further in requiring the user to choose a *scenario* before modelling takes place. To rediscover Lowe's attack on the NSPK protocol (§2.3.3), these tools require the user to tell the tool to investigate a scenario where Alice starts one run with the spy, and Bob expects Alice to start a run with him. The problem is then compiled so that just the right agent identifiers are included, and the roles of the agents are set up exactly for the attack scenario (in [Chevalier and Vigneron, 2002], the number of nonces is also bounded). It is arguable how much 'rediscovery' is really going on here - finding the right scenario constitutes most of the work for finding this attack. Of course one could automatically generate all possible scenarios for two agents and a spy, but this is not included in the timings given in [Basin et al., 2003] and [Chevalier and Vigneron, 2002]. In §8.2.1

above, we analysed how CORAL finds the NSPK attack with no previous knowledge of the roles that are required to find it. This explains a little why CORAL's times are so much slower. Additionally, for a protocol involving a pre-existing symmetric key, like the Andrew RPC protocol, these tools assume there is only one key available and give it an atomic value, thus ruling out problems with modelling the symmetry of the key.

Athena, [Song et al., 2001], can discover the attacks in comparable times without any pre-setting of roles and protocol instances. It relies on other simplifications - for example, only atomic keys can be used, and agents can only encrypt and decrypt with keys they already have, type attacks cannot be discovered, and each role in the protocol must be finite. This would rule out study of protocols like SSL and SET, which can be analysed in the Paulson model, [Paulson, 1998], and group protocols, including the protocols we analysed using CORAL in the next two chapters.

We were pleased that little effort was required to specify the protocols in our formalism and discover the attacks using CORAL. The script for preparing intruder messages described in §7.2.2 functioned properly, saving considerable time. Besides changing the messages, the only change to the formalism that was required to complete the test set was for the Hwang-Chen protocol, where we needed to model agents signing messages with their secret keys. This required us to add two clauses to the intruder model, one to allow him to sign messages using his own key, and one allowing him to remove the signature from an item signed by someone else using their secret key. With the addition of these clauses the attack was found without difficulty. Furthermore, no new heuristics were required for the test protocols.

8.9 Summary

In summary, we were pleased with CORAL's coverage of the test set, and though the times are much slower than for some tools, the fact that CORAL achieved these results without a predefined 'scenario' and without restrictions on the number of agents, nonces or roles, or the nature of keys or encryption/decryption operations meant we were confident of success in analysing some new protocols outside the scope of existing tools. In the next two chapters, we describe how we used CORAL to do that.

Protocol	Attack Type	Attack Time
ISO Two Pass mutual authentication	Replay	2.24 sec
Andrew RPC	Replay	51.21 sec
Andrew RPC	Parallel Session	29 min 43.36 sec
Woo-Lam Π	Parallel Session	9.98 sec
Needham-Schroeder shared key	STS	5.64 sec
Neuman-Stubblebine (repeated authentication part)	STS	11 min 28.99 sec
Otway-Rees	Type Attack	12.32 sec
Neuman-Stubblebine (complete)	Type Attack (on final part)	51.84 sec
Hwang-Chen SPLICE	MITM	2.38 sec
Needham-Schroeder signature protocol	MITM	0.69 sec

Table 8.3: CORAL Attacking Protocols from the Clark-Jacob corpus

Chapter 9

Case Study 1: The Asokan–Ginzboorg Protocol

It is important in the evaluation of any protocol analysis tool to test its ability to find new protocol attacks. One advantage of the inductive formalism used in CORAL is that is gives us the ability to reason about protocols where an arbitrary number of agents might be involved in a single run, e.g. group key protocols. Attacking these protocols is something other automated approaches have struggled with (see §11.4). To test CORAL's ability to find new protocol attacks, and also to test its suitability for analysing multi-agent protocols, we selected two relatively new protocols for analysis. The first is a protocol for group key agreement that had not been analysed before, the Asokan–Ginzboorg protocol, [Asokan and Ginzboorg, 2000]. We describe the process of modelling the protocol and discovering 3 new attacks in this chapter. The second protocol is for group key management in a scenario where agents may join and leave the group at any time, and is covered in the next chapter.

9.1 Description of the Asokan–Ginzboorg Protocol

Bluetooth is a method for data communication that uses short-range radiolinks to replace cables between computers and peripherals. A key idea of the technology is that no setting up should be required in order to get a new Bluetooth enabled device talking

to all the other Bluetooth devices in a room. Protocols exist to facilitate this, but they contain little provision for ad-hoc security, i.e. in a situation where we don't know in advance who is going to want to connect, and whether or not they are authorised to do so. Instead, it is left up to application programmers to take these considerations into account.

Asokan and Ginzboorg have proposed a protocol for one such application, [Asokan and Ginzboorg, 2000]. The scenario they consider is this: a group of people are in a meeting room and want to set up a secure session amongst their laptops. They know and trust each other, but their computers have no shared prior knowledge and there is no trusted third party or public key infrastructure available. The protocol proceeds by assuming a short group password is chosen and displayed, e.g. on a whiteboard. The password is assumed to be susceptible to a *dictionary attack*, i.e. it could be guessed in a reasonable time by an attacker trying all the words from a normal dictionary. However, the participants in the meeting use the 'soft secret' of the password to establish a secure secret key. The protocol is *contributory*, i.e. all participants contribute a secret number of their own to the final key, and the key is calculated from these numbers using a carefully chosen function, thus ensuring a subgroup of dishonest participants cannot conspire to keep the key within a certain restricted range. The idea of restricting the key to a small range would be to make it easily breakable by a co-conspirator outside the room. The key calculation function would be chosen to ensure each agent's contribution may vary the key over its full range, much like a hashing function.

Asokan and Ginzboorg describe two protocols for establishing such a key in their paper, [Asokan and Ginzboorg, 2000]. The first involves 'black box' public key and shared key encryption, such as is generally modelled by protocol analysis tools. It is this protocol we have analysed in CORAL. The second protocol uses a lower level encryption method involving Diffie-Hellman style exponentiation that we can't model in CORAL yet (however, see §12).

Here is a description of the first Asokan–Ginzboorg protocol (which we will hereafter refer to as simply 'the Asokan–Ginzboorg protocol'). Let the group be of size n for some arbitrary $n \in \mathbb{N}, n \geq 2$. We write the members of the group as M_i, $1 \leq i \leq n$,

with M_n acting as group leader.

1.	M_n	$\rightarrow$	ALL	:	$M_n, \{\| E \|\}_P$	
2.	M_i	$\rightarrow$	M_n	:	$M_i, \{\| R_i, S_i \|\}_E$	$i = 1, \ldots, n-1$
3.	M_n	$\rightarrow$	M_i	:	$\{\| \{Sj, j = 1, \ldots, n\} \|\}_{R_i}$	$i = 1, \ldots, n-1$
4.	M_i	$\rightarrow$	M_n	:	$M_i, \{\| S_i, h(S_1, \ldots, S_n) \|\}_K$	some i, $K = f(S_1, \ldots, S_n)$

Informally, what is happening in a protocol run is this:

1. M_n broadcasts a message containing a fresh public key, E, encrypted under the password, P, written on the whiteboard.

2. Every other participant M_i, for $i = 1, \ldots, n-1$, sends M_n a contribution to the final key, S_i, and a fresh symmetric key, R_i.

3. Once M_n has a response from everyone in the room, she collects together the S_i in a package along with a contribution of her own (S_n) and sends out one message to each participant, containing this package $S_1, \ldots, S_n$ encrypted under the respective symmetric key R_i.

4. One participant responds to M_n with the package he just received passed through a one way hash function $h(.)$ and encrypted under the new group key $K = f(S_1, \ldots, S_n)$, with f a commonly known function.

Asokan and Ginzboorg argue that it is sufficient for each group member to receive confirmation that one other member knows the key: everyone except M_n receives this confirmation in step 3. M_n gets confirmation from a random member of the group in step 4. Once this message is received, the protocol designers argue, agents $M_1, \ldots, M_n$ must all have the new key $K = f(S_1, \ldots, S_n)$. A spy eavesdropping on Bluetooth communications from outside the room cannot know the key, and nor can he prevent the agents in the room from setting up a key by sending spurious messages of his own. We investigate these claims in §9.4.

9.2 Modelling the Protocol

Our methodology for modelling the protocol was the same as for fixed 2 or 3 principal protocols (see §6.5), i.e. to produce one rule for each protocol message describing how a trace may be extended by that message, taking into account the tests which an honest agent will apply. So for message 2, our rule express the fact that an agent will only send a message 2 if he has seen a message 1 in the trace, and will only use fresh numbers S_i and R_i in his message. However, for the Asokan–Ginzboorg protocol, message 3 posed a problem. For a group of n participants, $n-1$ message 3s will be sent out at once, each encrypted under a different key. Moreover, the group leader for the run must check that she has received $n-1$ message 2s. In order to model this without predetermining the size of the group, we needed to accommodate the possibility of different numbers of messages being added to the trace in one instant. This problem was solved by the use of a logic programming style approach. Note that this was only required for modelling honest agents sending message 3, since they have to conform to the protocol. The intruder can send any combination of message 3s, no matter what message 2s have appeared in the trace. He is only constrained by what knowledge he can extract from previous messages in the trace, by the same rules as for regular protocols.

9.2.1 Modelling Message 3

Figure 9.1 gives the clauses we used to model message 3 of the protocol. This requires some explanation: we will examine it clause by clause. Clause 1 models the addition of message 3s to the protocol. The first three literals require that *A* is an honest agent, *MN* another agent, and *Trace* is a valid trace. The 4th literal requires that a message 1 has been sent by agent *MN* at some point in the trace. The 5th literal then calls the `all_msg2s_received` function on the old trace. This instantiates the new trace variable, *NewTrace*, which we assert as valid in the antecedent literal.

The purpose of the `all_msg2s_received` function is to check that sufficient message 2s have been received for agent *MN* to make a response, and also to work out what that response should be. To avoid unnecessary duplicate representations for the equivalent protocol runs, we require that *MN*, the leader in this run, be the highest numbered

1.
eqagent(A,MN)=false $\wedge$ *eqagent(A,spy)=false* $\wedge$ *m(Trace)=true* $\wedge$
member(sent(MN,all,pair(principal(MN),encr(key(E),key(P)))),Trace)=true $\wedge$
all_msg2s_received(Trace,A,MN,E,nil,NewTrace,FinalPackage)=true $\rightarrow$
m(NewTrace)=true

2.
member(sent(X,MN,pair(principal(a),
encr(pair(nonce(Ri),nonce(Si)),key(E)))),Trace)=true$\rightarrow$
all_msg2s_received(Trace,a,MN,E,Package,
cons(sent(MN,a,encr(cons(nonce(Si),Package),nonce(Ri))),Trace),
cons(nonce(Si),package)=true

3.
member(sent(X,MN,pair(principal(s(MX)),
encr(pair(nonce(Ri),nonce(Si)),key(E)))),Trace)=true$\wedge$
all_msg2s_received(Trace,MX,MN,E,cons(nonce(Si),Package),
NewTrace,FinalPackage)=true$\rightarrow$
all_msg2s_received(Trace,s(MX),MN,E,Package,
cons(sent(MN,s(MX),encr(FinalPackage,nonce(Ri))),
NewTrace),FinalPackage)=true

Figure 9.1: Clauses for modelling message 3

agent in our numbering system which runs $a, s(a), s(s(a)), \ldots$. The arguments to the `all_msg2s_received` function run like this:

Trace	The trace of messages exchanged
MX	At the first call to the function, the agent receiving the first message 3. In subsequent calls, the next agent, until $MX = a$, the first agent in our model.
MN	Agent who initiated the run
E	Public key used in message 1 of the run
Package	Accumulator for the packaged S_i
NewTrace	The new trace with the message 3s in
FinalPackage	The final package of S_is to be sent out, remains uninstantiated until the base case of all_msg2s_received is satisfied.

Clause 2 in Figure 9.1 is the base case of the definition of `all_msg2s_received`. It checks that is there is a message 2 (apparently) from agent a in the trace sent under the correct public key and contributing $nonce(Si)$ to the final package $S_1, \ldots, S_n$, used to form the group key. If so, the antecedent literal asserts that this trace qualifies as having sufficient message 2s if the first recipient of a message 3 is a, and instantiates the response to be just that message 3, sent under the appropriate key R_i. The package of $S_1, \ldots, S_n$ to be sent is instantiated as $nonce(Si)$ added to the packaged $S_1, \ldots, S_n$ passed in the accumulator (argument 5). The 7th argument is also instantiated to this final package.

Clause 3 in Figure 9.1 models the recursive case of the definition. The antecedent literals require that all appropriate message 2s are in the trace up to that apparently from agent MX, and that a message 2 from agent $s(MX)$ is also in the trace. Then the succedent literal states that all message 2s must have been received up to $s(MX)$, making the appropriate instantiations to the new trace argument. Note that the message 3 added to the trace refers to the final package of $S_1, \ldots, S_n$, which is as yet still a pure variable but gets instantiated in the base case. In the recursive call in the antecedent, agent $s(MX)$'s contribution to the package is added to the accumulator argument.

This is fairly complex stuff for a protocol model, but shows some advantages of a

theorem proving approach for these very ¤exible protocols. We have variables available for things like accumulators and putting together response sequences, and can do some fairly involved deduction to work out what responses are required. A key feature of this part of the formalism is that it works backwards, i.e. given pure variables it will give traces containing a valid set of message 1s, 2s and 3s. This is vital, since it is in the nature of CORAL to start from a security conjecture and work backwards from the final message to the first to produce a counterexample trace. The full listing of the specification for the Asokan–Ginzboorg protocol is given in Appendix B.

9.2.2 Ordering Considerations

One final complication of the Asokan–Ginzboorg model is that it requires us to deal with *compound keys*, i.e. keys that are built out of the contributed numbers coming from the other members of the group. In message 3 the leader sends out a package that contains the numbers needed to make the key, $S_1, \ldots, S_n$. This is easily represented as a list in our model, $cons(s1, cons(\ldots cons(sn, nil)\ldots)$. However, we need to allow the spy to be able to take the list apart and make new packages. This requires additional *synth* and *analz* (see §6.6) rules to be added, viz.:

in(cons(nonce(X),Y),analz(XSET))=true
  → *in(nonce(X),analz(XSET))=true*

in(cons(nonce(X),Y),analz(XSET))=true
  → *in(Y,analz(XSET))=true*

in(nonce(X),analz(XSET))=true∧
in(cons(Y,Z),synth(analz(XSET)))=true
  → *in(cons(nonce(X),cons(Y,Z)),synth(analz(XSET)))=true*

in(nonce(X),synth(analz(XSET)))=true
  → *in(cons(nonce(X),nil),synth(analz(XSET)))=true*

This gives the spy the abilities we would like him to have, but it has the unwanted side effect of making our adapted term ordering non-well-founded (see §6.7), because we now have an infinite sequence:

$$in(U, analz(Y)) \succ$$
$$in(cons(U,V), analz(Y)) \succ$$
$$in(cons(U,(cons(V,W))), analz(Y)) \ldots$$

A similar problem has been encountered by other designers of protocol analysis tools in one form or another for such keys, and the most common solution is to fix a bound on the depth of the terms. However, this would constrain the size of the group, which we don't want to do. So, we have left things alone. In theory, this means our inference system for this protocol is not necessarily refutation complete. Nevertheless, it still successfully discovers new attacks as we shall see below. This is probably because although CORAL could consider the possibility of the spy obtaining a particular nonce from larger and larger packages of key contributions, this would require larger and larger clauses to be considered, which will not be favoured by CORAL's basic heuristic of considering the least weight clause first. There may be better solutions to this problem though, and we discuss this in § 12.

9.3 Modelling Spies in a Wireless Network

The Dolev-Yao model (see §2.3.2) is widely agreed upon as a realistic model of a spy in a fixed network. However, some of the Dolev-Yao assumptions seem inappropriate in a wireless situation. Although an attacker may be able to totally disrupt communications by jamming the radio signal, it seems unlikely that he would be able to selectively intercept some messages and remove them from the airwaves, while allowing other messages to pass[1]. In fact, Asokan and Ginzboorg mention that they intend the protocol to be tolerant to *disruption attacks*[2] by an attacker who can add fake messages, but not block or delay messages, so we assume these abilities for our spy.

There is also the question of whether the spy should be accepted as an honest participant by the other players, as he generally is in fixed-network protocol models. An honest participant will be present in the room and able to see the password on the

[1] We are assuming here that a "single-hop" wireless network is used in the meeting room, i.e. participants receive messages direct from the sender. If an intruder was able to interpose a machine in a multi-hop network, then the usual Dolev-Yao assumptions would apply.

[2] A disruption attack is an attack whereby a spy prevents the honest agents from completing a successful run of the protocol.

whiteboard, whereas the protocol is primarily intended to protect against attacks by a spy in an adjacent room, who can interfere with communications but who does not know the password. However, the protocol is also explicitly designed, by means of the contributory key generation, to protect participants against conspiracy by a group of other participants trying to restrict the agreed key to a pre-chosen range. Some possible dishonesty from agents in the room must therefore be considered. We decided to test the protocol against two attackers: one inside the room, and one outside. Different outcomes are considered successful for the different spies.

Spy 1 - Outside the Room

This spy cannot see the whiteboard, so does not know the password. His objective is to effect a disruption attack. Since he cannot block messages or remove them from the airwaves, he must do this by adding messages in such a way as to disrupt the protocol run, e.g. by making honest participants accept keys which are not mutually shared.

Spy 2 - In the Room

This spy's capabilities differ from the first in that he knows the passwords written on the whiteboard, and is accepted as an honest agent. A disruption attack would be trivial for this spy (he could just refuse to send any messages), so instead his objective is to gain control of communication in the room, i.e. by making all participants agree on different keys that only he knows.

9.4 Attacking the Protocol

The advantage of our approach is that we can model flexible group protocols without having to restrict ourselves to a small concrete instance with a fixed number of participants. This means that we don't have to guess how many players are needed to achieve an attack, CORAL will search for attacks involving any number of participants.

As we have explained in the previous chapters, finding attacks in CORAL results from finding counterexamples to security properties. These are formulated in a similar

way to Paulson's model. We must formulate the required properties in terms of the set of possible traces of messages.

9.4.1 Attacks by a Spy Outside the Room

The following conjecture was used to check for disruption attacks:

%% some honest xi has sent message 4, so has key *f(Package)*:
eqagent(XI,spy)=false $\wedge$
member(sent(XI,XK,pair(principal(XI),
$encr(pair(nonce(SI),h(Package)),f(Package)))),Trace) = true$

%% genuine messages 3 and 1 are in the trace to some agent *XJ*:
member(sent(MN,XJ,encr(Package,nonce(RJ))),Trace)=true
member(sent(MN,all,pair(principal(MN),encr(key(E),key(P)))),Trace)=true

%% but XJ never sent a message 2 under public key E with nonces SJ
%% (which is in *Package*) and RJ (which the message 3 meant for
%% him was sent under). That means he doesn't have RJ, and so can't
%% get the key from his message 3.
member(nonce(SJ),Package)=true
member(sent(XJ,MN,pair(principal(XJ),encr(pair(nonce(RJ),nonce(SJ)),
key(E)))),Trace)=false
$\rightarrow$

This conjecture may look somewhat contrived, but it is a natural way of expressing that a run had been finished (i.e. a message 4 has been sent) but that there is some agent who does not now have the key. We assume that the disruption comes about as a result of one agent not being able to read the message 3 intended for him. This is because the leader can see how many people are in the room, and so will at the very least send out one message 3 to each other member of the group. With the spy outside the room, only honest agents can generate a correct message 1, so the leader cannot be the spy.

