Scyther - Semantics and Verification of Security Protocols

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 6 november 2006 om 14.30 uur

door

Casimier Joseph Franciscus Cremers

geboren te Geleen

Dit proefschrift is goedgekeurd door de promotor:

prof.dr. J.C.M. Baeten

Copromotoren:
dr. S. Mauw
en
dr. E.P. de Vink

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Cremers, Casimier Joseph Franciscus

Scyther – semantics and verification of security protocols /
door Casimier Joseph Franciscus Cremers. –
Eindhoven : Technische Universiteit Eindhoven, 2006.
Proefschrift. – ISBN 90-386-0804-7. – ISBN 978-90-386-0804-4
NUR 993
Subject headings: computer networks ; protocols / semantics / programming ; formal methods / software verification / computer networks ; information security / model checking / internet
CR Subject Classification (1998): C.2.2, F.3.2, D.2.4

IPA Dissertation Series 2006-20

Printed by University Press Facilities, Eindhoven

Cover photograph and cover design by C.J.F. Cremers

Netherlands Organisation for Scientific Research

The work on this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The author was employed at the Eindhoven University of Technology and supported by the Dutch Organization for Scientific Research (NWO) within the project "Security Analysis for Multi-Applet Smart Cards" (SAMASC), grant number 612.064.102.

Preface

I first got into contact with computers when I was 10 years old. At a friend's house I saw a Philips MSX-2 home computer, for which one could write programs in MSX-Basic. The thought of making the computer do something I wanted instantly attracted me, even more so than the games you could play on it. I nagged my parents to buy me one. Because they did not immediately give in, I decided to proceed and borrowed a book on Basic in the local library[1]. When I was 11, a Dutch broadcasting company wrote out bi-weekly programming contests. Although I did not have a computer, I participated anyway. I sent in my solution and, much to everyone's surprise, won the contest. I remember receiving for this a book called "Fatherhood" by B. Cosby, much to my disappointment.

The highly anticipated home computer arrived when I was 13, and I moved from Basic to machine code. I wrote my first game in machine code somewhat later, with which I won a much bigger MSX contest at age 15. From that point on, my high school years were dominated by designing programs during the day (i. e. during school) and programming them in the evening, as much as I was allowed. When I finished high school, my goal was clear. I did not want to do more programming. Rather, I wanted to learn more about the theoretical foundations of computing, and decided to study Computer Science at the university.

My first year at the university went smoothly and in an optimistic mood I decided to simultaneously study philosophy at another university, from my second year on. As the interest in MSX slowly waned, it was replaced by another upcoming and irresistible trend: the internet. Not realising I already had enough on my hands, I also started an internet company. As a result I could not put sufficient time in any of my interests to make them work, causing huge delays. I decided to focus on theoretical computer science and finished my Masters degree. Having chosen a single path I felt committed enough to start a Ph.D. in the Formal Methods group at the university.

During the first year of my Ph.D. studies, I read papers and brainstormed about anything that seemed remotely interesting. My supervisors, Sjouke Mauw and Erik de Vink, tried (and for the most part, managed) to guide me into interesting directions. One of the directions was trying to understand what the essential elements of security protocols were, as the literature immediately seemed to jump to computational

[1] Like a part of the internet, but printed on paper.

models (and conclusions), skipping any foundations. This triggered the development of the formal security protocol model described in this thesis. Development of a new protocol model meant reinventing some existing security properties, and as a side result new ones were even invented, such as a new form of authentication.

Although the developed model seemed to work on paper, validation was needed. Manual verification of protocols with known properties seemed to work just fine, so the next step was to develop tools and really put the theory to the test. After an initial attempt by a student, I developed a prototype tool. Tests with this tool seemed to validate the theory, and I could reproduce results regarding protocols from the literature. Furthermore the performance of the tool was much better than the other security tools I had access to at that point, even though this had not been the original motivation. Meanwhile, reading more literature revealed that there was a new security protocol tool, of which a very high performance was claimed. Although this tool was not available to me to actually confirm this performance, the method that was described in the paper seemed very similar to what my manual proofs looked like. This lead to the development of a new algorithm, consisting of a blend of the model already developed, and the method presented in the paper. When the new tool prototype was completed, the performance was unlike anything that I had seen thus far. This opened the way to applying the tool to case studies which had never been done before, such as investigating compositionality problems. From this point on, there were more interesting routes open than I had time to write papers about. There are still many half-developed ideas which are not included in this thesis, but I hope to flesh out in the near future.

Since my initial interest in computers years ago, I have completed many side quests. I enjoyed all these thoroughly, learnt a lot from them, and I dare say they have enriched me as a person. But, in the end, it seems there is no escape, and I always find myself returning to computer science. It is time to continue on the main quest.

Acknowledgements

For the last four years, I owe many people thanks. Not only because people taught me things, but also because of their support, as well as the great working atmosphere.

I spent by far the most time in discussion with Sjouke Mauw, which was always a great pleasure. From Sjouke I learnt many things (except for juggling), for which I am very grateful. Without his influence this thesis would have looked completely different (i. e. worse) and would probably have progressed less smoothly. Sjouke's faith in me helped me to accomplish things I might not have achieved otherwise. A lot of time I also spent with Erik de Vink, who would always strive for technical perfection. Typically, Erik asked the right questions which forced me to make my thoughts comprehensible for others. I thank them both for allowing me the final sprint of these last six months, which has probably been fairly horrific for all parties involved. The warriors are still standing! (Just don't nudge them, because they might fall over.)

I owe thanks to my roommates, first Jerry den Hartog, and later Hugo Jonker, for putting up with my rants, interesting working hours, and plenty of coffee breaks. Jerry was always a great first filter for my brainstorming sessions during the first few years, and his precision helped clarify problem areas. Later, I thoroughly enjoyed Hugo's cheerful mood and his amazing ability to improve himself. Thanks guys!

That I enjoyed my Ph.D. studies so much was also a joint effort of the FM group members. For this, the main responsible person is surely Jos Baeten, who runs the Formal Methods group in his very unique way, making it seem like an easy job(s). This has been very inspiring. I especially appreciated the way in which everybody was always ready to give a helping hand. This included advice on various topics, from paper writing to (university) politics, as well as simply helping me to arrange things during times of pressure. For this, I owe all FM-buckets.

Regarding this thesis, I thank Rob Nederpelt for improving the Dutch summary. I also thank the students who assisted during the years on case studies, prototypes and other Scyther-related investigations: Gijs Hollestelle, Lutger Kunst, Niek Palm, and Ingmar Schnitzler.

I also want to thank Ricardo Corin and Martijn Warnier. Together we started the association of Security Ph.D. students in the Netherlands (SPAN). The relaxed atmosphere at the SPAN meetings ensured that we always had a great time. Martijn and I also had some great meetings outside of work, at which we would contemplate the future whilst enjoying a drink, until it was time to catch the last train.

I thank the reading committee, prof.dr. D.A. Basin, prof.dr. W.J. Fokkink and prof.dr. B.P.F. Jacobs for assessing this thesis. Their insightful comments have helped to improve this thesis. I also thank dr. S. Etalle for partaking in my promotion committee.

I owe my parents, Fred and José, and my brother Koen, for unconditionally supporting me through easy, and less easy times. I could not wish for anything more.

Finally, writing this thesis would never have been possible without the ever loving support of my wife. Given my tendency to work interesting (i. e. all) hours, my great concentration skills (i. e. neglecting everything else), and relaxed and optimistic way of planning (i. e. requiring insane schedules around deadlines for everything), this certainly is no easy task. Yen-Ha: thank you for everything.

Contents

Preface **i**

1 Introduction **1**

1.1 Historical context 1

1.2 Black box security protocol analysis 4

1.3 Research question 6

1.4 Overview of the thesis 7

2 Operational Semantics **9**

2.1 Domain analysis 9

2.2 Security protocol specification 12

2.2.1 Role terms 13

2.2.2 Event order 17

2.2.3 Static Requirements 18

2.3 Describing protocol execution 21

2.3.1 Runs 21

2.4 Threat model 27

2.4.1 Network threat model 28

2.4.2 Agent threat model 29

2.5 Conclusions 31

3 Security Properties **33**

3.1 Security properties as claim events 33

3.2 Secrecy 35

3.3 Authentication 36

3.3.1 Aliveness . . . 36
3.3.2 Synchronisation . . . 38
3.3.3 Message agreement . . . 44
3.4 Authentication hierarchy . . . 47
3.5 Verifying injective synchronisation . . . 54
3.5.1 Injectivity of synchronisation . . . 55
3.5.2 The *LOOP* property . . . 59
3.6 Proving security properties of the NS/NSL protocol . . . 62
3.7 Conclusions . . . 66

4 Verification **69**
4.1 Trace patterns . . . 70
4.1.1 Representing classes of traces by trace patterns . . . 70
4.1.2 Realizable trace patterns . . . 73
4.1.3 Explicit trace patterns . . . 77
4.1.4 Complete characterization . . . 79
4.2 Characterization algorithm . . . 83
4.2.1 Basic idea . . . 83
4.2.2 Optimization: decryption chains . . . 84
4.2.3 Algorithm preliminaries . . . 85
4.2.4 Algorithm . . . 88
4.2.5 Termination . . . 92
4.3 Verification of security properties by means of characterization . . . 93
4.3.1 Verifying secrecy properties . . . 94
4.3.2 Verifying authentication properties . . . 94
4.3.3 Correctness and complete characterizations . . . 95
4.4 Prototype implementation: Scyther . . . 95
4.4.1 Requirements and design . . . 95
4.4.2 Implementation . . . 96
4.4.3 Validation . . . 98
4.4.4 Choosing a heuristic . . . 99
4.4.5 Choosing a bound on the number of runs . . . 102
4.5 Conclusions . . . 104

5 Multi-Protocol Attacks **107**
5.1 Multi-protocol attacks . . . 108
5.2 Experiments . . . 109
5.3 Results . . . 110
5.4 Attack scenarios . . . 114
5.5 Preventing multi-protocol attacks . . . 118
5.6 Conclusions . . . 121

6 Generalizing NSL for Multi-Party Authentication **123**
6.1 A multi-party authentication protocol . . . 125
6.2 Analysis . . . 126
6.2.1 Properties of generalized NSL . . . 126
6.2.2 Proof of correctness . . . 127
6.2.3 Observations . . . 134
6.2.4 Message minimality . . . 135
6.3 Variations on the pattern . . . 136
6.4 Attacks on two previous multi-party authentication protocols . . . 138
6.5 Conclusions . . . 140

7 Related Work **143**
7.1 Security protocol models . . . 143
7.1.1 Current models . . . 143
7.1.2 Modeling security properties . . . 146
7.1.3 Complete characterization . . . 148
7.2 Protocol analysis tools . . . 149
7.3 Other related work . . . 153

8 Conclusions and Future Work **155**
8.1 Conclusions . . . 155
8.2 Summary of contributions . . . 156
8.3 Future work . . . 158

Bibliography **161**

Index of subjects **177**

Samenvatting **181**

Summary **183**

CV **184**

1

Introduction

In this thesis we investigate the black-box analysis of security protocols. We start off by giving a brief historical context in Section 1.1. We then describe black-box analysis of security protocols in Section 1.2, and state some of the drawbacks of current methods. The research question addressed in this thesis is formulated in Section 1.3. In Section 1.4 we give an overview of the thesis.

1.1 Historical context

This thesis is not about cryptography.

Cryptography, or the art of "secret writing", dates back to about 600 BC, when Hebrew scholars used the so-called Atbash cipher. In order to encode the word `slob`, they would encode each character individually, by reversing the order of the alphabet: `a` is swapped with `z`, `b` is swapped with `y`, andsoforth. Thus, the word `slob` would be encoded as `holy`, and vice versa. As long as nobody discovers your scheme, the fact that you are writing about a slob is secret.

Around 400 BC, the Spartans allegedly used a *Scytale* for encryption, which can be considered the first *device* used for encryption. In truth, the Scytale is just a rod with a random diameter. The idea is that both the sender and recipient know what diameter the rod should have. The sender wraps a long thin piece of paper (or, according to legend, a belt) around the rod, and writes his message from left to right on the paper, as in Figure 1.1.

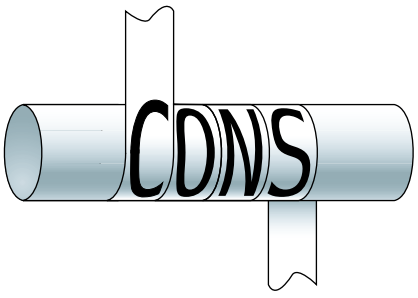

Figure 1.1: Scytale

If we want to send a secret message `consoles`, and the diameter of the rod is such that two characters can be written around the rod, we write `c o n s` along the front side of the rod, as in Figure 1.1 on the previous page. We turn the rod and write the remaining characters `o l e s`. Now, if we unwrap the paper from the rod, and read the characters, we find that the encrypted message is `coolness`, as can be seen in Figure 1.2.

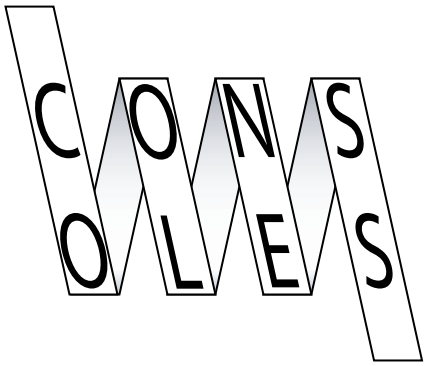

Figure 1.2: Unfolded Scytale

If the recipient wants to decode it, he wraps the paper again around a rod of the same diameter. If he uses a wrong diameter, e.g. such that three characters can be written around the rod, the message will read `clsonsoe`, and he will not receive the correct message. Here, the diameter of the rod acts as a *key*. Even when the encryption method is known to the enemy, encrypting and decrypting requires a key which is only known by the sender and the recipient.

Throughout history, the use of encoding schemes has evolved significantly. Traditional encryption relies on the fact that an encoding scheme is kept secret. This design principle is sometimes referred to as "security through obscurity", and is still often applied today.

It is possible to design stronger encryption schemes. In the 19th century, Auguste Kerckhoff stated that an encryption scheme should be secure even if everything except the key is known to the enemy. This is often referred to as Kerckhoff's law. In the 20th century Claude Shannon formulated a similar notion, stating that "the enemy knows the system", which is referred to as Shannon's maxim.

Just before and during the second world war, cryptography was used extensively, also by the German forces. Although different machines were used, the most well-known are the Enigma devices, based on a system with multiple rotors. Initial attempts to crack the code started as early as 1932 by Polish researchers. Based on their findings, by the end of the war a team of researchers in Bletchley park (which included Alan Turing) was able to break coded messages on a daily basis. This result was not due to the fact that they had recovered German Enigma machines which they could analyse. Rather, it involved the development of dedicated code-breaking machines (which took several years) that tried to recover the key from the

encrypted text. Thus, to some extent, the German Enigma machines were secure in the sense of Kerckhoff's law.

From 1949 onwards, when Claude Shannon published his seminal paper on information theory, many scientific publications have appeared on the topic of cryptography, and for a large part these focus on finding new encryption schemes. These encryption schemes were no longer just inventions with claims, but they were developed on a more proper mathematical foundation: cryptographic schemes are derived from known hard mathematical problems. To prove the security of a scheme, one proves that if an adversary could crack the scheme, he would also be able to solve a mathematical problem, which is known to be very difficult.

In 1976 Diffie and Hellman publish their key paper in which they introduce a mechanism that later became known as asymmetrical encryption. We can give a somewhat imprecise metaphor to provide intuition about this scheme: the Diffie Hellman encryption scheme allows everybody to create their own particular padlock with corresponding key. Suppose a woman called Alice wants to securely receive messages which she only can read. She creates a padlock and key. Instead of having to securely transmit the key, she keeps the key to herself, and simply hands out copies of the (open) padlock to everybody else. If her brother, Bob, wants to send a secret message to her, he puts it in a box, closes the box using Alice's padlock. Now nobody except for Alice will be able to read the message. This scheme circumvents the problem with traditional symmetrical cryptography, where both the sender and the recipient must have the same key.