Note that conjecture is negative, i.e. it says that for all possible traces, no trace can have the combination of genuine messages 4,3 and 1 without a corresponding message 2. When first run with this conjecture, CORAL produces the following counterexample for a group of size 2:

$$
\begin{array}{llcll}
1. & M_2 & \rightarrow & \text{ALL} & : M_2, \{\!|\, E \,|\!\}_P \\
1'. & spy_{M_1} & \rightarrow & \text{ALL} & : M_1, \{\!|\, E \,|\!\}_P \\
2'. & M_2 & \rightarrow & M_1 & : M_2, \{\!|\, R_2, S_2 \,|\!\}_E \\
2. & spy_{M_1} & \rightarrow & M_2 & : M_1, \{\!|\, R_2, S_2 \,|\!\}_E \\
3. & M_2 & \rightarrow & M_1 & : \{\!|\, S_2', S_2 \,|\!\}_{R_2} \\
3'. & spy_{M_1} & \rightarrow & M_2 & : \{\!|\, S_2', S_2 \,|\!\}_{R_2} \\
4.' & M_2 & \rightarrow & M_1 & : M_2, \{\!|\, S_2, h(S_2', S_2) \,|\!\}_{f(S_2', S_2)}
\end{array}
$$

At the end of the run, M_2 now accepts the key $f(S_2', S_2)$ as a valid group key, but it contains numbers known only to M_2. The attack requires that the spy manages to send message 2 before the honest agent M_1 can send her reply. In a single hop network, there is a certain amount of luck involved. It might require M_1 to invoke the command to establish a key just after M_2 does. This attack also assumes that the implementation of the protocol does not protect against parallel sessions, i.e. by stopping an agent from taking part in two concurrent attempts to set up a key. This is certainly possible, so we should alter the protocol to protect against it. To do so is quite straightforward (see §9.5). However, when we corrected the protocol to protect against this attack, and ran CORAL again with the same conjecture, CORAL found the following counterexample for a group of size 3:

$$
\begin{array}{llcll}
1. & M_1 & \rightarrow & \text{ALL} & : \{\!|\, M_1, E \,|\!\}_P \\
2. & M_2 & \rightarrow & M_1 & : M_2, \{\!|\, R_2, S_2 \,|\!\}_E \\
2. & spy_{M_3} & \rightarrow & M_1 & : M_3, \{\!|\, R_2, S_2 \,|\!\}_E \\
3. & M_1 & \rightarrow & M_2 & : \{\!|\, S_2, S_2, S_1 \,|\!\}_{R_2} \\
3. & M_1 & \rightarrow & M_3 & : \{\!|\, S_2, S_2, S_1 \,|\!\}_{R_2} \\
4. & M_2 & \rightarrow & M_1 & : M_2, \{\!|\, S_2, h(S_2, S_2, S_1) \,|\!\}_{f(S_2, S_2, S_1)}
\end{array}
$$

This is another disruption attack, where the spy eavesdrops on the first message 2 sent, and then fakes a message 2 from another member of the group. This results in the protocol run ending with only two of the three person group sharing the key. This attack can also be prevented by a small change to the protocol (see §9.5 below). With these two attacks prevented, CORAL finds no further disruption attacks.

9.4.2 Attacks by a Spy Inside the Room

When considering a scenario where one of the agents inside the room is a spy, we were forced to consider what kind of behaviour might constitute an attack. Simple disruption attacks would be trivial - the spy could just refuse to send any messages. We decided to consider what might be possible when all the players in the room think they have agreed on a key, but they have in fact agreed on different ones. What if the spy knows all these keys? He could filter all the information exchanged, perhaps making subtle but important changes to a document in a pre-defined way, such that the other agents in the room are none the wiser. We checked for these kinds of attacks by giving CORAL the following conjecture:

% we have distinct honest agents XI and XJ
eqagent(XI,spy)=false $\wedge$
eqagent(XJ,spy)=false $\wedge$
eqagent(XJ,XI)=false $\wedge$

% they both sent message 2s
member(sent(XI,MN,pair(principal(XI),
$encr(pair(nonce(RI),nonce(SI)),key(E)))),Trace) = true \wedge$
member(sent(XJ,MN,pair(principal(XJ),
$encr(pair(nonce(RJ),nonce(SJ)),key(E)))),Trace) = true \wedge$

% and received message 3s under the correct keys, RI and RJ
member(sent(MN,XI,encr(Package1,nonce(RI))),Trace)=true$\wedge$
member(sent(MN,XJ,encr(Package2,nonce(RJ))),Trace)=true$\wedge$

% but the packages they received were different
eq(Package1,Package2)=false $\rightarrow$

Note again that the conjecture is negative, i.e. it states that there is no trace for which this combination of conditions can hold. CORAL refuted this property, producing the counterexample trace[3].

[3] It should be noted that, as a result of examining the protocol to create the model, we realised that an attack like this would be possible before CORAL found it.

1.	spy	$\rightarrow$	ALL	:	$spy, \{\| E \|\}_P$
2.	M_1	$\rightarrow$	spy	:	$M_1, \{\| R_1, S_1 \|\}_E$
2.	M_2	$\rightarrow$	spy	:	$M_2, \{\| R_2, S_2 \|\}_E$
3.	spy	$\rightarrow$	M_1	:	$\{\| S_1, S_2, S_{spy} \|\}_{R_1}$
3.	spy	$\rightarrow$	M_2	:	$\{\| S_1, S_2, S'_{spy} \|\}_{R_2}$
4.	M_1	$\rightarrow$	spy	:	$M_1, \{\| S_1, h(S_1, S_2, S_{spy}) \|\}_{f(S_1, S_2, S_{spy})}$

This attack is just a standard protocol run for three participants, except that in the first message 3, the spy switches in a number of his own (S'_{spy} in the place of S_2). This means that M_1 accepts the key as $f(S_{spy}, S_1, S'_{spy})$, whereas M_2 accepts $f(S_{spy}, S_1, S_2)$, and both of the keys are known to the spy.

Note that only the initiator of the run can effect this attack - the use of public key cryptography in message 2 prevents another participant from being able to carry out a similar deception. The attack can be easily adapted to a group of any size. We discuss how to prevent this attack below, §9.5.

9.5 An Improved Version of the Asokan–Ginzboorg Protocol

The disruption attacks are a result of the agents' identifiers being sent in cleartext in messages 1 and 2. This allows a spy to deceive the honest players with faked versions of these messages. The non-matching keys attack is a result of there being no way to confirm that everyone has received the same key. We can say the protocol is non-symmetric, in that the leader has an advantage over the other players, since he is in control of what keys everyone receives.

To protect against the first disruption attack, we can include the agent identifier inside the encrypted package in message 1. This will prevent the faked message 1'. The agent identifier in message 2 should also be encrypted. This change prevents the spy from faking message 2s as he does in the second disruption attack. With these changes, CORAL finds no more disruption attacks after considerable run-time.

Protecting against the non-matching keys attack is a more subtle problem: one way

to do it would be to have message 4 broadcast to all participants. This would make sense in a normal run as it would allow other participants to see that someone had sent message 4, and that the run was finished. Other participants could open the encrypted package and check that the hashed number matches the hash of the key they received in message 3. If anyone is unhappy, they could cry foul and get everyone to do another run. The spy could fake a message 4 from an agent who has been given a duff key, but since the message is broadcast to everyone, the agent concerned could recognise that he is being impersonated and call a halt to proceedings.
Here is the revised protocol:

$$
\begin{array}{lllll}
1. & M_n & \rightarrow & \text{ALL} & : \{\!|\, \boxed{M_n}\,, E |\!\}_P \\
2. & M_i & \rightarrow & M_n & : \{\!|\, \boxed{M_i}\,, R_i, S_i |\!\}_E, i = 1, \ldots, n-1 \\
3. & M_n & \rightarrow & M_i & : \{\!|\, \{Sj, j = 1, \ldots, n\} |\!\}_{R_i}, i = 1, \ldots, n-1 \\
4. & M_i & \rightarrow & \boxed{\text{ALL}} & : M_i, \{\!|\, S_i, h(S_1, \ldots, S_n) |\!\}_K, \text{ some } i.
\end{array}
$$

9.6 Summary

In general the results achieved in the case study were very pleasing. We were able to model a group protocol in a general way without predetermining the size of the group, and we discovered three new attacks, all quite novel and requiring unusual security properties to be formulated. However, formulating the 2 security properties was a non-trivial task. It took several attempts to get them right, the first incorrect attempts yielding traces that were quite clearly not attacks. In each case, the bugs were easy to track down. For example, for the conjecture about resistance to disruption attacks, we realised it was necessary to specify that nonce Sj be part of the package sent in a message 4. This need to debug the conjecture slowed down the attack finding process considerably, because CORAL takes a long time to run on these protocols, so some bugs took hours to come to light. In extremis, the second disruption attack took 73 hours run time to find (the first took 1 hour 30 mins, and the non-matching keys attack 3 hours 12 mins). This is not really acceptable compared to run times for competing tools, though no competing tools can attack group protocols in such a general fashion. This point is investigated more fully in the next chapter, where we evaluate CORAL

in the context of the most closely related work in the field. Some ideas for improving CORAL's run times are given in § 12.

Chapter 10

Case Study 2: The Tanaka–Sato Protocol

This chapter describes our analysis of a different kind of group protocol. The protocol in the previous chapter dealt with the problem of group key establishment, whereas the Tanaka–Sato protocol, [Tanaka and Sato, 2001], addresses the problem of group key *management*, where agents may join and leave the group at any time, and the group key must remain secure. Following the pattern of the previous chapter, we first describe the protocol, then describe the modelling process. We present the results, including two new attacks discovered by CORAL, and then propose an improved version of the protocol.

10.1 Description of the Tanaka–Sato Protocol (Taghdiri–Jackson version)

Unlike the Asokan–Ginzboorg protocol modelled in the previous chapter, the Tanaka–Sato protocol has been analysed using formal methods before, by Taghdiri and Jackson, [Taghdiri and Jackson, 2003]. The protocol was formalised in the Alloy specification language, [Jackson, 2002], and a SAT checker was used to search for counterexamples to desirable properties of the protocol. Several counterexamples were found, the most serious one indicating that current members of the group will accept as valid messages

broadcast by ex-members of the group. Taghdiri and Jackson proposed an improved protocol. However, their formal protocol model differed from the norm established over the last 25 years in that no active attacker was included. If anything, it would seem even more likely that a multicast protocol would be subject to attack by an active intruder compared to a unicast protocol, as argued in [Mittra, 1997]. There are inherently more opportunities for interception of traffic, and the 'crowd' of principals would typically make it easier for an intruder to pose as another legitimate principal. Such protocols should therefore be subjected to analysis under the full Dolev-Yao attacker model, as is standard for unicast protocols. In this chapter, we describe the modelling of Taghdiri and Jackson's improved protocol, and show how CORAL discovered two new attacks, just as serious as the ones the improvements were supposed to prevent.

The protocol that Tanaka and Sato originally proposed, called the Pull-Based Asynchronous Rekeying Framework, [Tanaka and Sato, 2001], was primarily concerned with minimising the burden of key updates in terms of network traffic and processor time. Two main design features were introduced for this purpose: the first was the division of the group into subgroups, each under the management of a key distribution server (KDS). The communication between the KDSs is assumed to be not only secure but also conducted under a reliable totally ordered multicast protocol (RTOMP). Taghdiri and Jackson, [Taghdiri and Jackson, 2003], modelled this by assuming that as soon as one KDS updates its key, all the other KDSs instantaneously update theirs, effectively reducing the model to a single server. Our model also makes this assumption. The second design feature of the protocol is that agents retain a list of keys rather than just one key. They discard an old key as invalid t units of time after having received a more up-to-date key, where t is set with respect to the delay in the network. Keys are distributed only when an agent sends a request to the server. An agent will make such a request when he wants to send a multicast message, or if he receives a message encrypted under a key he doesn't already have. In both cases, he will send a message to the server giving the ID number of the newest key he has, and the server will send back all newer keys. Only the newest key is used for multicast broadcasting.

This retention of a list of keys was shown in Taghdiri and Jackson's analysis to lead to major security problems. The most serious attack involved members of the group

accepting messages from a principal outside the group. An member of the group A can simply broadcast a message from inside the group, leave, and then broadcast a message using the same key. Though the group key has been updated as a result of A leaving, the other agents in the group will still accept the second message as valid as they all have the old key. To counter this, Taghdiri and Jackson suggested changes to the protocol. Each agent should retain only the most recent key he has received, and upon receiving a multicast message, should contact the server to confirm that it is encrypted under the newest key. This may result in some message loss, because delays in the network might mean that by the time a multicast message has been received and a key request sent to the server, the group key has changed, but this was reckoned to be acceptable compared to the potential security breach.

The Tanaka-Sato protocol assumes the existence of a unicast authentication protocol that allows the server to establish an individual key (IK) with a new member joining the group. This IK is used to encrypt all communication between that member and the server. We model the underlying authentication protocol by assuming the existence of a long-term key shared by each valid potential member of the group with the KDS. Since we are looking for attacks on the protocol rather than trying to verify it, we can easily justify this. We can simply take the attacks we discover and examine them to see if the specific way we implemented the authentication phase was exploited. The attacks described in this paper would be effective for any initial authentication protocol. Additionally, we make the standard assumption that the spy has access to one valid long term key, i.e. he is able to pose as a legitimate agent.

Here is a description of the improved version of the protocol as described by Taghdiri and Jackson:

Joining the Group

$$
\begin{array}{llllll}
1. & M_i & \rightarrow & S & : & \{\!|\, \text{join} \,|\!\}_{K_{M_i}} \\
2. & S & \rightarrow & M_i & : & \{\!|\, Ik_{M_i}, Gk(n) \,|\!\}_{K_{M_i}}
\end{array}
$$

In message 1, M_i wants to join the group, so sends a join request under his long term key K_{M_i}. The server generates a fresh individual key , $Ik_M i$, and a new group key $Gk(n)$. Each group key has a unique ID number (n). The new individual key and group key are sent to the joining member in message 2.

Leaving the Group

1. $M_i \rightarrow S : \{\!|\text{leave}|\!\}_{Ik_{M_i}}$
2. $S \rightarrow M_i : \{\!|\text{ack.leave}|\!\}_{Ik_{M_i}}$

In message 1, M_i sends a request to leave encrypted under his individual key Ik. The server acknowledges the leave in message 2, and generates a new group key. This key is not distributed though, and if another membership change occurs before a request for a key is received, it will never be distributed.

Sending a message

1. $M_i \rightarrow S : \{\!|\text{send},n|\!\}_{Ik_{M_i}}$
2. $S \rightarrow M_i : \{\!|\, n', Gk(n')|\!\}_{Ik_{M_i}}$
3. $M_i \rightarrow ALL : \{\!|\text{message}|\!\}_{Gk(n')}$

In message 1, agent M_i signals to the server that he would like to send a message by sending what the protocol designers call a 'sequence request' message together with the ID number of the newest key he has, n. The server checks that M_i is in the group, and then sends back the newest key $Gk(n')$. If no joins or leaves have occurred since M_i last received a key, it may be that $n = n'$, but this will not be the case in general. In message 3, agent M_i broadcasts his message to the group.

Receiving a message

1. $M_j \rightarrow S : \{\!|\text{read},n|\!\}_{Ik_{M_j}}$
2. $S \rightarrow M_j : \{\!|\, Gk(n')|\!\}_{Ik_{M_j}}$

Suppose a multicast message has been broadcast, as in message 3 of the 'sending a message' fragment above. When another agent M_j receives the message, he first sends a request to the server for the newest key. He then receives the newest key $Gk(n')$, and will only accept the multicast message if it was encrypted under that key.

10.1.1 Commentary

The revised protocol as proposed by Taghdiri and Jackson contains some redundancy as a result of their security improvements. For example, there is no reason for the server to send the key to a new member when he joins, since he is required to ask for a key update whenever he sends or receives a multicast message. Additionally, the

sequence number sent in the request for a key update before sending a message also seems redundant. Previously, the server would have used it to decide which keys to send back, but in the revised version, the server only ever sends back the most recent key. It would be better to replace this with a nonce, as we argue after presenting the attacks we discovered, in §10.4.

10.2 Modelling the Protocol

A feature of our model described in Chapter 6 is that all the information about the state of the system, i.e. the state and knowledge of all the principals involved, is stored in the trace and inferred from the trace each time it is needed. This was particularly useful when we were modelling the Asokan–Ginzboorg group key agreement protocol in the previous chapter. This was the key feature that allowed us to model message 3 in a general way, and so to perform the analysis without pre-setting the size of the group. However, for the Taghdiri-Jackson protocol, this was not so helpful. Some information about the state of the system does not normally appear in the trace. For example, when an agent leaves the group, the server generates a new key, but this key does not appear in the trace. A further consideration with this protocol is that we often need to know who is in the group, in order to model the control conditions, i.e. tests that honest agents apply before sending a protocol message. An honest agent will only apply to join the group if he is not already in it, and will only send a message if he is in the group etc. With the trace based model, a lot of examination of the trace would be required to determine group composition, and we would need to do this almost every time an agent sent a message, creating an enormous search problem. So, for this protocol, some slight changes to the model were made, to include some information about the state of the principals. The unary function $m()$ that was previously used to store just the message trace is now an arity 4 function storing the trace, a counter, the current group key stored by the server, and the composition of the group stored as a list of triples. The triples store the agents name, the individual key which he shares with the server for this session in the group, and the most recent group multicast key he has received. We define a boolean function *ingroup* on these lists of triples that

determine whether or not a particular agent is in the group. A further change is our modelling of freshness. We used to use the *parts* operator as used by Paulson, but in our model for this protocol, we have a counter, and use this to model fresh values. Our motivation for this was that so many fresh values have to be created in a typical scenario, for individual keys, group keys and multicast messages, that our checking of the *parts* literals would quickly slow down the search process. We model multicast messages as $hello(T)$, where T is the counter value when the message was sent, thus ensuring all (honestly sent) messages are unique.

Having chosen to use a counter-based model, our heuristic for eager elimination of *parts* literals (see §7.3) was now no use. However, an analogous heuristic using an occurs check immediately suggests itself: if the counter variable occurs in term X in a literal $ingroup(X,Y,Z) = true$, then the clause is redundant, since this would require an agent at some point in the past to join the group and obtain a group key or individual key that is only available now. A similar check can be applied to $member(X,Y) = true$ literals. The implementation of this heuristic required only a minor adjustment from the existing *parts* heuristic, and results in the elimination of a lot of unreachable states from the search. This rule could be generally applied to backward searching tools using a tick based model.

As an illustration, in Figure 10.1, we give the clauses required for modelling the sub-protocol for the sending of multicast messages. Note that we record the composition of the group at each point in time in the fourth argument of the *send* constructor. This is important for making conjectures about security properties later on. Note also that *ingroup* is an arity 3 function, with the third argument returning the a list of group members without the agent named in the first argument. This is used when agents leave the group or update their keys, as in the third clause in Figure 10.1. A further point to note is that we still infer state information about principals from the trace, for example to decide if they should be expecting a key update message in the third clause. Our new model is something of a hybrid between a Paulson style trace model and a state-based model like that used, for example, in [Basin et al., 2003]. We would like to experiment further with more state-based model using CORAL's backwards search (see §12.1).

```
%% SEND a message
m(Trace,Group,Keysequence,Tick)=true ∧
ingroup(triple(principal(Mi),Ikey,key(Sq)),Group,Newgp)=true
→ m(cons(sent(Mi,server,encr(send(Sq),Ikey),Group),Trace),Group,Keysequence,s(Tick))=true
%% server gives key
m(Trace,Group,Keysequence,Tick)=true ∧
ingroup(triple(principal(Mi),Ikey,Oldk),Group,Newgp)=true ∧
member(sent(X,server,encr(send(Sq),Ikey),Tgroup),Trace)=true
→ m(cons(sent(server,Mi,encr(pair(key(Keysequence),send(Sq)),Ikey),group),Trace),
   group,Keysequence,s(Tick))=true
%% agent broadcasts his message, updates his key
m(Trace,Group,Keysequence,Tick)=true ∧
ingroup(triple(principal(Mi),Ikey,Oldk),Group,Newgp)=true ∧
member(sent(X,Mi,encr(pair(key(Xk),send(Sq)),Ikey),Tg1),Trace)=true ∧
member(sent(Mi,server,encr(send(Sq),Ikey),Tg2),Trace)=true
→m(cons(sent(Mi,all,encr(hello(s(Tick)),key(Xk)), cons(triple(principal(Mi),Ikey,key(Xk)),Newgp)
   ),Trace),cons(triple(principal(Mi),Ikey,key(Xk)),Newgp),Keysequence,s(Tick))=true
```

Figure 10.1: Clauses for modelling the 'send' sub-protocol

10.3 Attacking the Protocol

In [Pereira and Quisquater, 2003], Pereira and Quisquater attempt to lay down a list of desirable security properties for group protocols. They define *implicit key authentication*, that an outsider cannot learn the group key; two ¤avours of *perfect forward secrecy*, i.e. that the compromise of long term keys does not compromise past session keys; and *resistance to known-key attacks*, i.e. that compromise of session keys does lead the loss of future session keys. However, the properties Taghdiri and Jackson found not to be satisfied by the original protocol design fall outside of this categorisation. Essentially, this seems to be because, unlike the protocols in the Pereira–Quisquater case study, we are dealing with a protocol involving a trusted key server. This means that there are no key establishment sessions involving all the principals in

the group. Since principals aren't party to decisions the server makes about updating the key to account for new group membership, this leaves them open to different kinds of attack.