This breakthrough has resulted in a number of variations on these schemes, leading to international standards for encryption mechanisms. Both symmetric and asymmetric encryption schemes are used extensively today for the encryption of internet traffic, wireless communications, smart-card applications, cell phone communications, and many other applications.

From this brief historical account, one might conclude that a secure encryption scheme is the "holy grail" of communication security. Once the perfect scheme is found, all communications will be safe, and we don't ever have to worry again. Unfortunately, this is not the case. Cryptography is not enough to guarantee security in communications. And that is why this thesis is not about cryptography.

1.2 Black box security protocol analysis

If you have the perfect bike lock and chain, but fix the chain around the bike in the wrong way, somebody can still steal your bike. In much the same way, the security of a computer system depends on the way the components interact. Encryption is like a bike chain. It is a useful mechanism that can be used in the construction of secure systems, but you can apply it in the wrong way. It is not a guarantee for security by itself.

Security protocols are a means to ensure some form of secure communication, and they usually try to establish this by using some form of encryption. Security protocols underly most of our current communication systems, such as secure internet communications, cell phone networks, as well as the communication between credit cards, ATM machines, and banks. For these applications, it is crucial that no malicious party can disturb the intended workings of the protocol, or eavesdrop on something he was not supposed to hear. It is not sufficient to have a security protocol of which one is pretty sure that it is secure. Instead, we want some guarantees about its security.

In order to make a statement about the security guarantees of a protocol, we turn again to mathematics. We create a mathematical model of both the protocol and the network, which is under control of an adversary. Such a model allows us to prove e. g. that the adversary is not able to disturb the protocol or learn any secrets. As these models already become complex for simple encryption schemes, it is only feasible to reason about the security of full protocols by abstracting away from some (cryptographic) details. Consequently, the need to reason about security protocols has led to the introduction of an idealized abstraction of encryption in 1983 by Dolev and Yao [79], with two main properties. First, cryptography is assumed to be perfect: a message can only be decrypted by somebody who has the right key (there is no way to crack the scheme). Second, messages are considered to be abstract terms: either the intruder learns the complete message (because he has the key), or he learns nothing. We call analysis models based on these abstractions *black box*, in the sense that they consider encryptions as abstract functions with some particular properties. Instead of modeling all cryptographic details and properties, we assume that somebody already has invented a perfect cryptographic scheme, which we can use in building secure protocols.

Next to the two assumptions about cryptography, Dolev and Yao introduced a third abstraction concerning computer networks. The network is assumed to be under full control of the adversary. He can remove sent messages and examine their

contents, insert his own messages, or reroute or simply transmit messages. Together, these three properties are known as the Dolev-Yao model: cryptography is perfect, messages are abstract terms, and the network is under full control of the intruder.

Given a security protocol, it is possible to develop mathematical techniques to derive security properties of a protocol under the Dolev-Yao assumptions. As a result, the work of Dolev and Yao has been responsible for a branch of research which can be roughly summarized as *black-box security protocol analysis*. However, building the three properties sketched above into a precise mathematical model, with clear assumptions, and clearly defined security properties, has proven to be a hazardous task.

The defining example for this area of research illustrates some of the subtleties. It concerns the so-called Needham-Schroeder public key protocol from [145], published in 1978, five years before the abstract model of Dolev and Yao. The basic version of this protocol consists of three messages that are exchanged between two partners. It is intended to provide *authentication* for both agents. It was assumed to be correct for over 20 years; now it is or is not correct, depending on the situation in which it is used. The reason for this change is not to be found in more powerful analysis methods. Instead, it is the assumptions about the intruder that have changed.

In 1989 Burrows, Abadi and Needham published their ground breaking paper [44] on a logic for authentication (which became known as BAN logic), which also depends on the same black box assumptions as the Dolev-Yao model. Using this logic, they were able to prove that several protocols satisfy a form of authentication.[1] Among these protocols was the Needham-Schroeder public key protocol. Thus, it seemed that the (intuitively correct) protocol was now formally proven correct. A version of the protocol made its way into the Kerberos [20] protocol. Almost twenty years after the publication of the Needham-Schroeder protocol, in 1996, Gavin Lowe claimed to have found an attack on the protocol. It turns out that Lowe's attack required a stronger intruder than the one Dolev and Yao originally had in mind. Around 1980, networks were considered to be something that was used by honest users: attacks would come from the outside. Thus, the intruder was not one of the regular users of the system. During the nineties, this view of networks changed. Now, many large networks were used by users, which were not necessarily trusted. Lowe's attack requires that the intruder is, or has compromised, a regular user. As a result, the intruder model was modified, and the intruder was assumed from now on to control a number of regular users of the system.

In the same paper, Lowe introduced a mechanized procedure to find attacks on such protocols. A high level description of the protocol is processed by a program called Casper, which models the behaviour of the protocol and the possible operations of the intruder in terms of a process algebra. Similarly, the security properties of the protocol are translated into a second set of processes that model the ideal behaviour of the system, which would occur if the security properties are satisfied.

[1] In retrospect the main contribution of this paper seems to be that the logic made some of the Dolev-Yao assumptions explicit, and gave a possible mathematical definition of the notion of authentication.

The Casper tool uses a model-checking tool for this process algebra, to verify whether the actual protocol model has the same set of behaviours as the ideal behaviour model. If these behaviours are the same, the intruder has no real influence over the protocol execution. Using this procedure, Lowe was able to automatically find the attack on the Needham-Schroeder protocol, and was also able to show that no such attack exists on a repaired version of the protocol, which later became known as the Needham-Schroeder-Lowe protocol.

After Lowe's breakthrough, a large number of security protocol formalisms and tools have been developed. We discuss some of them in Chapter 7. On the one hand, many security protocol formalisms focus solely on protocol description, and there are no tools available for them that are based on a formal semantics. On the other hand, most of the tools have only implicit links with formalisms, and lack formal definitions of the model and properties that are actually being verified. This makes it very difficult to interpret the results.

These observations lead to the research question addressed in this thesis.

1.3 Research question

The goal of this thesis is to investigate how to combine a formal semantics for security protocols with semantically correct verification, in the sense that the verification results correspond to the formal protocol semantics.

Research question: *How to integrate developments in state-of-the-art black-box security protocol analysis into a formal semantics with tool support?*

By state-of-the-art we refer to recent advancements in the efficiency of protocol verification, in which protocols such as the Needham-Schroeder protocol can be verified or falsified within seconds. By black-box we refer to the Dolev-Yao style of formal analysis. We want to combine a formal semantics, with intuitive formal definitions of security properties, with efficient tool support.

We start off by developing an explicit foundation in terms of an operational semantics, which allow us to formally model security protocols, and define all possible behaviours of such protocols. Next, we provide formal definitions of existing and new security properties. The combination of formal definitions of protocol behaviour and security properties allows us to verify whether a property holds. We develop a fully automatic method to verify or falsify security properties and implement this in a tool set. We apply the automatic tools to establish results about the interaction between security protocols. We further illustrate the manual application of the security protocol formalism by developing a family of multi-party protocols.

Our research covers the full range of the analysis methods, from developing a formal model describing protocol analysis, to the development of verification tools, and applications of both the model and tools to case studies.

1.4 Overview of the thesis

We briefly sketch the organization of the thesis, and summarize the contributions made in each chapter.

Chapter 2: Operational Semantics

In Chapter 2 we present a new security protocol model for defining security protocols and their behaviour. We make protocol execution in this model explicit by means of an operational semantics. The result is a role-based security protocol model that is agnostic with respect to the number of concurrent protocols. The model makes several assumptions about protocol analysis explicit, and allows e.g. for a derivation of the intruder knowledge from the protocol description. Within the protocol model, security properties are modeled as local claim events. This chapter is based on [65].

The contribution made in this chapter consists of the development of a new security protocol model, used to describe protocols and their behaviour. The model provides several novel features.

Chapter 3: Security Properties

The model of Chapter 2 is extended with definitions of several security properties in Chapter 3, including notions of secrecy and definitions of existing authentication properties. We expand the existing set of authentication properties by developing a very strong notion of authentication which is called *injective synchronisation*. We give a hierarchy of authentication properties, and give a syntactic criterion for establishing injectivity of synchronisation. We apply the formal definitions to prove properties of the Needham-Schroeder-Lowe protocol manually. The chapter is based on [65, 70, 69, 67].

The contribution in this chapter consists of definitions of existing security properties, as well as the definition of a new strong authentication property called synchronisation. The new authentication properties are related to the existing ones by means of a hierarchy. For the synchronisation property, a result is proven with respect to the injectivity of this property.

Chapter 4: Verification

Based on the security protocol model, an algorithm is presented in Chapter 4 which can be used to verify security properties or find attacks, but also has the feature of being able to give a *complete characterization* of a protocol. This algorithm has been implemented in a prototype tool called *Scyther*. The performance of this tool is state-of-the art for security protocol analysis, and we apply the tool to a large number of protocols.

The contributions in this chapter consist of an improved verification algorithm based

on an existing algorithm. The concepts behind the algorithm are reformulated within the security protocol model presented here. This results in an improved algorithm. As a side effect, the new algorithm shares features with the semantics, such as being able to verify the new authentication properties. We establish a connection between the output of the new algorithm and an existing notion of characterization. Next, the algorithm is refined further with a mechanism to ensure termination. We show that for the vast majority of cases, this does not decrease its effectiveness. The algorithm involves heuristics, for which we investigate several alternatives and investigate their effectiveness. Termination of the new algorithm is guaranteed by introduction of an additional parameter. We investigate the consequences of particular choices for this parameter.

Chapter 5: Multi-Protocol Attacks

The prototype tool is used in Chapter 5 for the automated analysis of the parallel execution of multiple protocols. This situation can occur e. g. in embedded systems, like smart-card protocols or cellular phone applications. This results in the discovery of several new attacks and results about safe practices. This chapter is based on [62].

For this chapter, the theoretical contribution consist of a definition of so-called multi-protocol attacks. Practical contributions consist of the analysis of multi-protocol attacks on a large set of protocols, and the subsequent discovery of several new multi-protocol attacks. Two patterns are identified which capture possible problems with protocols that are executed over the same network.

Chapter 6: Generalizing NSL for Multi-Party Authentication

A further application of the model and tool is examined in Chapter 6, where the Needham-Schroeder-Lowe protocol is generalized to a family of multi-party authentication protocols. A proof of correctness is sketched of secrecy and synchronisation of the generalized version of Needham-Schroeder-Lowe is given. The developed protocols can serve as an efficient basis for multi-party synchronising protocols. This chapter is based on [66].

The contributions in this chapter include a link between the notion of multi-party synchronisation and the minimal number of messages in a protocol. A family of multi-party authentication protocols is developed. A parameterized proof is sketched which proves the correctness of the generalized version of the Needham-Schroeder-Lowe protocol.

Finally, we discuss related work in Chapter 7 and close off with conclusions and future work in Chapter 8.

2

Operational Semantics

In this chapter we develop a formal semantics of security protocols, based on a concise domain analysis. The main virtue of the semantics is that it separates concerns as much as possible, clearly distinguishing protocol descriptions from their dynamic behaviour, and the intruder model. Further characteristics of the model are a straightforward handling of parallel execution of multiple protocols, locality of security claims, the binding of local constants to role instances, and an explicitly defined initial intruder knowledge.

The model is designed to serve as an intuitive core model, and to allow for the analysis of the basic security protocol concepts. As a consequence, elements that occur only in some security protocols (e. g. time or flow-control) are not included in the model, with the intent of adding them in future work. Here, our primary concern is to identify the most important elements that play a role in security protocol analysis, and make their relations precise.

We will first indicate the concepts that are involved in security protocols in Section 2.1. In Section 2.2, we define the security protocol level at which the roles of a protocol are specified. The roles only define behaviour schemes, which are instantiated into runs in Section 2.3. This section also contains the agent model by describing the operational rules which define the behaviour of a network of agents without an intruder. In Section 2.4 we add an intruder to the model.

2.1 Domain analysis

We first conduct an analysis of the main concepts of security protocols. The purpose of this analysis is to make the design decisions explicit and to decompose the problem into smaller parts.

We start with an informal description of involved concepts. A security protocol distinguishes a number of behaviours. Each such behaviour we will call a role. We have, for instance, the *initiator* role and the *responder* role in a protocol. A system consists of a number of communicating agents. Each agent performs one or more roles (possibly from several security protocols). Thus, the system does not execute "the protocol". Rather, the system executes protocol roles, performed by agents. A role performed by an agent is called a run. For instance, agent

Alice can perform two initiator runs and one responder run in parallel. The agents execute their runs to achieve some security goal (e.g. the confidential exchange of a message). While agents try to achieve their goals, an intruder may try to oppose them. The *capabilities* of the intruder determine its strength in attacking a protocol run. However, threats do not only come from the outside. Agents taking part in a protocol run may also conspire with the intruder and try to invalidate the security goals. In order to resist attacks, an agent can make use of *cryptographic primitives* when constructing messages.

Given this global description, we can identify the following components of the security protocol model.

Protocol specification
Agent model
Communication model
Threat model
Cryptographic primitives
Security requirements

We will discuss each of these aspects, list their points of variation and make appropriate design decisions. Of course, every subdivision of the problem is artificial, but we found that here it helps in adding structure and restricting the problem space. The sub-models mentioned are not independent entities. For instance, the protocol specification makes use of the provided cryptographic primitives and the communication model is connected to the intruder model if the intruder has control over (some part of) the network.

Protocol specification. The protocol specification describes the behaviour of each of the roles in the protocol. We consider the protocol specification as a parameter of our semantics. We define an (abstract) syntax to specify a security protocol. A complete coverage of the wide range of security protocols in full detail goes beyond the scope of this thesis. Instead, we opt to describe the protocols on a suitable abstraction level. This will restrict the applicability of our results, in the sense that some protocols will not fit into our model. On the positive side, this choice allows us to develop high-level reasoning methods and tools, which can be applied to a large class of protocols. One of the choices we make is to specify a role in a security protocol as a sequential list of events. In practice, a security enhanced communication protocol requires a more expressive specification language, but for an abstract description of e.g. an authentication protocol a sequential list will suffice.[1] Furthermore, we will consider security claims as special events. Timers (and all other time related information) are not included in our model. A protocol specification is not complete without a specification of the initial knowledge required to execute a role and the declaration of functions, global constants and variables occurring in the protocol specification. An important part of security protocols is the generation of fresh values which are used e. g. for challenge-response mechanisms (often called *nonces*),

[1] Thus, we exclude for example branching and looping from the basic model presented in this thesis, and consider these to be possible refinements.

or as session keys. The protocol specification is expressed in a formal language for which we will define an abstract syntax and static requirements.

Agent model. Agents execute the roles of the protocol. The agent model is based on a *closed world assumption*. By this we mean that honest agents show no behaviour other than the behaviour described in the protocol specification. Thus, unless specified explicitly in the protocol, an honest agent will never leak classified information. The closed world assumption does not imply that an agent will only execute one run of the protocol. We assume that an agent may execute any finite number of runs in parallel (in an interleaved manner). The agent model also describes how an agent interprets a role description. An agent executes its role description sequentially, waiting at read events until an expected input message becomes available. This implies that an agent ignores unanticipated messages. More specifically, an incoming message will be matched against the expected message format as described by the protocol specification. Our semantics will be parameterized over this matching function, e.g. to allow for detection of type-flaw attacks.

Communication model. The communication model describes how the messages between the agents are exchanged. We have chosen for the model of asynchronous communication. In order to express various different threat models, as discussed in the next paragraph, we define the communication model using two multiset buffers. These buffers are shared by all agents.