The property vital to a multicast key management protocol is that throughout the evolution of the group, agents currently outside the group should not be accepted as group members by the agents inside the group. We could perhaps call this *group multicast authenticity*. This property has two ¤avours: the first, which Taghdiri and Jackson call 'outsider can't read', implies that no agent outside the group should be able to read a message sent by a member of the group. The second, which they call 'outsider can't send', implies that members of the group should not accept as valid a message sent from outside the group. We posed the former property as a conjecture to CORAL in this form:

```
% A trace ending with an honest agent broadcasting under Mk
m(cons(sent(Mj,all,encr(hello(Y),Mk),Xgroup),
   cons(sent(X,Mj,encr(pair(Mk,send(Sq2)),Ikey),Xgroup),
   cons(sent(Mj,server,encr(send(Sq2),Ikey),Xgroup),
   Trace))),Group,Keyseq,Tick)=true ∧
eqagent(Mj,spy)=false ∧

% But Mk is known to the spy
in(Mk,analz(Trace)=true ∧

% and the spy is not legitimately in the group
ingroup(triple(principal(spy),X3,X2),Xgroup,Newgp2)=false
→
```

This conjecture is negative, i.e. it states there should be no trace *Trace* ending with the 3 messages specified in the first literal, with the spy outside the group, and with the message $hello(y)$ being sent under a key the spy knows ($analz(X)$ is the set of terms the spy can learn from a trace X). The three final messages had to be specified together because otherwise CORAL finds a rather trivial attack where the spy leaves the group between the server sending a key update out to M_j and M_j broadcasting his message. Then he can read the message quite legitimately, since he was in the group when it was sent. Given the above form of conjecture, CORAL gives the following counterexample:

1.	spy	$\rightarrow$	server	:	$\{\|spy\|\}_{longtermK(spy)}$
2.	server	$\rightarrow$	spy	:	$\{\|ik(1),K(1)\|\}_{longtermK(spy)}$
3.	a	$\rightarrow$	server	:	$\{\|a\|\}_{longtermK(a)}$
4.	server	$\rightarrow$	a	:	$\{\|ik(3),K(2)\|\}_{longtermK(a)}$
5.	spy	$\rightarrow$	server	:	$\{\|send(1)\|\}_{ik(1)}$
6.	server	$\rightarrow$	spy	:	$\{\|K(2),send(1)\|\}_{ik(1)}$
7.	a	$\rightarrow$	server	:	$\{\|send(2)\|\}_{ik(3)}$
8.	server	$\rightarrow$	a	:	$\{\|K(2),send(2)\|\}_{ik(3)}$
9.	a	$\rightarrow$	all	:	$\{\|hello(9)\|\}_{K(2)}$
10.	spy	$\rightarrow$	server	:	$\{\|leave\|\}_{ik(1)}$
11.	server	$\rightarrow$	spy	:	$\{\|ackleave\|\}_{ik(1)}$
12.	a	$\rightarrow$	server	:	$\{\|send(2)\|\}_{ik(3)}$
13.	spy	$\rightarrow$	a	:	$\{\|K(2),send(2)\|\}_{ik(3)}$
14.	a	$\rightarrow$	all	:	$\{\|hello(14)\|\}_{K(2)}$

This is an attack on the protocol which hinges on the spy sending a replayed key update message in message 13. Since in general the key may or may not have changed since she last saw it, agent *a* will accept the same key again. The problem is that there is no freshness information sent in the request for a key, just the sequence number of the key an agent currently holds. Enclosing a nonce inside the package sent to the server requesting a key update would blunt this attack. Having discovered this attack, we realised that there should be a similar one whereby a spy can send a message from outside the group and have it accepted by an agent inside the group. To confirm this, we gave CORAL the following conjecture:

% A trace ending with someone accepting a message from the spy
m(cons(sent(X,Mj,encr(key(Mk),ik(Ikey)),Xgroup),cons(
sent(Mj,server,encr(read,ik(Ikey)),Xgroup),cons(
sent(spy,all,encr(hello(Y),key(Mk)),Group),trace))),Group,Keyseq,Tick)=true
$\wedge$
eqagent(Mj,spy)=false

% but the spy is not in the group
ingroup(triple(principal(spy),X3,X2),Xgroup,Newgp)=false

Given this conjecture, CORAL finds the following counterexample:

1.	a	$\rightarrow$	server	:	$\{\!|\, a \,|\!\}\ ongtermK(a)$
2.	server	$\rightarrow$	a	:	$\{\!|\, ik(1), K(1) \,|\!\}\ ongtermK(a)$
3.	spy	$\rightarrow$	server	:	$\{\!|\, spy \,|\!\}\ ongtermK(spy)$
4.	server	$\rightarrow$	spy	:	$\{\!|\, ik(3), K(2) \,|\!\}\ ongtermK(spy)$
5.	spy	$\rightarrow$	server	:	$\{\!|\, read \,|\!\}\ k(3)$
6.	server	$\rightarrow$	spy	:	$\{\!|\, K(2) \,|\!\}\ k(3)$
7.	a	$\rightarrow$	server	:	$\{\!|\, read \,|\!\}\ k(1)$
8.	server	$\rightarrow$	a	:	$\{\!|\, K(2) \,|\!\}\ k(1)$
9.	spy	$\rightarrow$	server	:	$\{\!|\, leave \,|\!\}\ k(3)$
10.	server	$\rightarrow$	spy	:	$\{\!|\, ackleave \,|\!\}\ k(3)$
11.	spy	$\rightarrow$	all	:	$\{\!|\, hello(12) \,|\!\}_K\ (2)$
12.	a	$\rightarrow$	server	:	$\{\!|\, read \,|\!\}\ k(1)$
13.	spy	$\rightarrow$	a	:	$\{\!|\, K(2) \,|\!\}\ k(1)$

This attack is similar to the first one, also relying on the spy replaying an old key update message, but this time in response to a read request. This attack could also be prevented by enclosing freshness information in key updates. We propose to do this using nonces in the improved protocol below.

10.4 An Improved Version of the Protocol

Sending a message

1. $M_i \rightarrow S \quad : \quad \{\!|\text{send}, \boxed{N_{M_i}}\,|\!\}_{Ik_{M_i}}$
2. $S \rightarrow M_i \quad : \quad \{\!|\, \boxed{N_{M_i}}\, , Gk(n')|\!\}_{Ik_{M_i}}$
3. $M_i \rightarrow ALL \quad : \quad \{\!|\text{message}|\!\}_{Gk(n')}$

Receiving a message

1. $M_j \rightarrow S \quad : \quad \{\!|\text{read}, \boxed{N_{M_j}}\,|\!\}_{Ik_{M_j}}$
2. $S \rightarrow M_j \quad : \quad \{\!|\, Gk(n'), \boxed{N_{M_j}}\,|\!\}_{Ik_{M_j}}$

The boxes highlight the changes to the protocol. The idea is that each time an agent wants to update his key, he includes a fresh nonce in his request. This is returned to

him along with an update key. So, although the message may be delayed, the honest agent knows that the key was issued *after* he made his original request, which is the best we can hope for in this protocol scenario.

10.5 Summary

The attacks described above are serious and do not even require the spy to have full Dolev-Yao capabilities - he need only be able to replay an old message, he does not need to stop a message from being received or break messages apart etc. As such, it can hardly be argued that Taghdiri and Jackson's improved protocol was secure. After making the improvements required to secure against this attack, the protocol doesn't look like a good candidate for solving the problem of multicast key management. Almost all the original optimisations to ensure the scalability of the protocol have had to be removed for security reasons, and we now have to generate fresh nonces every time we want to send or read a message. There are many other protocols for this scenario, but few of them have been subject to any formal security analysis. There is much potential for future work here (see §12).

We were quite pleased with the way CORAL performed on this protocol. Firstly, the use of an inductive model meant we didn't have to make fundamental changes to our modelling strategy to accommodate an open-ended protocol with an unbounded number of agents, joins, leaves, messages sent and received etc. Secondly, modelling at the first-order Horn clause level in a theorem prover meant making the adaptations required to store and manipulate a list of current group members was just an evening's work. Thirdly, CORAL was able to discover attacks requiring a long trace of messages to be sent, indicating it has scaled up well, despite exploring a model without any pre-setting of the number of agents, joins and leaves etc.

There were two weaker aspects to CORAL's performance: one was the run times (up to 3.5 hours to find the second attack). This is not as long as was required for the Asokan–Ginzboorg protocol, but it is still a long time. The second weaker aspect was the difficulty of posing conjectures. It took several attempts to pose the security property in such a way that counterexamples really were attacks. However, we think that

having analysed this protocol, it would be much easier to now go ahead and analyse other similar protocols (see §12).

Chapter 11

Related Work

The field of automated security protocol analysis has matured considerably over the last few years. There are now dozens of rival tools and techniques. To evaluate CORAL we must therefore compare it to the best systems which are designed to achieve the same goals, i.e. to automatically detect and present attacks on protocols. Two highly regarded systems for this purpose are Athena, [Song et al., 2001], and the On-The-¤y Model Checker, [Basin et al., 2003]. The first two sections in this chapter compare CORAL to these two approaches.

A closely related approach is the Casrul system proposed by Chevalier et al. using the `daTac` theorem prover, [Chevalier and Vigneron, 2002]. We discuss the relationship in §11.3.

CORAL's advantage over other approaches is its ability to find attacks in a very ¤exible model, allowing us to, for example, attack a general model of a group protocol without predetermining the size of the group, or what roles agents will play once they join. We compare CORAL to other attempts to attack group protocols in §11.4.

The CORAL approach is closely related to Paulson's inductive method for protocol analysis, [Paulson, 1998]. We explore this relationship in §11.5. CORAL also builds on previous work on analysing protocols in first-order logic carried out by Weidenbach, [Weidenbach, 1999]. We compare CORAL to Weidenbach's work in §11.6. We brie¤y compare CORAL to the general refutation tools developed by Reif and Protzen in §11.7. There is a summary of the chapter in §11.8, including a discussion of the ultimate applicability of CORAL in §11.8.1.

11.1 Athena

Athena, [Song et al., 2001], is widely considered to be the state of the art as far as automatic protocol analysis tools are concerned. It is a dedicated tool designed from the ground up for security protocol analysis, using a combination of theorem proving and model checking techniques.

11.1.1 Description of Athena

Athena uses a model based on the strand space of Fábrega et al [Fábrega et al., 1999]. We looked at the strand space model in §2.4.5. We will recap here the definitions of *strands*, *bundles* and *strand spaces*: a strand space is a collection of *strands*, with a graph structure which expresses causal relations. A strand is a sequence of *events* that a single party may engage in. An event is the sending or reception of a message, represented by a *node* in the graph. Nodes have a sign: a positive sign indicates a message was sent, and a negative sign indicates it has been received. There are two kinds of strands in the model, those for honest participants and intruder strands. A strand belonging to an honest agent contains that agent's actions in one particular run of the protocol. If an agent is involved in several runs, each of these will have its own strand. Nodes in separate strands are adjacent when they represent the sending and receiving of the same message.

A *bundle* is a finite acyclic subgraph of the strand space that is in a certain sense backwards-closed: all received messages occurring in strands in the bundle must have come from nodes also in the bundle, and if an event on a strand is in the bundle, then all preceding events on that strand must also be in the bundle.

Athena additionally uses the notions of *semi-bundles* and *goal-bindings*. A *semi-bundle* is like a bundle but closed only under backwards tracing of strands, not under sending of received messages. *Goal-bindings* are used to keep track of the search for ways a message might have originated, from another honest agent or from an intruder. In other words, goal-bindings record ways in which a semi-bundle could be expanded towards becoming a bundle. These concepts are used to remove some redundancy from the protocol trace representation. The idea is to eliminate search space explosion

caused by infinite forwarding of a message, where arbitrarily many intruder strands may receive and pass on the same message. CORAL uses the step compression heuristic, described in §7.2.1, to achieve the same effect. However, by the same technique, Athena can additionally remove the redundancy caused by duplicate representations of what are the effectively the same protocol runs where only the order of interleaving actions has changed. CORAL does a similar kind of pruning by subsumption checking (see §8.2.1). However, it is not as efficient as the representation-based approach Athena uses. In the strand space model, a single representation captures the exchange up to a certain amount of permutation of messages. For example, take the attack Paulson found on the simplified Otway-Rees protocol, §2.3.5 (p. 17). There is also an attack which is identical except that the first two messages are swapped round, since the spy requires no information from the honest agent's message 1 to send a faked message 1'. In the strand space model, both these traces would be represented by the same bundle. Furthermore, this applies not just to the final result, but to the intermediate stages of the search for an attack or a security proof. So Athena's representation is more efficient. CORAL's representation is simpler, and subsumption checking is not always sufficient to spot when two traces are effectively identical.

Athena has a dedicated logic for expressing security properties in terms of strands and bundles. Properties are expressed as sequents of the form

$$P;\Gamma \vdash \Delta \tag{11.1}$$

where Γ and Δ are sets of strands under the protocol P. The semantics of the sequent are that for any bundle C representing a run of the protocol P, if $\Gamma \subseteq C$ then $\Delta \cap C \neq \emptyset$.

In order to prove such a property, Athena tries to show that all possible bundles containing Γ contain at least one strand from Δ. Proof search is carried out by a dedicated inference algorithm. The first step is to convert the sequent to a form that also represents the *state*. Informally, this includes information about the set of possible semi-bundles under consideration and their possible goal-bindings, i.e. ways they could be extended to bundles. If the state contains a strand in Δ, then the property is proved. Otherwise, a split rule is applied, and a *next state function* calculates the possible different ways of extending the semi-bundle. Each of these ways generates a new sequent and state, and each of these sequents must be proved. If there are no unbound goals in

a sequent, that is it cannot be extended by further strands, but the state does not contain a strand in Δ, then the security property is false. In this situation, the state information contains the counterexample or attack.

Athena contains some additional optimisations to improve efficiency, such as pruning theorems. These prevent Athena from wasting time considering contradictory states. For example, if a state requires that an intruder be given a secret key k, but that key is never sent in any protocol traffic, then the state is contradictory. CORAL also includes a heuristic that prunes out these kinds of states (see §7.2.2). However, Athena's pruning mechanism is generic, and users can easily add pruning theorems of their own. In CORAL, they are currently 'hard wired' in the modifications to SPASS. However, the Comon-Nieuwenhuis method allows for user-defined lemmas to be used to prune out redundant states, so it is possible to to add lemmas to CORAL as well. The only restriction is that, because the answer literals in CORAL containing the trace are hidden from the general proving mechanism, lemmas cannot currently refer directly to properties of the trace represented by a particular clause. It would be possible to modify CORAL to accept these kinds of lemmas though, (see §12.1).

By using a dedicated representation, logic and inference algorithm, Athena achieves considerable improvements over previous tools in terms of efficiency. However, using a new and hence untrusted program to provide proofs of security is risky. For example, an attempt to implement the Athena algorithm in the SyMP model prover revealed a number of implementation considerations that are not described in the original Athena paper, [Berezin and Groce, 2001]. If steps are not taken to treat bound variables in a particular way not described by Athena's authors, false attacks may be produced. This erodes confidence in Athena's proofs of security.

11.1.2 Results

Athena's results on what has become the standard corpus of 2 and 3 party protocol examples, the Clark-Jacob library, [Clark and Jacob, 1997], are impressive. 30 protocols from the library have been checked by Athena, with a security property taking on average 0.16 seconds to check. CORAL is much slower, taking for example 37 seconds to discover the well known attack on the Needham-Schroeder protocol (see §8).

Athena's authors do not mention explicitly the rediscovery of the attack on the simplified Otway-Rees protocol, [Mao and Boyd, 1993], which CORAL has rediscovered, but there seems to be no reason why Athena should not have discovered this. This attack is significant because it requires an honest player to play two parallel roles, and generate two nonces. This was something which older formal models for protocols would not allow, but newer models such as the inductive model used by Paulson and in CORAL, and the strand space model, do allow.

It does not seem, however, that Athena has been used to discover any new attacks on known protocols. The only new attacks reported are on a set of 1641 artificially generated protocols produced by an automatic protocol generator written by the Athena authors. CORAL, however, has discovered five new attacks on two serious protocols proposed new applications (see Chapter 9 and Chapter 10).

Athena has not been used to analyse any group protocols. Indeed, the only mention of the use of the strand space model to analyse group protocols seems to be in [Millen and Shmatikov, 2003]. However, the method sketched here requires the analyst to choose in advance how many principals will be involved in the group. As we saw in Chapter 9, and is further shown in [Pereira and Quisquater, 2001] where attacks require a group to be of size at least 4, this choice is critical, as it can affect whether or not attacks are detected. CORAL can model group protocols in a general way, allowing attacks on different sized groups to be found, as is evidenced in Chapter 9.

It seems that any method for group protocol analysis in Athena would require the group size to be pre-determined. This is because of the semi-bundle construct. Athena requires that semi-bundles are finite and closed under backwards traversal along strands. This is not possible in a general model of the Asokan–Ginzboorg protocol attacked by CORAL in Chapter 9. If a bundle contains a node modelling the sending of a general message 3, this must occur in a strand for the group leader role. This strand is infinite, as there may be an unbounded number of different, preceding message 3s in the strand, one for each member of the group, and we must in general consider the group to be of unbounded size. To work around this would require some non-trivial modifications to the strand space model and/or the notions used in Athena. Perhaps a set of message 3s could be considered to be sent simultaneously, represented

in some generic way by a sort of supernode, then only expanded later in the proof search when it became relevant. Some thought would be required to make this adaptation successfully. Additionally, in our second case study in §10, CORAL analysed a protocol where the same agents may take part in an unbounded number of runs of different sub-protocols, one after the other. Each time they must preserve values from one sub-protocol (such as keys distributed by the server) to use in the next sub-protocol (for example to leave the group). This would also produce infinite semi-bundles, which cannot be handled by the Athena tool.

Athena is also tied to the notion of the Dolev-Yao attacker. For wireless protocols like the one considered in Chapter 9, we model a weaker intruder who cannot block messages from being received. To do this in Athena would require some reprogramming, to force semi-bundles to be constructed in such a way that messages sent on honest strands must reach their recipient, and *before* any messages from intruder stands using terms contained in the honest message.

11.1.3 Summary

In summary, Athena gains efficiency from dedicated representation and inference algorithms, and from its generic interface for pruning theorems. It has been applied to more of the standard protocols than CORAL, and can check these protocols for attacks extremely fast. It can also prove protocols to be correct, which CORAL as yet does not, though it has the theoretical ability to do so (see §12.4). However, the use of a new dedicated algorithm and implementation introduces an element of required faith in its results that one might not consider to be so great for e.g. a proof in and LCF prover like Isabelle. This is of more significance when Athena concludes a protocol is secure, of course. In the case of attacks, one can always write the attack down and then debate its validity. Athena has not yet been used to discover any significant new attacks though.

The representation used in Athena improves on trace based models such as that used in CORAL in terms of the clever way it avoids considering many states which are equivalent to states already seen. However, along with the greater efficiency comes less ¤exibility, as is evidenced by the difficulties in representing group protocols like the Asokan–Ginzboorg protocol in Athena's model, and particularly protocols comprising

a suite of sub-protocols like the Tanaka–Sato protocol. This seems to be an unavoidable trade-off.