Threat model. In 1983 Dolev and Yao laid the basis for a network threat model that is currently the one most widely used [79]. In the Dolev-Yao model the intruder has complete control over the communication network. The intruder can intercept any message and insert any message, as long as he is able to construct its contents from his knowledge. The commonly used variants of the Dolev-Yao model also implicitly include conspiring agents. Here we make the notion of conspiring agents explicit, which allows for a systematic derivation of the initial knowledge of the intruder. Intruder models that are weaker than the Dolev-Yao model are also of interest, for instance when studying protocol stacks or special communication media. Wireless communication, for instance, implies that an intruder has the choice of jamming or eavesdropping, but not both for the same message. Therefore, we will consider the intruder model as a parameter of our semantics.

Cryptographic primitives. Cryptographic primitives are (idealized) mathematical constructs such as encryption. In our treatment of cryptographic primitives we adopt the so-called *black box approach*. This means that we do not exactly know which mathematical objects are used to implement such constructs, but that we only know their relevant properties. On top of this, we will only consider symmetric and asymmetric encryption. In line with the black box approach, we assume cryptography to be perfect, captured by the *perfect cryptography assumption*. This

assumption states that nothing can be learned of a plain text from its encrypted version, without knowing the decryption key.

Security requirements. Security requirements state the purpose of a security protocol. They are often expressed as safety properties (i.e. something bad will never happen). In this thesis we will only study secrecy and various forms of authentication. However, the semantics is set up in such a way that other trace-based security properties are easily expressible. In this chapter we will only describe how generic trace-based security properties are dealt with in this model. The details of specific security properties will be addressed in Chapter 3.

2.2 Security protocol specification

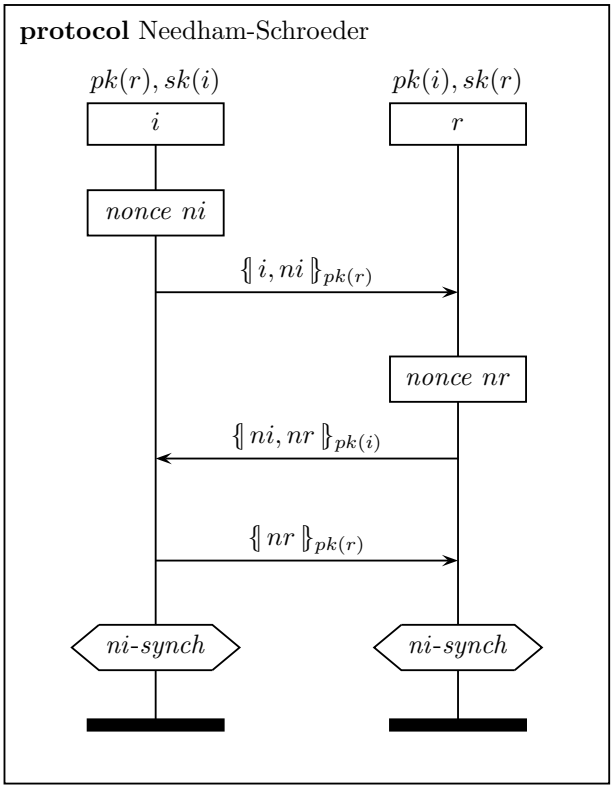

Figure 2.1: The Needham-Schroeder public key authentication protocol.

As a running example in this chapter, we use the short version of the Needham-Schroeder protocol from [145], referred to as *NS*. Throughout this thesis we use Message Sequence Charts (MSC) to illustrate security protocols and attacks. MSC is an ITU-standardized protocol specification language [105]. Figure 2.1 contains

an MSC of the *NS* protocol. The initiator i holds her own secret key $sk(i)$ and the public key $pk(r)$ of the responder r. Symmetrically, the responder r possesses his own secret key $sk(r)$ and the public key $pk(i)$ of the initiator i. We denote encryption of a message m with a key k as $\{\!|\, m \,|\!\}_k$. The initiator first creates a new nonce ni, denoted by the box, and then sends her name i together with the nonce ni, encrypted with the public key $pk(r)$, to the responder. After receipt of this, the responder generates a new nonce nr and sends it, together with the earlier nonce ni, covered by the public key $pk(i)$ to the initiator. She, in turn, unpacks the message and returns the nonce nr of the responder, encrypted with his public key. Security claims are denoted by hexagons. Both the initiator and the responder claim that the authentication property *ni-synch* holds. The details of such security claims will be explained in Chapter 3.

A protocol specification defines the exchange of message terms between agents. Note that the terms that occur in the protocol specification are different from those that we will define later for the execution model. Here we define role terms, which are used in the specification.

2.2.1 Role terms

We start by explaining a number of basic elements of these terms, such as constants, roles and variables. Next, we add constructors for pairing and tupling to construct the set *RoleTerm* that will be used in role descriptions.

Definition 2.1 (Basic term sets). *We define the basic sets used to construct Role terms:*

- *Var, denoting variables that are used to store received messages.*
- *Const, denoting constants which are freshly generated for each instantiation of a role, and are therefore considered local constants.*
- *Role, denoting roles*
- *Func, denoting function names*

In Table 2.1 on the next page we show some typical elements of these sets, as used throughout this thesis.

Definition 2.2 (Role Terms). *We define the set of Role Terms as the basic term sets, extended with constructors for pairing and encryption, and we assume that pairing is right-associative.*

$$\begin{aligned} \mathit{RoleTerm} ::= {} & \mathit{Var} \mid \mathit{Const} \mid \mathit{Role} \mid \mathit{Func}(\mathit{RoleTerm}^*) \\ & \mid (\mathit{RoleTerm}, \mathit{RoleTerm}) \\ & \mid \{\!|\, \mathit{RoleTerm} \,|\!\}_{\mathit{RoleTerm}} \end{aligned}$$

Description	Set	Typical elements
Role terms	*RoleTerm*	$rt1, rt2$
Variables	*Var*	V, W, X, Y, Z
Constants	*Const*	$ni, nr, sessionkey$
Roles	*Role*	i, r, s
Functions	*Func*	sk, pk, k, hash

Table 2.1: Basic sets and some typical elements

Functions from the set *Func* are considered to be global, and have an arity which must be respected in all terms. Note that there is no term f to express "the function f". When we want to refer to the function f (with domain X), we will use f as a shorthand for $\{f(x) \mid x \in X\}$.

If global constants occur in a protocol, we model them as functions of arity zero.

Terms that have been encrypted with a term, can only be decrypted by either the same term (for symmetric encryption) or the inverse key (for asymmetric encryption). To determine which term needs to be known to decrypt a term, we define a function that yields the inverse for any role term.

$$\_^{-1} : \mathit{RoleTerm} \to \mathit{RoleTerm}$$

We require that $\_^{-1}$ is its own inverse, i.e. $(rt^{-1})^{-1} = rt$.

Throughout this thesis we will assume that pk and sk are functions of arity 1 that map to asymmetric keys, such that $\forall R \in \mathit{Role} : pk(R)^{-1} = sk(R) \ \wedge \ sk(R)^{-1} = pk(R)$. In particular, we will use pk to denote the public keys of the agents, and sk to denote the private key of the agents.

Example 2.3 (Signing). *We model the secret key $sk(R)$ of a role R as the inverse of the public key $pk(R)$. Encryption of a message m with the public key is modeled as $\{\!|\, m \,|\!\}_{pk(R)}$. Signing a message does not directly correspond to encryption with the secret key. Rather, a so-called "message digest" is encrypted, and sent along with the message. In order to correctly model signing of a message m with a private key $sk(R)$, we need to introduce a hash function h. Signing m then results in the term $(m, \{\!|\, h(m) \,|\!\}_{sk(R)})$.*

All other terms rt are considered to be symmetric keys, for which we have $rt^{-1} = rt$ unless stated otherwise. Thus, although we use the notation $\{\!|\ |\!\}$ for both types of encryption, the type of encryption that is applied can be derived from the key that is used. Note that we explicitly allow for composed keys, e. g. as in $\{\!|\, rt1 \,|\!\}_{(rt2,rt3)}$.

We now turn to describing the protocol behaviour. We describe protocols as sets of roles, which in turn consist of role events.

Definition 2.4 (Role events). *We define the set of events RoleEvent using two*

new sets, labels Label and security claims Claim, which we explain below.

$$\begin{aligned} \mathit{RoleEvent} = \{ & \mathit{send}_\ell(R, R', rt), \\ & \mathit{read}_\ell(R', R, rt), \\ & \mathit{claim}_\ell(R, c, rt) \\ & \mid \ell \in \mathit{Label}, R, R' \in \mathit{Role}, rt \in \mathit{RoleTerm}, c \in \mathit{Claim} \} \end{aligned}$$

Event $\mathit{send}_\ell(R, R', rt)$ *denotes the sending of message* rt *by the agent bound to* R*, intended for the agent bound to* R'*. Likewise,* $\mathit{read}_\ell(R', R, rt)$ *denotes the reception of message* rt *by the agent bound to* R'*, apparently sent by the agent bound to* R*. Event* $\mathit{claim}_\ell(R, c, rt)$ *expresses that* R *upon execution of this event expects security goal* c *to hold with optional parameter* rt*. A claim event denotes a* local claim*, which means that it only concerns role* R *and does not express any expectations at other roles.*

The concept of local security claims will be discussed in detail in Chapter 3.

The set *RoleEvent* contains, at the level of the protocol description, all the actions to be performed in the protocol. We distinguish protocol events for sending, reading and claiming. Although one may include other protocol events as well, e.g. to model internal activity within a role, we do not do so here; the distinction of send protocol events, read protocol events, and claim events, suffices for most of our purposes. We leave generation of local constants (such as nonces) implicit, and assume that the generation of a local constant takes place just before its first occurrence in an event.

The labels ℓ that tag the events are needed to disambiguate similar occurrences of the same event in a protocol specification. A second use of these labels will be to express the relation between corresponding send and read events, as we will see in the next chapter.

For terms, we introduce an operator that allows us to identify parts of the term.

Definition 2.5 (Subterm operator $\sqsubseteq$). *The subterm operator* $\sqsubseteq$ *is inductively defined as follows.*

$$\begin{aligned} rt &\sqsubseteq rt \\ rt1 &\sqsubseteq (rt1, rt2) \\ rt2 &\sqsubseteq (rt1, rt2) \\ rt1 &\sqsubseteq \{\!|\, rt1 \,|\!\}_{rt2} \\ rt2 &\sqsubseteq \{\!|\, rt1 \,|\!\}_{rt2} \end{aligned}$$

The subterm operator $\sqsubseteq$ identifies subterms of a term, which also includes keys used in the encryption of terms. Note that terms of the form $f(rt)$ have no proper subterms, as they are considered to be non-composed terms.

Besides terms to be sent and received, a role specification describes the initial knowledge needed to execute the role. This initial knowledge is a set of terms.

Definition 2.6 (Role Knowledge). *We define the role knowledge set RoleKnow as the set of all sets of role terms that do not contain variables as a subterm.*

$$RoleKnow = \mathcal{P}(\{rt \in RoleTerm \mid \forall rt' : rt' \sqsubseteq rt \Rightarrow rt' \notin Var\})$$

As a consequence, we have that $RoleKnow \subset \mathcal{P}(RoleTerm)$.

Because the role events fully describe the behaviour of a role, and there is no need to e.g. "declare" local constants and variables, the role knowledge is not needed for the execution of the protocol. The role knowledge is introduced for two reasons. First, it allows for a static role consistency test, to see whether the role can be executed given a starting knowledge set. Second, it will enable us to systematically derive the initial intruder knowledge, at the end of this chapter.

Definition 2.7 (Role specification). *A role specification consists of a list of events, and some initial knowledge. Thus, the set of all role specifications is defined as* $RoleSpec = RoleKnow \times RoleEvent^*$.

We write $[\varepsilon 1, \varepsilon 2]$ to denote the list of two events $\varepsilon 1$ and $\varepsilon 2$. We write $[\varepsilon 1, \varepsilon 2] \cdot [\varepsilon 3]$ to denote the concatenation of two lists. Often we will omit the square brackets for single events and write $\varepsilon 1 \cdot \varepsilon 2$ to denote $[\varepsilon 1, \varepsilon 2]$.

We write $A \nrightarrow B$ to denote a partial function, which maps some elements of A to elements of B.

Definition 2.8 (Protocol specification). *A protocol specifies the behaviour for a number of roles by means of a partial function from the set Protocol. Thus we have that* $Protocol = Role \nrightarrow RoleSpec$.

We will use $MR^P(R)$ as a shorthand for the initial knowledge of role R in a protocol specification P. In many cases we omit the parameter P if the intended protocol is clear from the context. The notation MR is a contraction of M, which we will later use to denote knowledge, and R, used to denote roles.

Role events are required to be unique within a protocol description, which can be enforced by the labeling scheme. This allows us to define a function $role : RoleEvent \rightarrow Role$. Given a role event, the function *role* yields the role the event belongs to. We leave the protocol as an implicit parameter when the intended protocol is clear from the context.

Example 2.9 (Role description). *The following role description models the initiator role of the Needham-Schroeder protocol, without any security requirements.*

$$\begin{aligned} ns(i) = \big(& \{i, r, ni, sk(i), pk(i), pk(r)\}, \\ & send_1(i, r, \{\!|\, i, ni \,|\!\}_{pk(r)}) \cdot \\ & read_2(r, i, \{\!|\, ni, V \,|\!\}_{pk(i)}) \cdot \\ & send_3(i, r, \{\!|\, V \,|\!\}_{pk(r)}) \cdot \\ & claim_4(i, ni\text{-}synch)\big) \end{aligned}$$

This role description follows from Figure 2.1 on page 12 by selecting the left-most axis and its associated events. Notice that we have to clarify which constructs in the terms are variables (because they receive their value at reception of a message) and which are constants (because they are determined by the role itself). Therefore, we consider $i, r \in \mathit{Role}$, $ni \in \mathit{Const}$, $sk, pk \in \mathit{Func}$, $pk(i)^{-1} = sk(i)$, $pk(r)^{-1} = sk(r)$, $1, 2, 3, 4 \in \mathit{Label}$, *and* $V \in \mathit{Var}$.

2.2.2 Event order

Each role of the protocol corresponds to list of events. In other words, a (sequential) structure is imposed on the set of protocol events belonging to a role R. This total ordering is denoted by $\prec_R$. Thus, for any role $R \in \mathit{Role}$ and $\varepsilon 1, \varepsilon 2 \in \mathit{RoleEvent}$ such that $\mathit{role}(\varepsilon 1) = R$ and $\mathit{role}(\varepsilon 2) = R$ we have that

$$\varepsilon 1 \prec_R \varepsilon 2 \ \vee \ \varepsilon 1 = \varepsilon 2 \ \vee \ \varepsilon 1 \succ_R \varepsilon 2$$

We consider an abstract security protocol P as a collection of communicating sequential processes. Each of the sequential components is carried by a specific role. The communication is governed by the labels that decorate the events, as the labels directly define the communication relation $\rightsquigarrow$.

Definition 2.10 (Communication relation). *For all* $\ell \in \mathit{Label}$ *and* $m1, m2 \in \mathit{Role} \times \mathit{Role} \times \mathit{RoleTerm}$, *the communication relation* $\rightsquigarrow$ *is defined as:*

$$\varepsilon 1 \rightsquigarrow \varepsilon 2 \iff \exists \ell, m1, m2 : \varepsilon 1 = \mathit{send}_\ell(m1) \wedge \varepsilon 2 = \mathit{read}_\ell(m2)$$

This relation prescribes how send protocol events and read protocol events correspond.