11.2 On-The-Fly Model Checker

Basin, Mödersheim and Viganò have written an on-the-fly model checker (OFMC) for searching for security protocol attacks, [Basin et al., 2003]. Its results have also been impressive.

11.2.1 Description of the OFMC

The OFMC features two key innovations over the ‘standard’ model checking approaches, e.g. Lowe’s, [Lowe, 1996]. The first is *lazy data-types*. A lazy data type is one in which constructors such as `cons` build data types without evaluating their arguments. This allows protocol models to be formalised as infinite trees, with the tree only being computed on-the-fly as the states are explored. This presents a clean way of handling the infinite state space that occurs in a security protocol model. Additionally, the OFMC employs a *lazy intruder model*, whereby instead of generating all the intruder’s knowledge at each point in the space of possible traces, the intruder’s knowledge is represented in a symbolic fashion. At each point in the search space, the OFMC checks to see if the intruder can generate a message that an honest player expects.

The OFMC requires as input a parameterised version of the protocol under analysis. That is, the user must choose a scenario for investigation, which must specify the exact runs of the protocol to be considered. For example, for the NSPK protocol, the user would specify a run between an agent Alice and the spy, and between Alice and Bob in order to discover the attack. This cuts down the search space considerably. Input can be given to the OFMC in the same high-level language as is used by the Casrul system, [Jacquemard et al., 2000].

The search procedure for the OFMC is the same as for most model checking approaches, i.e. to start with an empty trace and search forwards to generate the set of reachable states. An attack is defined by a set of *goal states*, i.e. states in which secu-

rity has been violated in some way. The OFMC compares each generated state with the goal states to see if an attack has been found. Both secrecy and authenticity properties can be specified.

The OFMC's primary aim is to discover attacks. It is a semi-decision procedure for finding attacks on faulty protocols, as CORAL is at present. Additionally, a limit to the number of sessions can be used to guarantee termination and give some indication of protocol security.

11.2.2 Results

The OFMC has been applied to 33 ¤awed protocols from the Clark-Jacob corpus, [Clark and Jacob, 1997], and has found attacks on 32 of them, including a previously unknown ¤aw. For each ¤awed protocol, a ¤aw is found in under 4 seconds. However, for all these timings, the OFMC was given exactly the right scenario for the attack. The OFMC can also find the attack on the simplified Otway-Rees protocol, which as we explained in §11.1.2 above, is an important test example from outside the standard corpus, [Compagna, 2002]. Perhaps more impressively, the OFMC has also been used to find a previously unknown ¤aw in a protocol proposed by Siemens for multimedia communication in mobile devices (H.530). This ¤aw was quite a serious one, and Siemens have now changed the protocol to protect against it, [Basin et al., 2003].

It seems that the OFMC has not been used to analyse any group protocols yet. It should be able to so this so long as the number of agents in the group is fixed in advance. Modelling a general-sized group protocol in the OFMC does not seem to be so straightforward. The problem lies in the fact that, although the intruder knowledge is modelled symbolically, the state transitions still have the interpretation of one message being sent, or the intruder carrying out one action, albeit with variables representing some of the terms in the message or action. In order to model the Asokan–Ginzboorg protocol in a general way (see Chapter 9), CORAL had to use variables to represent arbitrary numbers of message 2s and 3s, with these variables later being instantiated as required. There does not seem to be a way to do this in the model used by the OFMC, although it could perhaps be adapted to do so.

The need for an explicit scenario would cause problems when trying to model a

protocol like the one we analysed in CORAL in Chapter 10. In this multicast protocol, we not only have an unbounded number of agents to consider, but additionally each agent may play any number of roles, i.e. he may join the group, send messages, receive messages, and leave the group in any valid combination, any number of times. Different kinds of interaction are required to effect the attacks we found – for example for the first attack, an agent must send a message in order to generate the message which the spy can later replay, and in the second attack, an honest agent must receive a message in order to generate the message the spy will later replay. It would be hard to guess the right scenario in terms of roles without already knowing the attack.

11.2.3 Summary

The OFMC finds attacks on standard protocols much faster than CORAL, for similar reasons to those behind Athena's speed: the representation is more tailored to the problem, and the inference algorithms, particularly the lazy generation of intruder knowledge, are designed specifically for the job. CORAL also generates intruder knowledge in a lazy way, although as it is performing backwards search, it is really doing knowledge checking rather than knowledge generation. However, CORAL's intruder knowledge is handled by axioms in the model which are then repeatedly applied by the prover to compute messages the intruder would have to intercept. The intruder knowledge handling in the OFMC is hard wired into the model checker, making it much faster.

The OFMC, like Athena, does not seem to have the flexibility to attack a general model of a group protocol. Instead, one would have to choose the number of players in advance. It does not seem to be able to model a protocol where the same agent may take part in an arbitrary number of sub-protocols, because it requires the scenario to be fixed in advance.

11.3 The Casrul System

Chevalier et. al proposed a first-order model which has some similarities to the one we use in CORAL, [Chevalier and Vigneron, 2002]. Their main innovation is the use

of puppet principals, i.e. dummy principals who can be sent messages by the intruder in order to act as oracles. These oracles may give the intruder the terms he needs to generate attacks. The real principals are restricted to a particular scenario, as they are for the OFMC described above, and the dummy principals are set to always generate the same nonces. These two restrictions ensure the system terminates. The Casrul system also uses a lazy intruder model, like the OFMC, whereby instead of generating all the terms the intruder knows at a certain point, the system just checks to see if the intruder can generate a particular message that an honest player (or a puppet principal) expects.

CORAL's model is similar in that the heuristic described in §7.2.3 also stops the intruder from considering terms which don't match protocol messages, but because Casrul model is bound to particular scenario, this can be used to restrict further the messages the intruder might generate, i.e. Casrul will only generate those that fit the scenario, and not just those which match the pattern of the protocol. The Casrul model also takes advantage of the associative and commutative (AC) unification supported in the theorem prover `daTac`, [L.Vigneron, 1996]. This facilitates the definition of an AC operator for combining elements in a set, which is used to represent the set of all sent messages, and the set of all states of principals. Now AC unification can be used to allow the rules for state transitions to be applied to any previously sent message, and any principal. Casrul gains a significant efficiency advantage over CORAL here. We have to explicitly define a *member* function for extracting candidates from a list of previously sent messages. Each time a member literal is satisfied, many inference rules may have to be applied. This is particularly laborious if we are checking for non-membership, i.e. satisfying a $member(X, Trace) = false$ literal, when we have to check every message in the trace. The use of lists does mean that CORAL's model retains the ability to reason about the order in which messages were sent, but this could perhaps be recovered in Casrul if necessary, using a counter or similar mechanism.

Casrul's run times on the standard protocols from [Clark and Jacob, 1997] are significantly slower than those for the OFMC or Athena, and are broadly comparable with CORAL. For example, for the Otway-Rees protocol, times are almost identical; for the Hwang-Chen Splice protocol, CORAL is significantly faster; and for the Andrew RPC

protocol, Casrul is significantly faster (though it is not clear which attack the timing is for). Both systems were tested on Pentium Linux boxes of similar vintage (though exact hardware details are not given in [Chevalier and Vigneron, 2002]). Bearing in mind that the Casrul system must have the scenario for the attack specified before the search starts, CORAL compares favourably here. The Casrul system typically requires the consideration of far fewer states than CORAL, which would seem to give it better potential for scaling up. However, the Casrul system as it stands is bound to the notion of the Dolev-Yao intruder and to the analysis of specific scenarios, which would hinder adequate formalisation of the protocols we used as case studies in Chapters 9 and 10. Additionally, it doesn't seem possible to model an arbitrarily-sized group protocol in the Casrul system, since a rule to send arbitrary numbers of different messages at the same instant would not be first-order in the model they use.

Casrul has been used to discover one new attack on the Denning-Sacco symmetric key protocol, [Denning and Sacco, 1982], though some may consider it slightly dubious. It requires an extreme case of type confusion, where an agent mistakes a package of the form $T, \{\!| BK_{AB}T |\!\}_{K_A}$ for a timestamp T. Since the package itself begins with a timestamp, this might seem unlikely, but the authors do give particular implementation scenarios where it could possibly occur, such as if the recipient of the package reads the timestamp and then disregards the rest of the message.

In summary, Casrul has some efficiency advantages over CORAL in terms of the number of states explored thanks mainly to its use of a theorem prover supporting AC unification and a model which is restricted to a specific scenario. However, run times are about the same as CORAL and the Casrul model is not as expressive. It has discovered a single new attack on a standard protocol, but has not been used to model any group protocols.

11.4 Other Work on Group Protocols

Various other attempts have been made to analyse group protocols. The most successful in terms of finding new attacks was [Pereira and Quisquater, 2003], where the case study was from the CLIQUES protocol suite, [Ateniese et al., 2000]. Pereira and

Quisquater discovered a number of new attacks, using a pen-and-paper approach and borrowing some ideas from the strand space model. Their attacks were quite subtle, involving properties of the Diffie-Hellman exponentiation operation widely used in the CLIQUES suite. They also involved the spy doing some quite imaginative things, like joining the group, leaving, and then forcing the remaining members to accept a compromised key. This showed the value of by-hand analysis taking algebraic properties of cryptographic functions into account, but only when undertaken by experts.

Meadows made an attempt to extend the NRL protocol analysis tool (NPA), [Meadows, 1996b], to make it suitable for analysing group protocols, [Meadows, 2000a]. Again the CLIQUES protocols were used as an example. However, NPA was not able to rediscover the attacks Pereira and Quisquater had discovered, because of the intricate series of actions the spy has to perform to effect the attack. NPA is tied to quite constrained notions of secrecy and authenticity, which may be where the problem lay.

It would be interesting to see whether these kinds of attacks could be found automatically by CORAL. We hope that the very flexible inductive model used might mean that these attacks are within CORAL's scope. However, some work would be required to model the associative and commutative properties of the exponentiation operation used. We are at an advantage here because of our access to equational reasoning in SPASS– we already use this to model the commutative properties of symmetric keys (see §8.3). Modelling these properties is a topic which has recently started to attract more research interest, [Millen and Shmatikov, 2003, Bertolotti et al., 2003]

As we mentioned at the beginning of Chapter 10, the Pull-Based Asynchronous Rekeying Framework of Tanaka and Sato, [Tanaka and Sato, 2001], has previously been modelled by Taghdiri and Jackson, [Taghdiri and Jackson, 2003]. The Taghdiri-Jackson model is general in terms of the number of members of the group, but does not model a malicious intruder trying to break the protocol. Instead, they just investigate correctness properties of the protocol, using a specification language called Alloy and a verification tool for this language based on a SAT-solver. One correctness property is found not to hold, revealing a ¤aw whereby a member of the group may accept a message from an ex-member of the group as being current. Taghdiri and Jackson proposed

a fix for the protocol, and could find no more attacks, but by modelling the protocol in CORAL (see Chapter 10) we were able to discover two new attacks, both as serious as the one found by Taghdiri and Jackson.

11.5 Paulson's Inductive Approach

The CORAL system was inspired by the idea of a counterexample finder for Paulson's inductive model. Paulson's formalism for security protocol analysis, [Paulson, 1998], is mechanised in the interactive theorem prover Isabelle/HOL, [Paulson, 1989]. To formalise a protocol, an inductive datatype is defined using rules that define how agents might add messages to the trace. Additionally, further rules establish what the knowledge the spy can extract from previous network traffic, and how the spy might add fake messages. The rules we use in our formalism are very similar (§6.5 and §6.6). In particular, the same *synth* and *analz* operators are used in our model. Paulson's model is unconstrained in terms of the number of agents who might possibly take part in a protocol, the number of nonces each agent might generate, the number of interleaving sessions, etc. CORAL's is similarly unbounded. The spy is not accepted as an honest agent in Paulson's work, but instead a subset of the agents are considered 'bad', that is the spy has their long term keys. This has since been proved to be equivalent to having the spy alone accepted as an agent by the other players, [Syverson et al., 2000]. So here CORAL's model is also equivalent. The security properties Paulson attempts to prove are very similar to the conjectures CORAL refutes.

The main difference between the formalisms is that while CORAL's is first-order, Paulson's is in higher-order logic. Using higher-order logic makes it easy to create a strongly typed model. However, in our model, we were able to use unary functions *number*, *agent* to specify sorts, and then modifiers like *key* and *nonce* to distinguish between message objects. In this way we were also able to create a typed formalism. Of course, we cannot recover the full power of higher order logic, but this does not seem to be necessary to specify and analyse most protocols. This is perhaps not so surprising, since a protocol model has conceptually simple semantics. The main problems with protocol analysis occur not in trying to specify the model, but from the combina-

torial blow up caused by the agent's infinite knowledge, the infinite number of agents etc.

In CORAL we have specified a number of protocols that Paulson has also investigated, and found the relevant attacks. Additionally we have found attacks on a group protocol. One of the unusual features of Paulson's work is that his approach has been used to prove properties of a (different) group protocol, [Paulson, 1997]. So, we have gone some way to make a convincing case for CORAL as a complementary counterexample finder for the Paulson approach. CORAL's long run-times are perhaps not so much of an issue in this context, as the process of finding a proof in the Isabelle/HOL prover is a fairly long one even for an expert user. CORAL could run on a parallel machine attempting to disprove the conjecture the user is trying to prove.

We tried giving CORAL 'failure information' from the Paulson proofs. For a faulty protocol, we gave CORAL hints, in the form of extra *member* specifications, from the final subgoal where Paulson's proof broke down. These false subgoals typically specify that several different messages are in a trace, and that this implies something false, such as that the key remains secure. However, giving this information to CORAL in fact made run-times much longer. This is perhaps because, even though these failed subgoals are far from full specifications for an attack, they are still *too much* information to give to CORAL. If CORAL is given a conjecture with many *member* literals in it, it tends to slow it down, since CORAL can arrange for the required material to be in the trace in many different orders, and then has to work out which of these might lead to a valid trace. The conjecture we used for looking for disruption attacks on the Asokan–Ginzboorg protocol is a good example of this (see §9.4.1). When Paulson looks at such a failed subgoal, he uses human common sense to help work out what a likely guess at what order the messages must be in an attack trace, for example that a message 3 is probably after a message 1. CORAL currently doesn't do this kind of reasoning, but we could perhaps define some rules like this to act as heuristics to assist CORAL in choosing which clause to consider next.

An important feature of Paulson's model that we wanted to preserve was its easy adaptability to non-standard protocols, as we discussed in §6.10. We believe that Chapters and 9 and 10 provide some good evidence that we have succeeded. The Asokan–

Ginzboorg protocol in Chapter 9 required us to model compound keys, a weaker intruder, and an arbitrary number of agents. A couple of extra Horn clauses were needed to allow the spy to manipulate compound keys. Our trace based model allowed us to make conjectures about attacks in a wireless network without difficulty. The trace based inductive model allowed us to model the addition of an arbitrary number of different messages to the trace in one instant, which was the vital feature for the modelling of the protocol without pre-setting the size of the group. For the Tanaka–Sato protocol in Chapter 10, we had to add knowledge about the composition of the group to the trace, and also to store the current group key at each point. This was very simple. Our model needed no adaptations to account for the fact that agents may take part in any valid combination of runs of the join, leave, send and receive sub-protocols. All of these features are a real challenge to rival attack finding systems.

Cohen's TAPS system, [Cohen, 2000], is related to Paulson's in that the proving of security invariants is somewhat similar to proving security properties by induction. It has the advantage over Paulson's technique that proofs are often completely automated. Cohen's method also provides no support for finding counterexamples. CORAL could perhaps help out here too. The proof obligations generated by TAPS are currently discharged by a resolution theorem prover, which suggests that it should be possible to adapt CORAL to use this model.

11.6 Weidenbach's First-Order Formalism

Weidenbach formalised the security protocol problem in a simple first order model, suitable for the SPASS theorem prover, [Weidenbach, 1999]. His stated aim was to combine the benefits of Paulson's inductive model with the push-button nature of model checking approaches. However, the simplifications he had to make to get the very fast SPASS proofs were too drastic. They would prevent many attacks from being discovered, and so give false guarantees of security. For example, the attack on the simplified Otway-Rees protocol, which CORAL rediscovered (§8.5), and even the very simple attack on the Clark-Jacob protocol, which CORAL has also rediscovered (§8.3), are beyond the scope of Weidenbach's model. This is because the two agents

in his model are restricted to playing only one part in the protocol. One is always the initiator, and the other always the responder. Additionally, there are only two nonces available for use, and the spy is not accepted as an honest agent. For the Otway-Rees attack, we need three nonces, and a spy who can pass himself off as an honest player. In fact, the attack on the Neuman-Stubblebine protocol, which Weidenbach used as a case study, is one of the few in the literature that his approach would have been able to find. We showed, for the sake of completeness, that CORAL can also find this attack in §8.4.

11.7 General Inductive Refutation Tools

In §3.2, we introduced two tools for finding counterexamples to false inductive conjectures, those of Protzen, [Protzen, 1992], and Reif, [Reif et al., 2001]. Though Reif's more recent tool is more powerful than Protzen's, neither would be adequate for finding security protocol attacks. This is because their approaches are designed for datatypes that are easy to enumerate. To generate all valid protocol traces is a non-trivial task, taking into account the calculation of possible faked messages from the spy. Both tools would tend to generate a lot of invalid traces, and get stuck very quickly.

In §5.3, we reported on experiments with CORAL on the kinds of simple non-theorems these systems were designed for. CORAL's performance was good. Although it is hard to make valid comparison with Protzen's work, given how much faster computer hardware has become in the last 10 years, Reif's experiments were carried out quite recently. He reported run times of several seconds on his hardest problem, but CORAL was able to find the counterexample in under a second. The reason for this is that Reif's system, and Protzen's, were designed to be subsystems of larger provers that would tackle the refutation of false conjectures in the context of a large inductive proof. In CORAL, we bring all the power of a full first-order theorem prover to bear on the problem of finding counterexamples. Given that computer hardware costs are relatively low, it would not be unrealistic to have CORAL running on a parallel machine doing refutation tasks as part of a larger proof effort. This would seem to be a promising approach for assisting an automated inductive theorem prover. We discuss

this in §12.5.

11.8 Summary

We can summarise our comparison of CORAL to related work in a few key points:

- Purpose built protocol analysis tools like Athena and OFMC have a huge speed advantage over CORAL, due to the fact that CORAL is essentially a modified first-order theorem prover, using general inference rules and algorithms. However, we also model protocols under fewer assumptions about the number of agents and what roles they will play.

- No other automated tools have been able to find attacks by an active intruder on a group protocols, as CORAL has. The nearest efforts have been the Alloy model used by Taghdiri and Jackson, which was general with respect to group size and found a rather obvious flaw, but did not include a malicious intruder. When we modelled this protocol in CORAL, with an active intruder, we found two new attacks. CORAL's success in this area is a product of the very general inductive model used. There seems to be an inevitable trade-off in designing protocol analysis tool between flexibility and speed.

- CORAL's model is a satisfactory first-order version of Paulson's higher order model in that it maintains the essential expressiveness in terms of numbers of agents, nonces etc. It also maintains the simplicity and level of abstraction that allows the model to be rapidly adapted to slightly different protocol scenarios.This is demonstrated by CORAL's modelling of a group key establishment protocol and a multicast key management protocol. CORAL would be a useful tool to run in parallel to an attempt to prove a protocol secure using Paulson's methods.

- CORAL builds and improves on Weidenbach's first-order protocol model, as it provides the intruder and also the honest agents with much more realistic capabilities. This enabled CORAL to find attacks that Weidenbach's model would

not allow. The price is that CORAL only finds attacks, not proofs of security at present, though this could perhaps be addressed by future research (see §12.4).

- Outside of the protocol analysis field, CORAL is also a powerful tool for quickly finding counterexamples to smaller false conjectures in simpler datatypes. It improves on previous work in this area.

11.8.1 Ultimate Applicability of the Technique

The comparison with other protocol analysis approaches gives us a picture of CORAL as a niche tool, which although able to find the standard attacks on two and three party protocols, is not in its present form the most efficient tool for this task. Instead, its strength is to be found in its ¤exibility. It is suitable for analysing more unusual protocols where the parameters such as number of participants, the roles they play in the protocol and size of key may vary, or, where the goals are unusual, and may require several parts of the trace of messages to be considered rather than just some final state. Additionally, CORAL's model can be very quickly adapted to new protocols which don't fit the pattern of the well-analysed Clark-Jacob corpus. This is illustrated by the fact that CORAL has been used to find five new attacks on serious protocols from the literature. The OFMC has found two new attacks, and none have been found using Athena. This was a direct result of the fact that we could very quickly make the small adaptations to the model required to analyse protocols that OFMC and Athena could not analyse in their present forms. CORAL's applicability then is a little like the applicability of a high level language like Prolog - it can be quickly adapted to deal with a new example which differs from previously considered protocols, but will not give the kind of performance that can be obtained from optimising a tool for a set of very similar protocols.

11.8.2 The Way Ahead

The further development of CORAL would require the decision to be made as to whether we attempt to fully retain generality, so that CORAL can continue to be used

for finding counterexamples in any inductive specification, or whether we specialise CORAL to be a protocol analysis tool. We consider the possibilities in Chapter 12.

Chapter 12

Further Work

There are many possible ways in which CORAL may be developed. Here we discuss some ideas for improving CORAL's performance on security protocol problems, for making it easier to use and for other application areas where CORAL might be useful tool.

12.1 Improving CORAL as a Protocol Analysis Tool

There are a number of small tweaks we could make to CORAL that would improve its efficiency as a special-purpose protocol analysis tool. When searching for an attack in our model, CORAL often generates clauses that contain a literal stating $true \neq false$. This is of course an axiom of our theory, and so the clause is detected as redundant by subsumption. So many clauses are deleted this way that it may be faster to detect and prune them away by a simple special-purpose syntactic check. At the moment, CORAL spends a lot of time doing subsumption checking, a little under half of the time spent doing reductions. Looking for reductions and detecting redundancy constitutes about 99% of the total run time for a hard problem. So, any time we can save here would be worthwhile.

The dedicated security protocol redundancy rules, described in Chapter 7, take very little time to check, but have a big impact on the search space. This suggests that more special purpose redundancy rules would further improve CORAL's performance.

We should also like to allow the user to add his own redundancy rules in the form of lemmas. As discussed in §11.1.1, CORAL already allows the user to add lemmas, but because of the way the answer literal is hidden, they cannot currently refer to properties of the trace represented by a clause. This restricts their efficacy. It would take a little work, but it would be quite possible to alter CORAL to extend the scope of these lemmas to reasoning about the trace, and so to give them the same power they have in the Athena tool.

Another idea is to hard-wire a certain amount of the inference that is standard across all protocols, e.g. the calculation of the spy's knowledge. This could be used to more effectively prune out states which require information to be revealed that is never sent in the trace. Currently, this may take a while to do, because of having to process several *synth* rules in order to break a faked message down into all the terms required. If we had the breaking down and rebuilding of messages hard-wired in CORAL, we could check redundancy for every $\mathit{synth}(\mathit{analz}(\ldots))$ term instead of every $\mathit{analz}(\ldots)$ term, and thereby save considerable time, particularly for protocols with long messages or complex message structure. This hard-wiring of the spy's knowledge calculations could also fix the problem we had in trying to keep the ordering well founded for the Asokan–Ginzboorg protocol, as mentioned in §9.2.2. We could do some meta-reasoning about the number of players in the group, and that way avoid problems involving calculating what the intruder might learn from a message with infinite key length. CORAL still successfully found attacks using the imperfect ordering, but it would be good to recover our refutation completeness for these compound key protocols.

CORAL is at present a general counterexample finding tool with some domain-specific heuristics. This means we should be able to experiment with changing our representation for the security protocol problem. In particular, as we saw in §11.1.1, the strand space model is attractive in that is cuts out some redundancy from the search space. It should be possible represent strands and semi-bundles as inductive first-order objects, and then to reason about them in CORAL. This would provide an interesting comparison, as we would be able to see just how much of an advantage the strand space representation gives to a protocol analysis tool.

Another interesting experiment would be to try to use CORAL on the models used by the OFMC and Casrul tools (see §11.2). This would allow backwards search and forwards search in the same model to be directly compared. The main obstacle to immediately using the Casrul model in CORAL is that SPASS does not have explicit support for AC unification. We would have to add explicit equations to the formalism to model associative and commutative properties of the relevant operators. It may be easier to implement the Comon-Nieuwenhuis strategy and CORAL's heuristics, appropriately adjusted to the new model, in `daTac`.

As CORAL is built on SPASS, a theorem prover capable of equational reasoning, we should be able to find a way to reason about some simple algebraic properties of the cryptosystems underlying protocols, such as Diffie-Hellman type operations. This would allow us to analyse the second Asokan–Ginzboorg protocol, which is quite different to the first, and seems not to be susceptible to the same attacks. Also it would allow us to look at the CLIQUES protocols, which have already been shown to be faulty as we discussed in §11.4. The problem of reasoning effectively about protocols utilising Diffie-Hellman type exponentiation operations is currently being tackled by several research groups, [Millen and Shmatikov, 2003, Bertolotti et al., 2003] . Our approach would be to extend the current simple equational part of our formalism with axioms specifying commutative and associative properties of keys constructed from exponentiation.

12.2 Further Protocol Experiments

Having used CORAL to discover attacks on one multicast key management protocol, one obvious area for experimentation is to try CORAL on some other protocols designed to address the same task. There are dozens of protocols for this scenario in the literature, most of which have received little or no formal attention. Our experience so far suggests that they are likely to be vulnerable to attack.

12.3 Other Encryption Models

One weak point of almost all current protocol analysis is the perfect encryption model they use, i.e. they assume that either a principal has the correct key and can read a message, or he hasn't and so he can't. Recently, several attacks have been found on the API[1]s used by hardware modules which implement electronic payment systems. These attacks involve finding a sensitive piece of information, e.g. an account holder's PIN, by making successive guesses and using the output from the API to improve the guess, [Bond and Anderson, 2001]. The attacks are potentially very serious, but are completely outside the scope of current tools. It would interesting and exciting to try to extend and develop CORAL to find attacks like these. Several changes would be required. We would have to take into account the complexity of obtaining different pieces of information, in terms, perhaps, of average number of guesses required. One idea would be to carry this information in constraint literals, similar to the answer literals CORAL currently uses. Axioms defining the API protocol could then specify the effect of a particular command on the complexity of obtaining specific terms. This idea would need a lot of work, but the hope is that the ¤exibility that CORAL showed in its adaptability to group protocols would make it suitable for adaptation to security APIs.

12.4 Proving Theorems

Though we have used it in this work to refute non-theorems, the Comon-Nieuwenhuis method for proof by consistency was originally conceived for proving inductive theorems. This means that in theory, CORAL can also show security properties of protocols to be correct when there are no attacks to be found. However, to make this work in practice would require some considerable work. The formulae to be proved are significantly larger than the kinds of examples that have been proved by proof by consistency in the past. The proof by consistency method has recently been used successfully for a verification case study on JavaCard bytecode, [Barthe and Stratulat, 2003].

[1] Application Program Interface - a protocol describing how other processes can interact with the process in question.

This was carried out using the SPIKE prover, [Bouhoula and Rusinowitch, 1995], which utilises a divergence critic developed by Walsh to assist in finding proofs, [Walsh, 1996]. Similar critics could be added to CORAL.

A key to any attempt at inductive proof is devising and proving appropriate intermediate lemmas in order to eventually prove the goal. Walsh's divergence critic is one way of suggesting lemmas. In the specific case of security protocols, there may be some fixed lemmas we can use to increase our chances of achieving saturation in the case of a correct protocol, or speeding up attack finding in the case of faulty one. For example, we could experiment with the various 'forwarding' and 'unicity' lemmas used by Paulson in his inductive protocol model.

Several redundancy detection techniques for general first-order proving have been suggested that are not implemented in SPASS, for example, contextual rewriting, [Nivela, 1993]. These could be implemented SPASS and used in CORAL in order to assess their usefulness for aiding proof by consistency.

12.5 Other Application Areas

There are other areas of automated reasoning where a tool for detecting and refuting incorrect inductive conjectures would be of use. The most significant of these is perhaps inside an automated inductive theorem prover. The application here would be to refuting incorrect lemmas and generalisations suggested by the system during the course of a proof. As mentioned above, most inductive proofs of any size require several intermediate lemmas to be proved before the main conjecture can be proved. These lemmas may have to be generalised to facilitate a proof in the system, or indeed the original conjecture itself may have to be generalised. Many tools contain mechanisms for suggesting these lemmas and generalisations, e.g. [Ireland, 1996], but the lemmas and generalisations may actually be false. If we can detect this early, for example using CORAL, then we can save a lot of work for the prover. Furthermore, the counterexample discovered could be useful in suggesting a correction to the formula. Given the parallel nature of CORAL's architecture, it should be simple to set up a version that would run on a separate machine, accepting conjectures to investigate from the main

prover as they arise in a proof attempt. This arrangement would prevent the resource-hungry theorem prover at the heart of CORAL from slowing the main inductive prover down.

12.6 Summary

In this chapter, we have suggested several ways in which CORAL might be developed in future work. These can be split into two main areas:

1. Specialising CORAL to security protocols, where ideas include hard-wiring more of the model into the inference process, developing new domain-specific heuristics, experimenting with more multicast key management protocols and adapting the model to deal with protocols requiring more detailed cryptographic models.

2. Applying CORAL to other problems, such as refuting incorrect conjectures inside an automated inductive theorem prover.

Further development in these areas will allow us to build a more complete picture of the potential of the Comon-Nieuwenhuis approach, as well attacking some of the major outstanding problems in automated cryptographic protocol analysis.

Chapter 13

Conclusions

In this chapter, we discuss in more detail the research contributions of the thesis that were outlined in Chapter 1, in the light of the presentation we have given in Chapters 2–11. This serves to summarise and evaluate the thesis.

13.1 Evaluation of Research Contributions

In Chapter 1, we presented very brie¤y the research contributions of this thesis. These were given in in the form of two hypotheses that we believe are supported by the evidence presented in this thesis. Having described the background, the theory, the implementation, the experiments we have carried out and related work in more detail, we are now in a position to discuss these hypotheses more fully.

Hypothesis 1 By using the Comon-Nieuwenhuis strategy to refute incorrect inductive conjectures in a first-order version of Paulson's security protocol model, we can effectively find attacks on faulty security protocols.

In Chapter 5 we showed how we adapted the SPASS proved to use the Comon-Nieuwenhuis strategy. We presented our model in Chapter 6, and showed how, together with the heuristics described in Chapter 7, it could be used to discover 10 standard protocol attacks in Chapter 8.

Hypothesis 2 The use of a simple first-order trace based formalism for analysing protocols allows us to quickly adapt to unusual protocols, such as group key agreement and key management protocols, which approaches optimised for standard 2 and 3 party protocols would struggle with

The model we presented in Chapter 6 was shown to be flexible enough to model general-sized group protocols in Chapter 9, and, with minor changes, to model a multicast group key management protocol in Chapter 10. As we saw in §11.1.1 and §11.2, some other models used for protocol analysis have less redundancy and so can find attacks on standard protocols from the Clark-Jacob corpus faster. However, these special purpose models with their associated special purpose inference mechanisms generally lack the ability to formalise arbitrary sized group protocols and protocols allowing the same agents to play an arbitrary number of different roles in a protocol. As we saw in Chapter 2, the field of standard protocol analysis is now somewhat saturated. CORAL's ability to adapt to new non-standard protocols is therefore important. CORAL's flexibility is a result of its simple, open ended, Horn clause inductive model, and the fact that this model is mechanised in a first-order theorem prover with support for equational reasoning.

How significant is the ability to model group protocols in a general way, i.e. without predetermining the size of the group? We have seen from our first case study that it is at least of some significance, since the Asokan–Ginzboorg protocol had an attack on a group of size 2 and one on a group of size 3. The attack for a three person group is impossible for a group of size 2, since in the smaller group, once a single message 2 has been sent, there is nobody remaining for the spy to impersonate by replaying that message 2. This means that if we had chosen the simplest model, we would not have found the attack. On another group protocol, Pereira and Quisquater found an attack on a group of minimum size 4, [Pereira and Quisquater, 2001]. So predetermining the size of the group certainly does prejudice the chances of finding an attack. However, if we can analyse and check a group protocol of fixed size very fast, then we could perhaps analyse many different sized groups from size 1 to size n, and then conclude the protocol is secure for groups up to n if we have not found any attacks. This would require the person doing the analysis to construct different models for each different

sized group however, increasing the risk of making a mistake. Given that protocol designers generally seem reluctant to use formal tools as it is, making the user do even more work is undesirable.

Our second case study, on the Tanaka-Sato ARF Framework, is also significant, for several reasons. One is that the protocol requires the same agents to play a possibly unbounded number of different roles during one run. Protocol analysis approaches tailored to the Clark-Jacob corpus would struggle to deal with this, and would certainly require significant adaptation, whereas CORAL's simple trace based model handled it very naturally. The case study also highlights the inadequacy of the only previous attempt at mechanised analysis of a multicast key management protocol. Taghdiri and Jackson made a model of this protocol, but neglected to model anyone who actually trying to break the protocol. CORAL's model included a standard active intruder, and this allowed us to discover 2 new attacks.

As a final remark in support of our second hypothesis, we observe that in the course of the project, CORAL found five previously unknown attacks on two very different group protocols, making it currently the leading automated tool for finding group protocol attacks.

13.2 Final Summary

- CORAL succeeds as a counterexample finder for inductive conjectures in a first-order version of Paulson's inductive protocol model, and thanks to its use of such an expressive formalism, as a tool for discovering novel attacks on protocols that other tools have difficulty modelling.

- Compared to some other protocol analysis tools, CORAL is quite slow. This is a result of its generality. Further specialisation towards protocol analysis, and in particular to particular types of protocols, would enable the implementation of heuristics to make it faster.

- Other possible future directions include: trying to prove security properties in the absence of attacks using critics and lemmas; developing an encryption model

adequate for finding attacks where many guesses are made and the results used to break cryptographic functions; applying CORAL to other areas where incorrect inductive conjectures must be refuted.

Appendix A

Protocol Model Files

This appendix contains the specification file for the Needham-Schroeder public key protocol. The specification for the Needham-Schroeder protocol was used as a worked example for explaining the formalism in Chapter 6, but then we explained in Chapter 7 how we optimised the formalism using step compression. Here, we give the specification in its final form, with the step compression optimisations marked in the comments.

Additionally, this appendix contains the changes to the protocol-specific part of the formalism required to model the Clark-Jacob, Neuman-Stubblebine and BAN Otway-Rees protocols.

A.1 The Needham-Schroeder Public Key Protocol

Recall that in our protocol notation, the Needham-Schroeder public key protocol runs like this:

1. $A \rightarrow B : \{\!|\, N_A, A |\!\}_{pubK_B}$
2. $B \rightarrow A : \{\!|\, N_A, N_B |\!\}_{pubK_A}$
3. $A \rightarrow B : \{\!|\, N_B |\!\}_{pubK_B}$

The clauses for modelling the protocol are:

% defining basic sorts

$\rightarrow$ *eqagent(U,U)=true*

eqagent(U,V)=false $\rightarrow$ *eqagent(s(U),s(V))=false*

$\rightarrow$ *eqagent(spy,s(U))=false*

$\rightarrow$ *eqagent(s(U),spy)=false*

$\rightarrow$ *eqagent(s(U),a)=false*

$\rightarrow$ *eqagent(a,s(U))=false*

$\rightarrow$ *eqagent(a,spy)=false*

$\rightarrow$ *eqagent(spy,a)=false*

$\rightarrow$ *agent(spy)=true*

$\rightarrow$ *agent(a)=true*

$\rightarrow$ *agent(s(a))=true*

agent(s(U))=true $\rightarrow$ *agent(s(s(U)))=true*

$\rightarrow$ *number(0)=true*

number(U)=true $\rightarrow$ *number(s(U))=true*

% The *eq* function

$\rightarrow$ *eq(U,U)=true*

eq(MSG1,MSG2)=false $\rightarrow$ *eq(sent(U,W,MSG1),sent(V,Y,MSG2))=false*

eqagent(A,B)=false $\rightarrow$ *eq(sent(A,W,X),sent(B,Y,Z))=false*

eqagent(A,B)=false $\rightarrow$ *eq(sent(W,A,X),sent(Y,B,Z))=false*

eq(H1,H2)=false $\rightarrow$ *eq(cons(H1,T1),cons(H2,T2))=false*

eq(T1,T2)=false $\rightarrow$ *eq(cons(H1,T1),cons(H2,T2))=false*

eq(MSG1,MSG2)=false $\rightarrow$ *eq(encr(MSG1,KEY1),encr(MSG2,KEY2))=false*

eq(KEY1,KEY2)=false $\rightarrow$ *eq(encr(MSG1,KEY1),encr(MSG2,KEY2))=false*

eq(M1,M2)=false $\rightarrow$ *eq(pair(M1,X),pair(M2,Y))=false*

eq(M1,M2)=false $\rightarrow$ *eq(pair(X,M1),pair(Y,M2))=false*

eq(principal(U),principal(V))=true $\rightarrow$ *eq(U,V)=true*

eq(U,V),false $\rightarrow$ *eq(nonce(U),nonce(V))=false*

eq(U,V)=false $\rightarrow$ *eq(s(U),s(V))=false*

$\rightarrow$ *eq(nonce(U),principal(V))=false*

$\rightarrow$ *eq(principal(U),nonce(V))=false*.