Example 2.11 (Event order and communication relation). *For the example protocol NS, the role orderings* $\prec_i$ *and* $\prec_r$ *on the roles* i *and* r, *respectively, are as follows:*

$$\begin{aligned}
&\mathit{send}_1(i, r, \{\!|\, i, ni \,|\!\}_{pk(r)}) \prec_i \mathit{read}_2(r, i, \{\!|\, ni, V \,|\!\}_{pk(i)}) \\
&\qquad\qquad \prec_i \mathit{send}_3(i, r, \{\!|\, V \,|\!\}_{pk(r)}) \prec_i \mathit{claim}_4(i, \mathit{ni\text{-}synch}) \\
&\mathit{read}_1(i, r, \{\!|\, i, W \,|\!\}_{pk(r)}) \prec_r \mathit{send}_2(r, i, \{\!|\, W, nr \,|\!\}_{pk(i)}) \\
&\qquad\qquad \prec_r \mathit{read}_3(i, r, \{\!|\, nr \,|\!\}_{pk(r)}) \prec_r \mathit{claim}_5(r, \mathit{ni\text{-}synch})
\end{aligned}$$

The communication relation $\rightsquigarrow$ *is given as:*

$$\mathit{send}_1(i, r, \{\!|\, i, ni \,|\!\}_{pk(r)}) \rightsquigarrow \mathit{read}_1(i, r, \{\!|\, i, W \,|\!\}_{pk(r)}) \tag{2.1}$$

$$\mathit{send}_2(r, i, \{\!|\, W, nr \,|\!\}_{pk(i)}) \rightsquigarrow \mathit{read}_2(r, i, \{\!|\, ni, V \,|\!\}_{pk(i)}) \tag{2.2}$$

$$\mathit{send}_3(i, r, \{\!|\, V \,|\!\}_{pk(r)}) \rightsquigarrow \mathit{read}_3(i, r, \{\!|\, nr \,|\!\}_{pk(r)}) \tag{2.3}$$

2.2.3 Static Requirements

In the previous section we have explained the abstract syntax for a protocol specification. A proper protocol specification will also have to satisfy a number of well-formedness requirements. Not any sequence of send, read and claim events is considered a security protocol. For example, we require that an agent must be able to construct the terms he sends out, and that he cannot examine the content of encrypted terms of which he does not know the key.

Well-Formed Roles. For each role, we require that it meets certain criteria. These range from the fairly obvious, e.g. each event in a role definition has the same role that executes it (referred to as the *actor* of the event), to more subtle requirements regarding the messages. For the messages we require that the messages that are sent can actually be constructed by the sender. This is satisfied if the message is in the knowledge of the sending role. For variables we require that they first occur in a read event, where they are instantiated, before they can occur in a send event.

For read events the situation is a bit more complex. As can be seen in the example above, which describes the initiator role of the Needham-Schroeder protocol, a read event imposes structure upon the incoming messages, in the form of a pattern. A receiver can only match a message against such an expected pattern if his knowledge satisfies certain requirements.

We introduce a predicate *WF* (Well-Formed) to express that a role definition meets these consistency requirements, using an auxiliary predicate *RD* (Readable) and a knowledge inference relation $\vdash$: $\mathit{RoleKnow} \times \mathit{RoleTerm}$.

Roles can compose and decompose pair terms. A term can be encrypted if the agent knows the encryption key, and an encrypted term can be decrypted if the agent knows the corresponding decryption key.

Definition 2.12 (Knowledge inference operator). *Let M be a role knowledge set. The knowledge inference relation $\vdash$: $\mathit{RoleKnow} \times \mathit{RoleTerm}$ is defined inductively as follows, for all role terms $rt, rt1, rt2, k$:*

$$\begin{aligned} rt \in M &\Longrightarrow M \vdash rt \\ M \vdash rt1 \wedge M \vdash rt2 &\Longrightarrow M \vdash (rt1, rt2) \\ M \vdash rt \wedge M \vdash k &\Longrightarrow M \vdash \{\!|\, rt \,|\!\}_k \\ M \vdash (rt1, rt2) &\Longrightarrow M \vdash rt1 \wedge M \vdash rt2 \\ M \vdash \{\!|\, rt \,|\!\}_k \wedge M \vdash k^{-1} &\Longrightarrow M \vdash rt \end{aligned}$$

Example 2.13 (Inference and encryptions). *From a term set that consists of the term $\{\!|\, m \,|\!\}_k$, but not k^{-1}, it is impossible to infer m or k, i. e.*

$$\left\{ \{\!|\, m \,|\!\}_k \right\} \nvdash m \ \wedge \ \left\{ \{\!|\, m \,|\!\}_k \right\} \nvdash k$$

Given $\{\!|\, m \,|\!\}_k$ and k^{-1}, we can infer m. If k is an asymmetric key (and thus $k \neq k^{-1}$) and given $\{\!|\, m \,|\!\}_k$ and k^{-1}, it is not possible to infer k.

$$k \neq k^{-1} \Rightarrow \left\{ \{\!|\, m \,|\!\}_k, k^{-1} \right\} \nvdash k$$

Example 2.14 (Inference and subterms). *Given two terms $m1$ and $m2$ with $\{m1\} \vdash m2$, it is in general not the case that $m2 \sqsubseteq m1$. A counterexample is given by $m1 = (rt1, rt2)$ and $m2 = (rt2, rt1)$, where $rt1 \neq rt2$. The converse does not hold either, e.g. because given terms k and m and $\{m\} \nvdash k$, we have that $k \sqsubseteq \{\!|\, m \,|\!\}_k$ but also that $\{\!|\, m \,|\!\}_k \nvdash k$.*

Example 2.15 (Inference and functions). *Given a term $f(rt)$ where $f \in Func$, there is no further inference rule that applies. Thus, it is impossible to infer rt. In this respect functions in our framework act as hash functions. We can safely model keys (e.g. $k(rt1, rt2), sk(rt1)$) as hash functions in the same way, because as we will see later, the agent names are known initially to the agents as well as the intruder. Modeling $\{k(rt1, rt2)\} \vdash rt1, rt2$ therefore does not increase the set of inferable terms for any of the parties involved. (Within the model, all sets to which the inferences operator is applied will allready include $rt1$ and $rt2$.)*

The predicate $RD : \mathcal{P}(RoleTerm) \times RoleTerm$ expresses which role terms can be used as a message pattern for a read event of an agent with a specific knowledge set. A variable can always occur in a read pattern. Any other term can only occur in a read pattern, if it can be inferred from the knowledge of the agent (consisting of the initial knowledge and knowledge gained from any preceding read events). Only then can it be compared to the incoming messages.

In order to be able to read a pair, we must be able to read each constituent, while extending the knowledge with what can be inferred from the other component. An encrypted message can be read if it can be inferred from the knowledge or if it can be inferred after decryption, which requires that the decryption key is in the knowledge.

Definition 2.16 (Readable predicate). *Let M be a role knowledge set and rt a role term, used as a pattern.*

$$RD(M, rt) = \begin{cases} True & \text{if } rt \in Var \\ RD(M \cup \{rt2\}, rt1) \wedge RD(M \cup \{rt1\}, rt2) & \text{if } rt \equiv (rt1, rt2) \\ (M \vdash \{\!|\, rt1 \,|\!\}_{rt2}) \vee (M \vdash rt2^{-1} \wedge RD(M, rt1)) & \text{if } rt \equiv \{\!|\, rt1 \,|\!\}_{rt2} \\ M \vdash rt & \text{otherwise} \end{cases}$$

In particular, if we look at the case in which $rt \equiv (rt1, rt2)$, being able to read such a message can correspond to first extracting some key from $rt2$ which can be used to decrypt $rt1$, or vice versa.

We can now construct the predicate $WF : Role \times RoleSpec$ that expresses that a role is well formed. The first argument of this predicate is always the role in which the event occurs. It is used to express that the role that executes an event (i. e. the sender of a send event) should match the role in which it occurs. Terms occurring in a send or claim event must be inferable from the knowledge, while terms occurring in a read event must be readable according to the definition above. Furthermore, an agent should only read messages directed to him, and should only send out messages

that are labeled with the correct sender. If a term is used as a parameter of a claim event, there is no such requirement, because whether or not this is appropriate depends on the function of the parameter within the claim, as we will see in the next chapter.