$\rightarrow eq(a,s(U))=false$

$\rightarrow eq(spy,s(U))=false$

$\rightarrow eq(0,s(U))=false$

$\rightarrow eq(s(U),a)=false$

$\rightarrow eq(s(U),spy)=false$

$\rightarrow eq(s(U),0)=false$

$\rightarrow eq(a,spy)=false$

$\rightarrow eq(spy,a)=false$

% The list *member* function

$eq(H1,H2) = false \wedge member(H1,L) = false$
$\quad \rightarrow member(H1,cons(H2,L)) = false$

$\rightarrow member(U,nil)=false$

$member(H,L)=true \rightarrow member(H,cons(W,L))=true$

$\rightarrow member(H,cons(H,T))=true$

$member(U,nil)=true \rightarrow$

% The *parts* operator

$in(U,parts(V)) = false \wedge in(U,parts(W)) = false$
$\quad \rightarrow in(U,parts(cons(sent(X,Y,W),V))) = false$

$in(U,parts(V)) = false \wedge in(U,parts(W)) = false$
$\quad \rightarrow in(U,parts(encr(V,W))) = false$

$in(U,parts(V)) = false \wedge in(U,parts(W)) = false$
$\quad \rightarrow in(U,parts(pair(V,W))) = false$

$eq(U,V)=false \rightarrow in(nonce(U),parts(nonce(V)))=false$

$eq(U,V)=false \rightarrow in(key(U),parts(key(V)))=false$

$\rightarrow in(nonce(U),parts(principal(V)))=false$

$\rightarrow in(nonce(U),parts(key(V)))=false$

$\rightarrow in(U,parts(nil))=false$

% The spy - *analz* and *synth* operators

$in(encr(U,pubk(spy)),analz(V))=true \rightarrow in(U,analz(V))=true$

$agent(U)=true \wedge in(V,synth(analz(W)))=true$
$\quad \rightarrow in(encr(V,pubk(U)),synth(analz(W)))=true$

in(pair(U,V),analz(W))=true → in(U,analz(W))=true

in(pair(U,V),analz(W))=true → in(V,analz(W))=true

in(U,synth(V))=true∧ in(W,synth(V))=true
→ in(pair(U,W),synth(V))=true

in(U,analz(V))=true → in(U,synth(analz(V)))=true

agent(U)=true → in(principal(U),analz(V))=true

→ in(U,synth(analz(nil)))=false

→ in(U,analz(nil))=false

%%% The rest of the clauses are specific to NSPK

% honest agents taking part in the protocol:

agent(A)=true ∧ agent(B)=true ∧ number(NA)=true ∧ m(Trace)=true ∧
in(nonce(N),parts(Trace))=false
→m(cons(sent(A,B,encr(pair(nonce(N),principal(A)),pubk(B))),Trace))=true

number(NB)=true∧ m(Trace)=true∧ in(nonce(NB),parts(Trace)),false
member(sent(X,B,encr(pair(nonce(NA),principal(A)),pubk(B))),Trace)=true
→ m(cons(sent(B,A,encr(pair(nonce(NA),nonce(NB)),pubk(A))),Trace))=true

m(Trace)=true∧
member(sent(X,A,encr(pair(nonce(NA),nonce(NB)),pubk(A))),Trace)=true∧
member(sent(A,B,encr(pair(nonce(NA),principal(A)),pubk(B))),Trace)=true
→ m(cons(sent(A,B,encr(nonce(NB),pubk(B))),Trace))=true

% spy learning things from the trace

member(sent(X,Y,encr(pair(nonce(NA),principal(A)),pubk(B))),Trace)=true
→in(encr(pair(nonce(NA),principal(A)),pubk(B)),analz(Trace))=true

member(sent(X,Y,encr(pair(nonce(NA),nonce(NB)),pubk(A))),Trace)=true
→in(encr(pair(nonce(NA),nonce(NB)),pubk(B)),analz(Trace))=true

member(sent(X,Y,encr(nonce(NB),pubk(B))),Trace)=true
→in(encr(nonce(NB),pubk(B)),analz(Trace))=true

% Spy adding messages to the trace

% spy faking a message 1
m(Trace)=true ∧
in(encr(pair(nonce(NA),principal(A)),pubk(B)),synth(analz(Trace)))=true
∧ eq(spy,B)=false →
m(cons(sent(spy,B,encr(pair(nonce(NA),principal(A)),pubk(B))),Trace))=true

% spy faking a message 2
m(Trace)=true∧
in(encr(pair(nonce(NA),nonce(NB)),pubk(A)),synth(analz(Trace)))=true
∧ eq(spy,A)=false →
m(cons(sent(spy,A,encr(pair(nonce(NA),nonce(NB)),pubk(A))),Trace))=true

% spy faking a message 3
m(Trace)=true∧
in(encr(nonce(NB),pubk(B)),synth(analz(Trace)))=true
∧ eq(spy,B)=false →
m(cons(sent(spy,B,encr(nonce(NB),pubk(B))),Trace))=true

A.2 Changes for the Clark-Jacob Protocol

Here we give the clauses required to model the Clark-Jacob protocol. These take the place of the clauses marked above as being specific to the Needham-Schroeder protocol. No other changes to the model were required. The Clark-Jacob handshake consists of just two messages:

1. $A \rightarrow B : \{\!|\, N_A |\!\}_{K_{AB}}$
2. $B \rightarrow A : \{\!|\, s(N_A) |\!\}_{K_{AB}}$

These were modelled by three clauses (because of the symmetric nature of the key, see §8.3).

m(Trace)=true ∧
agent(A=true ∧
agent(B)=true ∧
number(NA)=true ∧
member(sent(X,Y,encr(nonce(NA),Z)),Trace)=false
→
m(cons(sent(A,B,encr(nonce(NA),key(pair(A,B)))),Trace))=true

member(sent(Z,B,encr(nonce(NA),key(pair(A,B)))),Trace)=true ∧
m(Trace)=true
→
m(cons(sent(B,A,encr(s(nonce(NA)),key(pair(A,B)))),Trace))=true

% additional axiom to make keys symmetric

key(pair(X,Y))=key(pair(Y,X))

A.3 Changes for the Neuman-Stubblebine Protocol

For this protocol, we allowed type confusion between nonces and keys. The Neuman-Stubblebine protocol runs like this:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow S : B, \{\!|\ A, N_A, T_B\ |\!\}_{K_B}, N_B$
3. $S \rightarrow A : \{\!|\ B, N_A, K_{AB}, T_B\ |\!\}_{K_A}, \{\!|\ A, K_{AB}, T_B\ |\!\}_{K_B}, N_B$
4. $A \rightarrow B : \{\!|\ A, K_{AB}, T_B\ |\!\}_{K_B}, \{\!|\ N_B\ |\!\}_{K_{AB}}$

The clauses required were:

% message 1

agent(A=true ∧
agent(B)=true ∧
number(NA)=true ∧
m(Trace)=true ∧
% nonce freshness check
in(nonce(NA),parts(Trace)),false
→
m(cons(sent(A,B,pair(principal(A,nonce(NA))),Trace))=true

%% message 2 *m(Trace)=true* ∧
number(NB)=true ∧
member(sent(X,B,pair(principal(A,nonce(NA))),Trace)=true ∧
in(nonce(NB),parts(Trace))=false →
m(cons(sent(B,server,triple(principal(B),encr(triple(principal(A,nonce(NA), time(T)),longtermkey(B)),nonce(NB))),Trace))=true

% message 3 *m(Trace)=true* ∧
member(sent(X,server,triple(principal(B),encr(triple(principal(A,nonce(NA),time(T)), longtermkey(B)),nonce(NB))),Trace)=true
→
m(cons(sent(server,A,triple(encr(quad(principal(B),nonce(NA),key(NA),time(T)), longtermkey(A),encr(triple(principal(A,key(NA),time(T)), longtermkey(B)),nonce(NB))),Trace))=true

% message 4

m(Trace)=true ∧
agent(A=true ∧

agent(B)=true ∧
member(sent(X,A,triple(encr(quad(principal(B),nonce(NA),key(NA),time(T)),
longtermkey(A),packet,nonce(NB))),Trace)=true ∧
member(sent(A,B,pair(principal(A,nonce(NA))),Trace)=true
→
m(cons(sent(A,B,pair(packet,encr(nonce(NB),key(NA)))),Trace))=true

%% spy faking a message 1

m(Trace)=true ∧
in(pair(principal(A,nonce(NA)),synth(analz(Trace)))=true ∧
eq(spy,B)=false
→
m(cons(sent(spy,B,pair(principal(A,nonce(NA))),Trace))=true

%% spy faking a message 2

m(Trace)=true
in(triple(principal(B),encr(triple(principal(A,nonce(NA),time(T)),
longtermkey(B)),nonce(NB)),synth(analz(Trace)))=true ∧
→
m(cons(sent(spy,server,triple(principal(B),encr(triple(principal(A,nonce(NA),
time(T)),longtermkey(B)),nonce(NB))),Trace))=true

%%%% faked message 3
%%% note for type confusion, we allow Key instead of key(NA) here

m(Trace)=true ∧
in(triple(encr(quad(principal(B),nonce(NA),key(NA),time(T)),longtermkey(A),
encr(triple(principal(A,Key,time(T)),longtermkey(B)),nonce(NB)),
synth(analz(Trace)))=true ∧
eq(spy,A)=false ∧
→
m(cons(sent(spy,A,triple(encr(quad(principal(B),nonce(NA),key(NA),time(T)),
longtermkey(A),encr(triple(principal(A,Key,time(T)),
longtermkey(B)),nonce(NB))),Trace))=true

% faked message 4 - again Key instead of key(NA) used here

m(Trace)=true ∧
in(pair(encr(triple(principal(A,Key,time(T)),longtermkey(B)),
encr(nonce(NB),Key)),synth(analz(Trace)))=true
→
m(cons(sent(spy,B,pair(encr(triple(principal(A,Key,time(T)),longtermkey(B)),
encr(nonce(NB),Key))),Trace))=true

%% spy eavesdropping on message 1

member(sent(X,Y,pair(principal(A,nonce(NA))),Trace)=true
→ in(nonce(NA),analz(Trace))=true

% two clauses to model two possible things
% that can be obtained from eavesdropping on message 2

member(sent(X,Y,triple(principal(B),encr(triple(principal(A,nonce(NA),time(T)),
longtermkey(B)),nonce(NB))),Trace)=true
→
in(encr(triple(principal(A,nonce(NA),time(T)),longtermkey(B)),analz(Trace))=true

member(sent(X,Y,triple(principal(B),encr(triple(principal(A,nonce(NA),time(T)),
longtermkey(B)),nonce(NB))),Trace)=true
→ in(nonce(NB),analz(Trace))=true

%%% three things from a message 3

member(sent(server,A,triple(encr(quad(principal(B),nonce(NA),key(NA),time(T)),
longtermkey(A),encr(triple(principal(A,key(NA),time(T)),
longtermkey(B)),nonce(NB))),Trace)=true
→
in(nonce(NB),analz(Trace))=true

member(sent(server,A,triple(encr(quad(principal(B),nonce(NA),key(NA),time(T)),
longtermkey(A),encr(triple(principal(A,key(NA),time(T)),
longtermkey(B)),nonce(NB))),Trace)=true
→
in(encr(quad(principal(B),nonce(NA),key(NA),time(T)),longtermkey(A),analz(Trace))=true

member(sent(server,A,triple(encr(quad(principal(B),nonce(NA),key(NA),time(T)),
longtermkey(A),encr(triple(principal(A,key(NA),time(T)),
longtermkey(B)),nonce(NB))),Trace)=true
→
in(encr(triple(principal(A,key(NA),time(T)),longtermkey(B)),analz(Trace))=true

%two things from a message 4

member(sent(X,Y,pair(encr(triple(principal(A,Key,time(T)),longtermkey(B)),
encr(nonce(NB),Key))),Trace)=true
→
in(encr(triple(principal(A,Key,time(T)),longtermkey(B)),analz(Trace))=true

member(sent(X,Y,pair(encr(triple(principal(A,Key,time(T)),longtermkey(B)),
encr(nonce(NB),Key))),Trace)=true
→
in(encr(nonce(NB),Key),analz(Trace))=true

Whereas the Needham-Schroeder public key protocol only has messages of length 1 and 2, the Neuman-Stubblebine protocol has messages of length 2,3 and 4. So, we require the following additonal axioms:

% rules for parts:

in(X,parts(Y))=false∧
in(X,parts(Z))=false∧
in(X,parts(G))=false
  →*in(X,parts(triple(Y,Z,G)))=false*

in(X,parts(Y))=false∧
in(X,parts(Z))=false∧
in(X,parts(G))=false∧
in(X,parts(H))=false∧
  →*in(X,parts(quad(Y,Z,G,H)))=false*

% rules for analz

in(triple(X,Y,Z),analz(Xset))=true
  →*in(X,analz(Xset))=true*

in(triple(X,Y,Z),analz(Xset))=true
  →*in(Y,analz(Xset))=true*

in(triple(X,Y,Z),analz(Xset))=true
  →*in(Z,analz(Xset))=true*

in(quad(X,Y,Z,G),analz(Xset))=true
  →*in(Y,analz(Xset))=true*

in(quad(X,Y,Z,G),analz(Xset))=true
  →*in(X,analz(Xset))=true*

in(quad(X,Y,Z,G),analz(Xset))=true
  →*in(Z,analz(Xset))=true*

in(quad(X,Y,Z,G),analz(Xset))=true
  →*in(G,analz(Xset))=true*

% rules for synth

in(X,synth(Xset))=true∧
in(Y,synth(Xset))=true
  →*in(pair(X,Y),synth(Xset))=true*

in(X,synth(Xset))=true∧
in(Y,synth(Xset))=true∧
in(Z,synth(Xset))=true
  →in(triple(X,Y,Z),synth(Xset))=true

in(X,synth(Xset))=true∧
in(Y,synth(Xset))=true∧
in(Z,synth(Xset))=true∧
in(G,synth(Xset))=true
  →in(quad(X,Y,Z,G),synth(Xset))=true

% rules for eq
eq(M1,M2)=false → eq(triple(M1,G,H),triple(M2,X,Y))=false

eq(M1,M2)=false → eq(triple(G,M1,H),triple(X,M2,Y))=false

eq(M1,M2)=false → eq(triple(G,H,M1),triple(X,Y,M2))=false

eq(M1,M2)=false → eq(quad(M1,G,H,I),quad(M2,X,Y,Z))=false

eq(M1,M2)=false → eq(quad(G,M1,H,I),quad(X,M2,Y,Z))=false

eq(M1,M2)=false → eq(quad(G,H,M1,I),quad(X,Y,M2,Z))=false

eq(M1,M2)=false → eq(quad(G,H,I,M1),quad(X,Y,Z,M2))=false

A.4 Changes for the Otway-Rees Protocol

Here we show the different clauses required to model the Otway-Rees protocol. The Otway-Rees protocol (or to be exact, the Burrows-Abadi-needham version of the protocol, [Burrows et al., 1990]) runs like this:

1. $A \rightarrow B : N_A, A, B, \{\!| N_A, A, B |\!\}_{K_A}$

2. $B \rightarrow S : N_A, A, B, \{\!| N_A, A, B |\!\}_{K_A}, N_B, \{\!| N_A, A, B |\!\}_{K_B}$

3. $S \rightarrow B : N_A, \{\!| N_A, K_{AB} |\!\}_{K_A}, \{\!| N_B, K_{AB} |\!\}_{K_B}$

4. $B \rightarrow A : N_A, \{\!| N_A, K_{AB} |\!\}_{K_A}$

The clauses for modelling honest agents taking part in the protocol are:

% message 1: A to B

agent(A)=true ∧ *agent(B)=true* ∧
number(NA)=true ∧
m(Trace)=true ∧
% nonce *NA* must be fresh
in(nonce(NA),parts(Trace))=false
→
m(cons(sent(A,B,quad(nonce(NA),principal(A),principal(B),encr(
triple(nonce(NA),principal(A),principal(B)),longtermkey(A)))),Trace))=true

% message 2: B to server

m(Trace)=true∧
number(NB)=true∧
% Note B cannot read *Packet*, he just passes it on
member(sent(X,B,quad(nonce(NA),principal(A),principal(B),
Packet)),Trace) = true
% Nonce *NB* freshness check
in(nonce(NB),parts(Trace))=false
→
m(cons(sent(B,server,sex(nonce(NA),principal(A),principal(B),
packet,nonce(NB),
encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(B))))),Trace))=true

% message 3: server responds with msg to B

m(Trace)=true ∧
member(sent(X,server,sex(nonce(NA),principal(A),principal(B),
encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(A)),
nonce(NB),
encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(B)))),
Trace)=true
→
m(cons(sent(server,B,triple(nonce(NA),
encr(pair(nonce(NA),key(NA)),longtermkey(A)),
encr(pair(nonce(NB),key(NA)),longtermkey(B)))),Trace))=true

% message 4: B to A
m(Trace)=true ∧
member(sent(X,B,triple(nonce(NA),Packet,
encr(pair(nonce(NB),key(K)),longtermkey(B)))),Trace)=true
member(sent(B,server,sex(nonce(NA),principal(A),principal(B),
Packet2,nonce(NB),

encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(B)))),Trace)=true
$\rightarrow$
m(cons(sent(B,A,pair(nonce(NA),Packet)),Trace))=true

The Otway-Rees problem file also has different clauses for the spy's sending and receiving of messages, following the heuristics outlined in Chapter 7.

% spy faking a message 1

m(Trace)=true$\wedge$
in(quad(nonce(NA),principal(A),principal(B),encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(A))),synth(analz(Trace)))=true$\wedge$
eq(spy,B)=false $\rightarrow$
m(cons(sent(spy,B,quad(nonce(NA),principal(A),principal(B),encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(A)))),Trace))=true

% spy faking a message 2

m(Trace)=true $\wedge$
in(sex(nonce(NA),principal(A),principal(B),encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(A)),nonce(NB),encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(B))),synth(analz(Trace)))=true
$\rightarrow$
m(cons(sent(spy,server,sex(nonce(NA),principal(A),principal(B),encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(A)),nonce(NB),encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(B)))),Trace))=true

% spy faking a message 3

m(Trace)=true $\wedge$
in(triple(nonce(NA),encr(pair(nonce(NA),key(K)),longtermkey(A)),encr(pair(nonce(NA),key(K)),longtermkey(B))),synth(analz(Trace)))=true
$\wedge$ *eq(spy,A)=false*$\rightarrow$
m(cons(sent(spy,A,triple(nonce(NA),encr(pair(nonce(NA),key(K)),longtermkey(A)),encr(pair(nonce(NA),key(K)),longtermkey(B)))),Trace))=true

% spy faking a message 4

m(Trace)=true $\wedge$
in(pair(nonce(NA),encr(pair(nonce(NA),key(K)),longtermkey(A))),synth(analz(Trace)))=true $\wedge$
eq(spy,A)=false $\rightarrow$
m(cons(sent(spy,A,pair(nonce(NA),encr(pair(nonce(NA),key(K)),longtermkey(A)))),Trace))=true

% spy eavesdropping on a message 1

*member(sent(X,Y,quad(nonce(NA),principal(A),principal(B),
encr(triple(nonce(NA),principal(A),principal(B)),
longtermkey(A)))),Trace)=true
→ in(nonce(NA),analz(Trace))=true*

*member(sent(X,Y,quad(nonce(NA),principal(A),principal(B),
encr(triple(nonce(NA),principal(A),principal(B)),
longtermkey(A)))),Trace)=true
→in(encr(triple(nonce(NA),principal(A),principal(B)),
longtermkey(A)),analz(Trace))=true*

% spy eavesdropping on a message 2

*member(sent(X,Y,sex(nonce(NA),principal(A),principal(B),
encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(A)),
nonce(NB),encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(B)))),
Trace)=true
→in(nonce(NA),analz(Trace))=true*

*member(sent(X,Y,sex(nonce(NA),principal(A),principal(B),
encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(A)),
nonce(NB),encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(B)))),
Trace)=true
→in(nonce(NB),analz(Trace))=true*

*member(sent(X,Y,sex(nonce(NA),principal(A),principal(B),
encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(A)),
nonce(NB),encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(B)))),
Trace)=true
→in(encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(A)),analz(Trace))=true*

*member(sent(X,Y,sex(nonce(NA),principal(A),principal(B),
encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(A)),
nonce(NB),encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(B)))),
Trace)=true
→in(encr(triple(nonce(NA),principal(A),principal(B)),longtermkey(B)),analz(Trace))=true*

% spy eavesdropping on a message 3

*member(sent(X,Y,triple(nonce(NA),encr(pair(nonce(NA),key(Z)),longtermkey(A)),
encr(pair(nonce(NB),key(Z)),longtermkey(B)))),Trace)=true
→in(nonce(NA),analz(Trace))=true*

*member(sent(X,Y,triple(nonce(NA),encr(pair(nonce(NA),key(Z)),longtermkey(A)),
encr(pair(nonce(NB),key(Z)),longtermkey(B)))),Trace)=true
→in(encr(pair(nonce(NA),key(Z)),longtermkey(A)),analz(Trace))=true*

member(sent(X,Y,triple(nonce(NA),encr(pair(nonce(NA),key(Z)),longtermkey(A)),

encr(pair(nonce(NB),key(Z)),longtermkey(B)))),Trace)=true
→in(encr(pair(nonce(NB),key(Z)),longtermkey(B)),analz(Trace))=true

% spy eavesdropping on a message 4

member(sent(X,Y,pair(nonce(NA),encr(pair(nonce(NA),key(Z)),longtermkey(A)))),Trace)=true
→in(nonce(NA),analz(Trace))=true

member(sent(X,Y,pair(nonce(NA),encr(pair(nonce(NA),key(Z)),longtermkey(A)))),Trace)=true
→in(encr(pair(nonce(NA),key(Z)),longtermkey(A)),analz(Trace))=true

Finally, whereas previous protocol have had messages of length 1,2,3 and 4, the Otway-Rees protocol has messages of length 2,3,4 and 6. So, we require the following additional axioms:

% rules for parts:

in(X,parts(Y))=false∧
in(X,parts(Z))=false∧
in(X,parts(G))=false∧
in(X,parts(H))=false∧
in(X,parts(I))=false∧
in(x,parts(J))=false
→in(X,parts(sex(Y,Z,G,H,I,J)))=false

% rules for analz

in(sex(X,Y,Z,G,H,I),analz(Xset))=true
→in(Y,analz(Xset))=true

in(sex(X,Y,Z,G,H,I),analz(Xset))=true
→in(X,analz(Xset))=true

in(sex(X,Y,Z,G,H,I),analz(Xset))=true
→in(Z,analz(Xset))=true

in(sex(X,Y,Z,G,H,I),analz(Xset))=true
→in(G,analz(Xset))=true

in(sex(X,Y,Z,G,H,I),analz(Xset))=true
→in(H,analz(Xset))=true

in(sex(X,Y,Z,G,H,I),analz(Xset))=true
→in(I,analz(Xset))=true

% rules for synth

in(X,synth(Xset))=true$\wedge$
in(Y,synth(Xset))=true$\wedge$
in(Z,synth(Xset))=true$\wedge$
in(G,synth(Xset))=true$\wedge$
in(H,synth(Xset))=true$\wedge$
in(I,synth(Xset))=true
$\rightarrow$*in(sex(X,Y,Z,G,H,I),synth(Xset))=true*

% rules for eq

eq(M1,M2)=false $\rightarrow$ *eq(sex(M1,G,H,I,J,K),sex(M2,X,Y,Z,A,B))=false*

eq(M1,M2)=false $\rightarrow$ *eq(sex(G,M1,H,I,J,K),sex(X,M2,Y,Z,A,B))=false*

eq(M1,M2)=false $\rightarrow$ *eq(sex(G,H,M1,I,J,K),sex(X,Y,M2,Z,A,B))=false*

eq(M1,M2)=false $\rightarrow$ *eq(sex(G,H,I,M1,J,K),sex(X,Y,Z,M2,A,B))=false*

eq(M1,M2)=false $\rightarrow$ *eq(sex(G,H,I,J,M1,K),sex(X,Y,Z,A,M2,B))=false*

eq(M1,M2)=false $\rightarrow$ *eq(sex(G,H,I,J,K,M1),sex(X,Y,Z,A,B,M2))=false*

Appendix B

The Model for the Asokan–Ginzboorg Protocol

This chapter gives details of the specification file for the Asokan–Ginzboorg protocol, used for the case study in Chapter 9. Here again is a description of the Asokan–Ginzboorg protocol. It is explained in §9.1.

$$\begin{array}{llllll}
1. & M_n & \rightarrow & \text{ALL} & : & M_n, \{\!|\, E \,|\!\}_P & \\
2. & M_i & \rightarrow & M_n & : & M_i, \{\!|\, R_i, S_i \,|\!\}_E & i = 1, \ldots, n-1 \\
3. & M_n & \rightarrow & M_i & : & \{\!|\, \{Sj, j = 1, \ldots, n\} \,|\!\}_{R_i} & i = 1, \ldots, n-1 \\
4. & M_i & \rightarrow & M_n & : & M_i, \{\!|\, S_i, h(S_1, \ldots, S_n) \,|\!\}_K & \text{some } i,\ K = f(S_1, \ldots, S_n)
\end{array}$$

We now give the parts of the specification file specific to the protocol.

% Message 1 % MN initiates a run as leader

agent(MN)=true ∧
% uncomment this to keep the spy out of the room:
% *eqagent(MN,spy)=false* ∧
in(key(E),parts(Trace))=false ∧ *number(E)=true* ∧
number(Pass)=true ∧ *m(Trace)=true*
→
m(cons(sent(MN,all,pair(principal(MN),encr(key(E),key(Pass)))),Trace))=true

% Message 2

m(Trace)=true ∧
number(Ri)=true ∧ *number(Si)=true* ∧

in(nonce(Si),parts(Trace))=false ∧
in(nonce(Ri),parts(Trace))=false ∧
eqagent(Mi,MN)=false ∧
% again, uncomment this to keep the spy out of the room:
% *eqagent(MN,spy)=false* ∧
agent(Mi)=true ∧
member(sent(x,all,pair(principal(MN),encr(key(E),key(Pass)))),Trace)=true
→
m(cons(sent(Mi,MN,pair(principal(Mi),encr(pair(nonce(Ri),nonce(Si)),key(E)))),Trace))=true

% Message 3

eqagent(A,MN)=false ∧ *eqagent(A,spy)=false* ∧ *m(Trace)=true* ∧
member(sent(MN,all,pair(principal(MN),encr(key(E),key(P)))),Trace)=true ∧
all_msg2s_received(Trace,A,MN,E,nil,
cons(sent(MN,A,encr(Package,Ri)),NewTrace),FinalPackage)=true
→
m(cons(sent(MN,A,encr(Package,Ri)),NewTrace))=true

% Message 4
% anyone might send message 4. They must have sent a message 2 and
% received a message 3

m(Trace)=true ∧
agent(Mi)=true ∧
member(sent(Mi,MN,pair(principal(Mi),encr(pair(nonce(Ri),nonce(Si)),key(E)))),Trace)=true
∧ *member(sent(Y,Mi,encr(Package,nonce(Ri))),Trace)=true* ∧
→ *m(cons(sent(Mi,MN,pair(principal(Mi),encr(pair(nonce(Si),h(Package)),*
f(Package)))),Trace))=true

% base case for all_msg2s_received

member(sent(X,MN,pair(principal(a),
encr(pair(nonce(Ri),nonce(Si)),key(E)))),Trace)=true
→
all_msg2s_received(Trace,a,MN,E,Package,
cons(sent(MN,a,encr(cons(nonce(Si),Package),nonce(Ri))),Trace),
cons(nonce(Si),Package)=true

% recursive case for all_msg2s_received

member(sent(X,MN,pair(principal(s(MX)),
encr(pair(nonce(Ri),nonce(Si)),key(E)))),Trace)=true∧

all_msg2s_received(Trace,MX,MN,E,cons(nonce(Si),Package),
NewTrace,FinalPackage)=true
$\rightarrow$
all_msg2s_received(Trace,s(MX),MN,E,Package,
cons(sent(MN,s(MX),encr(FinalPackage,nonce(Ri))),
NewTrace),FinalPackage)=true

% spy eavesdropping on trace

member(sent(X,Y,pair(principal(xa),encr(key(E),key(Pass)))),Trace)=true
$\rightarrow$*in(encr(key(E),key(Pass)),analz(Trace))=true*

member(sent(X,Y,pair(principal(Mi),encr(pair(nonce(Ri),nonce(Si)),key(E)))),Trace)=true
$\rightarrow$ *in(encr(pair(nonce(Ri),nonce(Si)),key(E)),analz(Trace))=true*

member(sent(X,Y,encr(FinalPackage,nonce(Ri))),Trace)=true
$\rightarrow$*in(encr(FinalPackage,nonce(Ri)),analz(Trace))=true*)))

member(sent(X,Y,encr(pair(nonce(Si),h(Package)),f(Package))),Trace)=true
$\rightarrow$ *in(encr(pair(nonce(Si),h(Package)),f(Package)),analz(Trace))=true*)))

% spy faking a message 1

m(Trace)=true$\wedge$
in(pair(principal(MN),encr(key(E),key(Pass))),synth(analz(Trace)))=true
$\rightarrow$
m(cons(sent(spy,all,pair(principal(MN),encr(key(E),key(Pass)))),Trace))=true)))

% spy faking a message 2

m(Trace)=true$\wedge$
eqagent(spy,MN)=false$\wedge$
in(pair(principal(Mi),encr(pair(nonce(Ri),nonce(Si)),key(E))),synth(analz(Trace)))=true
$\rightarrow$
m(cons(sent(spy,MN,pair(principal(Mi),encr(pair(nonce(Ri),nonce(Si)),
key(E)))),Trace))=true

% spy faking a message 3

m(Trace)=true$\wedge$
eqagent(spy,Mi)=false$\wedge$
in(encr(FinalPackage,nonce(Ri)),synth(analz(Trace)))=true
$\rightarrow$
m(cons(sent(spy,Mi,encr(FinalPackage,nonce(Ri))),Trace))=true

% spy faking a message 4

m(Trace)=true∧
eqagent(spy,MN)=false∧
in(encr(pair(nonce(Si),h(Package)),f(Package)),synth(analz(Trace)))=true
→
m(cons(sent(spy,MN,pair(principal(Mi),encr(pair(nonce(Si),h(Package)),f(Package)))),Trace))=true

% spy knows some fresh nonces

in(nonce(N),parts(Trace)=false
→*in(nonce(N),analz(Trace))=true*

% to allow the spy to make fake compound keys
% the spy can break open a package and reassemble packages:

in(cons(nonce(X),Y),analz(Xset))=true
→ *in(nonce(X),analz(Xset))=true*

in(cons(nonce(X),Y),analz(Xset))=true
→ *in(Y,analz(Xset))=true*

in(cons(Y,z),synth(analz(Xset)))=true
→ *in(cons(nonce(X),cons(Y,z)),synth(analz(Xset)))=true*

in(nonce(X),synth(analz(Xset)))=true
→ *in(cons(nonce(X),nil),synth(analz(Xset)))=true*

Appendix C

The Model for the Tanaka–Sato Protocol

This appendix gives the clauses used for modelling the Taghdiri–Jackson version of the Tanaka–Sato protocol, described in Chapter 10.

% m function now takes args (Trace, Group, Keysequence, Tick)
% nobody in the group at the start

m(nil,nil,x,nought)=true

%%%% JOIN

m(Trace,Group,Keysequence,Tick)=true $\wedge$
ingroup(triple(principal(Mi),X,Y),Group,Xnewgroup)=false $\wedge$
agent(Mi)=true
$\rightarrow$ *m(cons(sent(Mi,server,*
encr(principal(Mi),longtermkey(Mi)),Group),Trace),
group,Keysequence,s(Tick))=true

%%% server acknowledges join, makes new key, sends key

m(Trace,Group,Keysequence,Tick)=true $\wedge$
member(sent(Y,server,encr(principal(Mi),longtermkey(Mi)),Tgroup),Trace)=true
$\wedge$ *agent(Mi)=true* $\wedge$
ingroup(triple(principal(Mi),X,Z),Group,Xnewgroup)=false
$\rightarrow$ *m(cons(sent(server,Mi,*

encr(pair(ik(Tick),key(s(Keysequence))),longtermkey(Mi)),
cons(triple(principal(Mi),ik(Tick),key(s(Keysequence))),Group)),Trace),
cons(triple(principal(Mi),ik(Tick),key(s(Keysequence))),Group),
s(Keysequence), s(Tick))=true

%%%% LEAVE

m(Trace,Group,Keysequence,Tick)=true $\wedge$
ingroup(triple(principal(Mi),Ikey,Gk),Group,Newgp)=true
$\rightarrow$ *m(cons(sent(Mi,server,encr(leave,Ikey),Group),Trace),Group,Keysequence,s(Tick))=true*

m(Trace,Group,Keysequence,Tick)=true $\wedge$
ingroup(triple(principal(Mi),Ikey,Gk),Group,Newgp)=true $\wedge$
member(sent(X,server,encr(leave,Ikey),Tgroup),Trace)=true
$\rightarrow$ *m(cons(sent(server,Mi,encr(ackleave,Ikey),Newgp),Trace),Newgp,s(Tick),s(Tick))=true*

%%%%% SEND a message

m(Trace,Group,Keysequence,Tick)=true $\wedge$
ingroup(triple(principal(Mi),Ikey,key(Sq)),Group,Newgp)=true
$\rightarrow$ *m(cons(sent(Mi,server,encr(send(Sq),Ikey),Group),Trace),Group,Keysequence,s(Tick))=true*

%% server gives key

m(Trace,Group,Keysequence,Tick)=true $\wedge$
ingroup(triple(principal(Mi),Ikey,Oldk),Group,Newgp)=true $\wedge$
member(sent(X,server,encr(send(Sq),Ikey),Tgroup),Trace)=true
$\rightarrow$ *m(cons(sent(server,Mi,encr(pair(key(Keysequence),send(Sq)),Ikey),*
Group),Trace),Group,Keysequence,s(Tick))=true

%% agent broadcasts his message, updates his key

m(Trace,Group,Keysequence,Tick)=true $\wedge$
ingroup(triple(principal(Mi),Ikey,oldk),Group,Newgp)=true $\wedge$
member(sent(X,Mi,encr(pair(key(Xk),send(Sq)),Ikey),Tg1),Trace)=true $\wedge$
member(sent(Mi,server,encr(send(Sq),Ikey),Tg2),Trace)=true
$\rightarrow$ *m(cons(sent(Mi,all,encr(hello(s(Tick)),key(xk)),*
cons(triple(principal(Mi),Ikey,key(xk)),newgp)),Trace),
cons(triple(principal(Mi),Ikey,key(xk)),newgp),Keysequence,
s(Tick))=true

%%%%% RECEIVE a message - sends request for key

m(Trace,Group,Keysequence,Tick)=true
ingroup(triple(principal(Mi),Ikey,Gk),Group,Newgp)=true
$\rightarrow$ *m(cons(sent(Mi,server,encr(read,Ikey),Group),Trace),*
Group,Keysequence,s(Tick))=true

%%% server gives key

m(Trace,Group,Keysequence,Tick)=true
ingroup(triple(principal(Mi),Ikey,Oldk),Group,Newgp)=true
member(sent(X,server,encr(read,Ikey),Somegp),Trace)=true
$\rightarrow$ *m(cons(sent(server,Mi,encr(key(Keysequence),Ikey),*
Group),Trace),Group,Keysequence,s(Tick))=true

%% agent updates his key

m(Trace,Group,Keysequence,Tick)=true $\wedge$
ingroup(triple(principal(Mi),Ikey,Oldk),Group,newgp)=true $\wedge$
member(sent(X,Mi,encr(key(Xk),Ikey),Somegp2),Trace)=true $\wedge$
member(sent(Mi,server,encr(read,Ikey),Somegp),Trace)=true
$\rightarrow$ *m(Trace,cons(triple(principal(Mi),Ikey,key(Xk)),Newgp),Keysequence,s(Tick))=true*

%%% Ingroup - third arg returns the group *without* the member if

%%% he's found.

ingroup(X,nil,Y)=false

ingroup(X,nil,Y)=true $\rightarrow$

$\rightarrow$ *ingroup(triple(principal(Mi),Ikey,Gk),*
cons(triple(principal(Mi),Ikey,Gk),Group),Group)=true

ingroup(Y,Gp,Rest)=true
$\rightarrow$ *ingroup(Y,cons(X,Gp),cons(X,Rest))=true*

eqagent(Mi,Mj)=false $\wedge$
ingroup(triple(principal(Mi),Ikey,Gk),Group,Rest)=false
$\rightarrow$ *ingroup(triple(principal(Mi),Ikey,Gk),*
cons(triple(principal(Mj),X,Y),Group),Rest)=false

Appendix D

Index of Terms

Here we give a reference to the definition of technical terms used in the thesis.

Term	Section	Page
Analz	6.6	81
Answer Literals	4.2.3	49
Authenticity	2.3.1	11
Binary Resolution	4.2	44
Bundle	2.4.5	27
Ciphertext	2.1	7
Clause	4.1	44
Completely Defined Equality Function	4.6	59
Compound Keys	9.2.2	117
Con¤uent	4.1	42
Congruence	4.1	42
Conjecture Superposition	4.4	53
Conjunctive Normal Form	4.2	43
Constructor	3.2	34
Contributory	9.1	112
Convergent	4.1	41
Decryption	2.1	8

Term	Section	Page	
Dictionary Attack	9.1	112	
Disruption Attacks	9.3	118	
Encryption	2.1	7	
Equality Factoring	4.2.1	45	
Events	2.4.5	27	
Factoring	4.2	43	
Fair Induction Derivation	4.4	52	Defn. 10
Free Constructors	4.6	59	Defn. 14
Freshness	2.2	9	
Given Clause	4.2	43	
Goal-Bindings	11.1.1	140	
Handshake	2.3.4	13	
Herbrand Model	4.1	42	
Horn Clause	4.1	42	
I-Axiomatisation	4.4	50	Defn. 4
Index (of a literal)	4.2	43	
Inductive Reducibility	4.6	59	
Irreducible	4.1	42	
Literal	4.2	44	
Local Rewriting	4.2.1	45	
Monotonic	4.1	42	
Multiset	4.1	41	
Next State Function	11.1.1	140	
Node	2.4.5	27	
Normal	4.4	51	Defn. 5
Normal I-Axiomatisation	4.4	51	Defn. 6
Normal Subsitution	4.4	51	Defn. 5
Ordering	4.1	43	
Paramodulation	4.2.1	45	
Parts	6.5	78	
Plaintext	2.1	7	

Term	Section	Page	
Proof By Consistency	4.3	50	
Public Key Cryptography	2.1	8	
recursive Path Ordering (RPO)	4.1	43	
Reduction Ordering	4.1	43	
Reduction Rules	4.2.1	45	
Reductive Definition	4.5	57	Defn. 12
Redundancy			
– general	4.2.1	45	Defn. 1 & 2
– in the Comon-Nieuwenhuis method	4.4	53	Defn. 7 & 8
Refutation Complete	4.2	44	
Replay Attack	2.3.4	13	
Resolution	4.2	44	
Resolvent	4.2	44	
Rewrite Rule	4.1	41	
Saturation			
– general	4.2.1	45	Defn. 3
– in the Comon-Nieuwenhuis method	4.4	53	Defn. 9
Secrecy	2.3.1	11	
Selection (of Literals)	4.2.1	45	
Semi-Bundles	11.1.1	140	
Session Key	2.1	7	
Set of Support	4.2	43	
Skolem Function	4.2	44	
State (in a strand space context)	11.1.1	140	
Step Compression	7.2.1	90	
Strand Space	2.4.5	27	
Strand	2.4.5	27	
Substitution	4.1	42	
Subsumption	4.2.1	45	

Term	Section	Page
Superposition	4.2.1	45
Superposition Left	4.2.1	45
Superposition Right	4.2.1	45
Symmetric Key Cryptography	2.1	8
Synth	6.6	81
Tautologies	4.2.1	45
Term Indexing	4.2.2	48
Term Rewrite System	4.1	42
Terminating	4.1	42
Timely	2.3.4	13
Type Attack	2.3.6	18
Usable	4.2	43
Well-Founded	4.1	41
Worked Off	4.1	41

Bibliography

[Abadi and Gordon, 1997] Abadi, M. and Gordon, A. (1997). A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press.

[Abadi and Needham, 1996] Abadi, M. and Needham, R. (1996). Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15.

[Anderson and Needham, 1995] Anderson, R. and Needham, R. (1995). *Computer Science Today: Recent Trends and Developments*, volume 1000 of *LNCS*, chapter Programming Satan's Computer, pages 426–440. Springer.

[Anderson and Roe, 1997] Anderson, R. and Roe, M. (1997). The GCHQ protocol and its problems. Available from Ross Anderson's webpage, `http://www.cl.cam.ac.uk/users/rja14/`.

[Asokan and Ginzboorg, 2000] Asokan, N. and Ginzboorg, P. (2000). Key-agreement in ad-hoc networks. *Computer Communications*, 23(17):1627–1637.

[Ateniese et al., 2000] Ateniese, G., Steiner, M., and Tsudik, G. (2000). New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–639.

[Avenhaus et al., 2003] Avenhaus, J., Kuehler, U., Schmidt-Samoa, T., and Wirth, C.-P. (2003). How to prove inductive theorems? quodlibet! In *19th Conference on Automated Deduction*, volume 2741 of *Lecture Notes in Computer Science*, pages 328–333. Springer.

[Bachmair, 1991] Bachmair, L. (1991). *Canonical Equational Proofs*. Birkhauser.

[Bachmair and Ganzinger, 1990] Bachmair, L. and Ganzinger, H. (1990). Completion of First-order clauses with equality by strict superposition (extended abstract). In *Proceedings 2nd International CTRS Workshop*, pages 162–180, Montreal, Canada.

[Bachmair and Ganzinger, 1994] Bachmair, L. and Ganzinger, H. (1994). Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247. Revised version of Research Report MPI-I-91-208, 1991.

[Barthe and Stratulat, 2003] Barthe, G. and Stratulat, S. (2003). Validation of the javacard platform with implicit induction techniques. In Nieuwenhuis, R., editor, *Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings*, volume 2706 of *Lecture Notes in Computer Science*, pages 337–351. Springer.

[Basin, 1999] Basin, D. (1999). Lazy infinite-state analysis of security protocols. In *Secure Networking — CQRE [Secure] '99*, number 1740 in Lecture Notes in Computer Science, pages 30–42, Düsseldorf, Germany. Springer-Verlag.

[Basin et al., 2003] Basin, D., Mödersheim, S., and Viganò, L. (2003). An on-the-¤y model-checker for security protocol analysis. In *Proceedings of the 2003 European Symposium on Research in Computer Security*, pages 253–270. Extended version available as Technical Report 404, ETH Zurich.

[Bella, 2003] Bella, G. (2003). Inductive verification of smart card protocols. *Journal of Computer Security*, 11(1):87–132.

[Bella et al., 2003] Bella, G., C.Longo, and Paulson, L. (2003). Verifying second-level security protocols. In Basin, D. and Wolff, B., editors, *Theorem Proving in Higher Order Logics*, number 2758 in LNCS, pages 352–366. Springer.

[Bella and Paulson, 1997] Bella, G. and Paulson, L. (1997). Using isabelle to prove properties of the kerberos authentication system. In *DIMACS Workshop on Design and Formal Veri£cation of Security Protocols*. Electronic pro-

ceedings available from `http://dimacs.rutgers.edu/Workshops/Security/program2/program.html`.

[Bella and Paulson, 2001] Bella, G. and Paulson, L. (2001). Mechanical proofs about a non-repudiation protocol. In Boulton, R. and Jackson, P., editors, *Theorem Proving in Higher Order Logics*, volume 2152 of *Lecture Notes in Computer Science*, pages 91–104.

[Bella et al., 2002] Bella, G., Paulson, L. C., and Massacci, F. (2002). The verification of an industrial payment protocol: the set purchase phase. In *ACM Conference on Computer and Communications Security*, pages 12–20.

[Berezin and Groce, 2001] Berezin, S. and Groce, A. (2001). *Athena Hacker's Manual*. CMU.

[Bertolotti et al., 2003] Bertolotti, I., Durante, L., Sisto, R., and Valenzano, A. (2003). Introducing commutative and associative operators in cryptographic protocol analysis. In *Proceedings of Formal Techniques of Networked and Distributed Systems - FORTE 2003*, LNCS, Berlin. Springer.

[Blanchet, 2002] Blanchet, B. (2002). From secrecy to authenticity in security protocols. In Hermenegildo, M. and Puebla, G., editors, *International Static Analysis Symposium (SAS'02)*, volume 2477 of *Lecture Notes in Computer Science*, pages 342–359, Madrid, Spain.

[Bond and Anderson, 2001] Bond, M. and Anderson, R. (2001). API level attacks on embedded systems. *IEEE Computer Magazine*, pages 67–75.

[Bouhoula and Rusinowitch, 1995] Bouhoula, A. and Rusinowitch, M. (1995). Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235.

[Boyer and Moore, 1979] Boyer, R. and Moore, J. (1979). *A Computational Logic*. Academic Press. ACM monograph series.

[Bundy et al., 1990] Bundy, A., van Harmelen, F., Horn, C., and Smaill, A. (1990). The Oyster-Clam system. In Stickel, M. E., editor, *10th International Conference on Automated Deduction*, pages 647–648. Springer-Verlag. Lecture Notes in Artificial Intelligence No. 449. Also available from Edinburgh as DAI Research Paper 507.

[Burrows et al., 1990] Burrows, M., Abadi, M., and Needham, R. (1990). A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36.

[Chevalier and Vigneron, 2002] Chevalier, Y. and Vigneron, L. (2002). Automated unbounded verification of security protocols. In Brinksma, E. and Larsen, K., editors, *Computer Aided Veri£cation, 14th International Conference*, volume 2404 of *Lecture Notes in Computer Science*, pages 324–337, Copenhagen, Denmark. Springer.

[Clark and Jacob, 1996] Clark, J. and Jacob, J. (1996). Attacking authentication protocols. *High Integrity Systems*, 1(5):465–474.

[Clark and Jacob, 1997] Clark, J. and Jacob, J. (1997). A survey of authentication protocol literature: Version 1.0. Available via `http://www.cs.york.ac.uk/jac/papers/drareview.ps.gz`.

[Cohen, 2000] Cohen, E. (2000). TAPS a first-order verifier for cryptographic protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 144–158, Cambridge, England.

[Cohen, 2003] Cohen, E. (2003). TAPS: The last few slides. In *Formal Aspects of Security*, volume 2629 of *Lecture Notes in Computer Science*, pages 183–190.

[Comon and Nieuwenhuis, 1998] Comon, H. and Nieuwenhuis, R. (1998). Induction = I-Axiomatization + First-Order Consistency. Technical report, Laboratoire Spécification et Vérification, Ecole Normale Supérieure de Cachan, Cachan, France. Presented as a talk to the 1998 Conference on Rewriting techniques and Applications, Tsukuba, Japan.

[Comon and Nieuwenhuis, 2000] Comon, H. and Nieuwenhuis, R. (2000). Induction = I-Axiomatization + First-Order Consistency. *Information and Computation*, 159(1-2):151–186.

[Compagna, 2002] Compagna, L. (2002). Private communication. via Email.

[Denning and Sacco, 1982] Denning, D. and Sacco, G. (1982). Timestamps in key distribution protocols. *Communications of the Association for Computing Machinery*, 24(8):533–536.

[Deplagne and Kirchner, 2001] Deplagne, E. and Kirchner, C. (2001). Induction as deduction modulo. Submitted. Available from `http://www.loria.fr/~ckirchne/`.

[Dershowitz and Jouannaud, 1990] Dershowitz, N. and Jouannaud, J.-P. (1990). Rewrite systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 243–320. Elsevier and MIT Press.

[Dolev and Yao, 1983] Dolev, D. and Yao, A. (1983). On the security of public key protocols. *IEEE Transactions in Information Theory*, 2(29):198–208.

[Donovan et al., 1999] Donovan, B., Norris, P., and Lowe, G. (1999). Analyzing a library of security protocols using casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*. Electronic proceedings available at `http://www.cs.bell-labs.com/who/nch/fmsp99/program.html`.

[Durgin et al., 1999] Durgin, N., Lincoln, P., Mitchell, J., and Scedrov, A. (1999). Undecidability of bounded security protocols. In Heintze, N. and Clarke, E., editors, *Proceedings of the Workshop on Formal Methods and Security Protocols — FMSP, Trento, Italy*. Electronic proceedings available at `http://www.cs.bell-labs.com/who/nch/fmsp99/program.html`.

[Fábrega et al., 1999] Fábrega, F., Herzog, J., and J., G. (1999). Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230.

[FIPS, 1977] FIPS (1977). Data encryption standard. Federal Information Processing Standards Publication. Reaffirmed 1988. Superseded by FIPS 46-2 (1993), FIPS 46-3 (1999).

[Ganzinger, 1999] Ganzinger, H., editor (1999). *Automated Deduction – CADE-16, 16th International Conference on Automated Deduction*, LNAI 1632, Trento, Italy. Springer-Verlag.

[Ganzinger and Stuber, 1992] Ganzinger, H. and Stuber, J. (1992). *Inductive theorem proving by consistency for £rst-order clauses*, pages 441–462. Teubner Verlag.

[Gollmann, 2000] Gollmann, D. (2000). On the verification of cryptographic protocols - a tale of two committees. In Schneider, S. and Ryan, P., editors, *Electronic Notes in Theoretical Computer Science*, volume 32. Elsevier.

[Gong, 1992] Gong, L. (1992). A security risk of depending on synchronized clocks. *Operating Systems Review*, 26(1):49–53.

[Gong and Syverson, 1998] Gong, L. and Syverson, P. (1998). Fail-stop protocols: An approach to designing secure protocols. *Dependable Computing for Critical Applications*, 5:79–100. IEEE Computer Society.

[Gordon and Jeffrey, 2001] Gordon, A. and Jeffrey, A. (2001). Authenticity by typing for security protocols. Technical Report MSR-2001-49, Microsoft Research.

[Gottlob and Leitsch, 1985] Gottlob, G. and Leitsch, A. (1985). On the efficiency of subsumption algorithms. *JACM*, 32(2):280–295.

[Green, 1969] Green, C. (1969). Theorem proving by resolution as a basis for question-answering systems. In Meltzer, B. and Michie, D., editors, *Machine Intelligence*, volume 4, pages 183–208. Edinburgh University Press.

[Hähnle et al., 1996] Hähnle, R., Kerber, M., and Weidenbach, C. (1996). Common syntax of the dfg-schwerpunktprogramm ”deduktion”. Interner Bericht 10/96, Universitt Karlsruhe. Current version available from `http://spass.mpi-sb.mpg.de/`.

[Heather et al., 2000] Heather, J., Lowe, G., and Schneider, S. (2000). How to prevent type ¤aw attacks on security protocols. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press.

[Huet and Hullot, 1982] Huet, G. and Hullot, J. (1982). Proofs by induction in equational theories with constructors. *Journal of the Association for Computing Machinery*, 25(2).

[Hutter and Sengler, 1996] Hutter, D. and Sengler, C. (1996). INKA: the next generation. In McRobbie, M. A. and Slaney, J. K., editors, *13th Conference on Automated Deduction*, pages 288–292. Springer-Verlag. Springer Lecture Notes in Artificial Intelligence No. 1104.

[Hwang et al., 1995] Hwang, T., Lee, N.-Y., Li., C.-H., Ko, M.-Y., and Chen, Y.-H. (1995). Two attacks on Neuman-Stubblebine authentication protocols. *Information Processing Letters*, 53:103–107.

[Ireland, 1996] Ireland, A. (1996). Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1-2):79–111.

[Jackson, 2002] Jackson, D. (2002). Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290.

[Jacquemard et al., 2000] Jacquemard, F., Rusinowitch, M., and Vigneron, L. (2000). Compiling and verifying security protocols. `http://www.loria.fr/~rusi/pub/lpar2000.ps.gz`.

[Jouannaud and Kounalis, 1989] Jouannaud, J.-P. and Kounalis, E. (1989). Proof by induction in equational theories without constructors. *Information and Computation*, 82(1).

[Kessler and Wedel, 1994] Kessler, V. and Wedel, G. (1994). Autolog- an advanced logic of authentication. In *Proc. IEEE Computer Security Foundations Workshop IV*, pages 90–99. IEEE.

[Kindred, 1999] Kindred, D. (1999). *Theory Generation for Security Protocols*. PhD thesis, Carnegie Mellon University.

[Letz and Stenz, 2001] Letz, R. and Stenz, G. (2001). Model elimination and connection tableau procedures. In Robinson, A. and Voronkov, A., editors, *Handbook of Automated Reasoning*, volume II, chapter 28, pages 2015–2114. Elsevier Science.

[Lowe, 1995] Lowe, G. (1995). An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133.

[Lowe, 1996] Lowe, G. (1996). Breaking and fixing the Needham Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer Verlag.

[Lowe, 1997] Lowe, G. (1997). A hierarchy of authentication specifications. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*.

[Lowe, 1999] Lowe, G. (1999). Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2,3):89–146.

[L.Vigneron, 1996] L.Vigneron (1996). Positive deduction modulo regular theories. In Büning, H. K., editor, *Computer Science Logic, 9th International Workshop*, volume 1092 of *Lecture Notes in Computer Science*, pages 468–485. Springer.

[Lynch, 1997] Lynch, C. (1997). Oriented equational logic programming is complete. *Journal of Symbolic Computation*, 23(1):23–45.

[Mao and Boyd, 1993] Mao, W. and Boyd, C. (1993). Towards formal analysis of security protocols. In *Proceedings of the Computer Security Foundationn Workshop VI*, pages 147–158.

[McCune, 1994] McCune, W. (1994). A Davis Putnam program and its application to finite first order model search. Technical report, Argonne National Laboratory.

[Meadows, 1996a] Meadows, C. (1996a). Formal verification of cryptographic protocols: A survey. In *Proceedings of Asiacrypt 96*.

[Meadows, 1996b] Meadows, C. (1996b). The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131.

[Meadows, 2000a] Meadows, C. (2000a). Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In Degano, P., editor, *Proceedings of the First Workshop on Issues in the Theory of Security*, pages 87–92, Geneva, Switzerland.

[Meadows, 2000b] Meadows, C. (2000b). Invariant generation techniques in cryptographic protocol analysis. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press.

[Millen et al., 1987] Millen, J., Clark, S., and Freedman, S. (1987). The interrogator: Protocol security analysis. *IEEE Transactions Software Engineering*, 13(2):274–288.

[Millen and Shmatikov, 2003] Millen, J. and Shmatikov, V. (2003). Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *IEEE Computer Security Foundations Workshop*.

[Milner, 1993] Milner, R. (1993). The polyadic pi-calculus: a tutorial. In Bauer, F. L., Brauer, W., and Schwichtenberg, H., editors, *Logic and Algebra of Speci£cation*, pages 203–246. Springer-Verlag. Also available as Technical Report ECSLFCS-91-180, Computer Science Department, University of Edinburgh, UK, October 1991.

[Mittra, 1997] Mittra, S. (1997). Iolus: A framework for scalable secure multicasting. In *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 277–288, Cannes, France.

[Monroy, 1993] Monroy, R. (1993). The Use of Abduction to Correct Faulty Conjectures. Master's thesis, University of Edinburgh.

[Monroy, 2000] Monroy, R. (2000). The use of abduction and recursion-editor techniques for the correction of faulty conjectures. In Ledru, Y., editor, *15th Conference on Automated Software Engineering*, Grenoble, France.

[Monroy, 2003] Monroy, R. (2003). Predicate synthesis for correcting faulty conjectures: The proof planning paradigm. In *Automated Software Engineering*, pages 247–269.

[Musser, 1980] Musser, D. (1980). On proving inductive properties of abstract data types. In *Proceedings 7th ACM Symp. on Principles of Programming Languages*, pages 154–162. ACM.

[Needham and Schroeder, 1978] Needham, R. and Schroeder, M. (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999.

[Needham and Schroeder, 1987] Needham, R. and Schroeder, M. (1987). Authentication revisited. *Operating Systems Review*, 21(7):7.

[Nessett, 1990] Nessett, D. (1990). A critique of the Burrows, Abadi, and Needham logic. *Operating Systems Review*, 24(2):35–38.

[Neuman and Stubblebine, 1993] Neuman, B. and Stubblebine, S. (1993). A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10–14.

[Nivela, 1993] Nivela, P. amd Nieuwenhuis, R. (1993). Practical results on the saturation of full first-order clauses: Experiments with the saturate system. In Kirchner, C., editor, *5th International Conference on Rewriting Techniques and Applications*, number 690 in LNCS, Montreal, Canada. Springer-Verlag.

[Otway and Rees, 1987] Otway, D. and Rees, O. (1987). Efficient and timely mutual authentication. *Operating Systems Review*, 21(7):8–10.

[Paulson, 1989] Paulson, L. (1989). The foundation of a generic theorem prover. *JAR*, 5:363–397.

[Paulson, 1998] Paulson, L. (1998). The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128.

[Paulson, 1997] Paulson, L. C. (1997). Mechanized proofs for a recursive authentication protocol. In *10th Computer Security Foundations Workshop*, pages 84–95.

[Pelletier et al., 2002] Pelletier, F. J., Sutcliffe, G., and Suttner, C. (2002). The development of casc. *AI Communications*, 15(2):79–90.

[Pereira and Quisquater, 2001] Pereira, O. and Quisquater, J.-J. (2001). Security analysis of the cliques protocols suites: 1st results. In *In Proceedings of IFIP Sec'01*, pages 151–166.

[Pereira and Quisquater, 2003] Pereira, O. and Quisquater, J.-J. (2003). Some attacks upon authenticated group key agreement protocols. *Journal of Computer Security*, 11(4):555–580. Special Issue: 14th Computer Security Foundations Workshop (CSFW14).

[Protzen, 1992] Protzen, M. (1992). Disproving conjectures. In Kapur, D., editor, *11th Conference on Automated Deduction*, pages 340–354, Saratoga Springs, NY, USA. Published as Springer Lecture Notes in Artificial Intelligence, No 607.

[Reif, 1995] Reif, W. (1995). The KIV Approach to Software Verification. In M. Broy and S. Jähnichen, editors, *KORSO: Methods, Languages and Tools for the Construction of Correct Software*, volume 1009. Springer Verlag.

[Reif et al., 2000] Reif, W., Schellhorn, G., and Thums, A. (2000). Fehlersuche in formalen Spezifikationen. Technical Report 2000-06, Fakultät fur Informatik, Universität Ulm, Germany. (In German).

[Reif et al., 2001] Reif, W., Schellhorn, G., and Thums, A. (2001). Flaw detection in formal specifications. In *IJCAR'01*, pages 642–657.

[Rivest et al., 1978] Rivest, R., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21:120–126.

[Robinson and Wos, 1969] Robinson, G. and Wos, L. (1969). Paramodulation and theorem-proving in first-order theories with equality. In Michie, D. and Meltzer, R., editors, *Machine Intelligence*, volume IV, pages 135–150. Edinburgh University Press.

[Robinson, 1965] Robinson, J. (1965). A machine-oriented logic based on the resolution principle. *JACM*, 12(1).

[Schumann, 1997] Schumann, J. (1997). Automatic verification of crytographic protocols with SETHEO. In McCune, W., editor, *CADE14 - Proceedings of the 14th International Conference on Automated Deduction*, Australia.

[Slaney, 1995] Slaney, J. (1995). *FINDER: Finite Domain Enumerator*. Australian National University. Available from `ftp://arp.anu.edu.au/pub/papers/slaney/finder/finder.ps.gz`.

[Song et al., 2001] Song, D., Berezin, S., and Perrig, A. (2001). Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74.

[Stajano and Anderson, 1999] Stajano, F. and Anderson, R. (1999). The cocaine auction protocol: On the power of anonymous broadcast. In Pfitzmann, A., editor, *Proceedings of Information Hiding Workshop*, Dresden, Germany.

[Steel et al., 2002a] Steel, G., Bundy, A., and Denney, E. (2002a). Finding counterexamples to inductive conjectures and discovering security protocol attacks. In *Proceedings of the Foundations of Computer Security Workshop*. Appeared in Proceedings of The Verify'02 Workshop as well. Also available as Informatics Research Report EDI-INF-RR-0141.

[Steel et al., 2002b] Steel, G., Bundy, A., and Denney, E. (2002b). Finding counterexamples to inductive conjectures and discovering security protocol attacks. *The AISB Journal*, 1(2).

[Steel et al., 2003a] Steel, G., Bundy, A., and Denney, E. (2003a). Using the CORAL system to discover attacks on security protocols. In Herbert, A. and Jones, K. S., editors, *Computer Systems: Theory, Technology and Applications, Papers for Roger Needham*. Springer.

[Steel et al., 2003b] Steel, G., Bundy, A., and Maidl, M. (2003b). Attacking the Asokan–Ginzboorg protocol for key distribution in an ad-hoc bluetooth network us-

ing CORAL. In *Proceedings of FORTE 2003 (work in progress papers). Available from* `http://homepages.inf.ed.ac.uk/s9808756/papers/`, Berlin.

[Syverson et al., 2000] Syverson, P., Meadows, C., and Cerversato, I. (2000). Dolev-Yao is no better than Machiavelli. In Degano, P., editor, *Proceedings of the First Workshop on Issues in the Theory of Security*, pages 87–92, Geneva, Switzerland.

[Taghdiri and Jackson, 2003] Taghdiri, M. and Jackson, D. (2003). A lightweight formal analysis of a multicast key management scheme. In *Proceedings of Formal Techniques of Networked and Distributed Systems - FORTE 2003*, LNCS, pages 240–256, Berlin. Springer.

[Tanaka and Sato, 2001] Tanaka, S. and Sato, F. (2001). A key distribution and rekeying framework with totally ordered multicast protocols. In *Proceedings of the 15th International Conference on Information Networking*, pages 831–838.

[Walsh, 1996] Walsh, T. (1996). A divergence critic for inductive proof. *Journal of Arti£cial Intelligence Research*, 4:209–235.

[Weidenbach, 1999] Weidenbach, C. (1999). Towards an automatic analysis of security protocols in first-order logic. In [Ganzinger, 1999], pages 314–328.

[Weidenbach, 2001] Weidenbach, C. (2001). Combining superposition, sorts and splitting. In Robinson, A. and Voronkov, A., editors, *Handbook of Automated Reasoning*, volume II, chapter 27, pages 1965–2013. Elsevier Science.

[Weidenbach et al., 1999] Weidenbach, C. et al. (1999). System description: SPASS version 1.0.0. In [Ganzinger, 1999], pages 378–382.