Definition 2.17 (Well-formedness). *Let R be a role, and let M be a role knowledge set. Let s be a list of role events.*

$$WF(R,(M,s)) = \begin{cases} True & \text{if } s \equiv \varepsilon \\ M \vdash (R',R) \wedge RD(M,rt) \wedge WF(R,(M \cup \{rt\},s')) & \text{if } s \equiv read_\ell(R',R,rt)\cdot s' \\ M \vdash (R,R',rt) \wedge WF(R,(M,s')) & \text{if } s \equiv send_\ell(R,R',rt)\cdot s' \\ M \vdash R \wedge WF(R,(M,s')) & \text{if } s \equiv claim_\ell(R,c,rt)\cdot s' \\ False & \text{otherwise} \end{cases}$$

For a protocol specification $P : Role \nrightarrow RoleSpec$ we require that all roles are well-formed with respect to their initial knowledge, which is expressed by $\forall R \in dom(P) : WF(R, P(R))$, where we use the notation $dom(f)$ to express the domain of a partial function f.

Example 2.18 (Correctness of role descriptions). *Consider the following incorrect role description:*

$$\begin{aligned} wrong1(i) = (\ &\{i,r,k\}, \\ &send_1(i,r,\{\!|\, i,r,V \,|\!\}_k)\cdot \\ &read_2(r,i,\{\!|\, V,r \,|\!\}_k)\ ) \end{aligned}$$

Role description wrong1 is not well-formed because it sends variable V before it is read. Based on Definition 2.6, the initial role knowledge does not contain variables. In the WF predicate, variables will only become part of the role knowledge (parameter M) after they have been read. In subsequent events, these variables can be inferred from m and can thus be part of sent messages as well.

Example 2.19 (Correctness of role descriptions). *Consider the following incorrect role description:*

$$\begin{aligned} wrong2(i) = (\ &\{i,r,k\}, \\ &read_1(r,i,\{\!|\, i,r,\{\!|\, V \,|\!\}_{k2} \,|\!\}_k)\cdot \\ &send_2(i,r,\{\!|\, V \,|\!\}_{k2})\ ) \end{aligned}$$

The read event in wrong2 contains a subterm $\{\!|\, V \,|\!\}_{k2}$. The intention is that V is initialised through this read. However, since $k2$ is a symmetric key, and $k2$ is not in the knowledge of the role, the value of V cannot be determined through this read. Therefore, this role description is not well-formed. A correct role description is the following:

$$\begin{aligned} wrong2corrected(i) = (\ &\{i,r,k\}, \\ &read_1(r,i,\{\!|\, i,r,W \,|\!\}_k)\cdot \\ &send_2(i,r,W)\ ) \end{aligned}$$

2.3 Describing protocol execution

In the previous section we have formalized the notion of a protocol description, which is a static description of how a protocol should behave. When such a protocol description is executed, dynamic aspects are introduced. This involves aspects from the agent model as well as the execution model from the domain analysis. This requires the introduction of some new concepts that did not occur on the protocol description level. In particular, we will model the dynamic behaviour of the model as a labeled transition system.

2.3.1 Runs

A protocol specification describes a set of roles. These roles serve as a blueprint for what the actual agents in a system can do. When a protocol is executed, each role can be executed a number of times, possibly in parallel and by multiple agents. We call such a single, possibly partial, execution of a role description a *run*. From an implementation point of view, a run is similar to a *thread* of an agent. In our model, two runs executed by the same agent are independent and share no variables.

Executing a role turns a role description into a run. This is referred to as instantiation. In order to instantiate a role we have to bind the role names to the names of actual agents and we have to make the local constants unique for each instantiation. Furthermore, we have to take into account that the bindings of values to the variables are local to a run too. Thus, the set of terms occurring in a run differs from the set of terms used in role descriptions.

We assume existence of a set *RID* to denote run identifiers and a set *Agent* to denote agents. Run terms are defined similarly to role terms. The difference is that abstract roles are replaced by concrete agents that local constants are made unique by extending them with a run identifier, and that variables are instantiated by concrete values. The run term set also includes the set *IntruderConst* of basic run terms generated by an intruder. This set will only be used from Section 2.4.2 onwards, where it will be defined and explained in more detail.

Definition 2.20 (Run terms).

$$\begin{aligned} RunTerm ::= \; & Const \sharp RID \\ | \; & Agent \\ | \; & Func(RunTerm^*) \\ | \; & (RunTerm, RunTerm) \\ | \; & \{\!|\, RunTerm \,|\!\}_{RunTerm} \\ | \; & IntruderConst \end{aligned}$$

Definition 2.21 (Instantiation function). *For each run, there is a relation between the role terms and the run terms. A role term is transformed into a run term*

Description	Set	Typical elements
Run terms	*RunTerm*	$t1, t2$
Instantiated constants		$ni\sharp 1, nr\sharp 2, sessionkey\sharp 1$
Agents	*Agent*	A, B, C, S, E

Table 2.2: Basic run sets and some typical elements

by applying an instantiation.

$$Inst = RID \times (Role \nrightarrow Agent) \times (Var \nrightarrow RunTerm)$$

The first component of an instantiation determines with which run identifier the constants are extended. The second component determines the instantiation of roles by agents. The third determines the valuation of the variables.

We extend the inverse function $^{-1}$ to *RunTerm*. The functions $roles : RoleTerm \to \mathcal{P}(Role)$ and $vars : RoleTerm \to \mathcal{P}(Var)$ determine the roles and variables occurring in a term. We extend these functions to the domain of *RoleSpec* in the obvious way.

Often we will extract the first component, the run identifier, from an instantiation. We will use the notation $runidof(inst)$ to denote the run identifier from an instantiation *inst*.

Definition 2.22 (Term instantiation). *Let $inst \in Inst$ be an instantiation, where $inst = (\theta, \rho, \sigma) \in Inst$. Let $f \in Func$ and let $rt, rt1, \ldots, rtn$ be role terms such that $roles(rt) \subseteq dom(\rho)$ and $vars(rt) \subseteq dom(\sigma)$. We define instantiation, $\langle inst \rangle : RoleTerm \to RunTerm$, by:*

$$\langle inst \rangle(rt) = \begin{cases} \rho(R) & \text{if } rt \equiv R \in Role \\ f\big(\langle inst \rangle(rt1), \ldots, \langle inst \rangle(rtn)\big) & \text{if } rt \equiv f(rt1, \ldots, rtn) \\ c\sharp\theta & \text{if } rt \equiv c \in Const \\ \sigma(v) & \text{if } rt \equiv v \in Var \\ (\langle inst \rangle(rt1), \langle inst \rangle(rt2)) & \text{if } rt \equiv (rt1, rt2) \\ \{\!|\, \langle inst \rangle(rt1) \,|\!\}_{\langle inst \rangle(rt2)} & \text{if } rt \equiv \{\!|\, rt1 \,|\!\}_{rt2} \end{cases}$$

Example 2.23 (Term instantiation). *We give two examples of instantiations that may occur in the execution of a protocol:*

$$\langle 1, \{i \mapsto A, r \mapsto B\}, \emptyset \rangle(\{\!|\, i, ni \,|\!\}_{pk(r)}) = \{\!|\, A, ni\sharp 1 \,|\!\}_{pk(B)}$$
$$\langle 2, \{i \mapsto C, r \mapsto D\}, \{W \mapsto ni\sharp 1\} \rangle(\{\!|\, W, nr, r \,|\!\}_{pk(i)}) = \{\!|\, ni\sharp 1, nr\sharp 2, D \,|\!\}_{pk(C)}$$

Definition 2.24 (Run). *A run is an instantiation of the events of a role, defined as $Run = Inst \times RoleEvent^*$.*

As we will see later on, each run in the system will have a unique run identifier by construction. Because the knowledge of a role is already statically defined by the role description, we can omit it from the run specification.

The system that we consider consists of a number of runs executed by some agents. Communication between the runs is asynchronous (buffered). In order to conveniently model different types of intruder behaviour, we will route communication through two buffers. We have one output buffer from the sending run and one input buffer from the receiving run (for a discussion on the expressive power of such a construction, see [84]). The intruder capabilities will determine how the messages are transferred from the output buffer to the input buffer.

Both the output buffer and the input buffer store sent messages. Messages contain a sender, a recipient, and a run term: $MSG = Agent \times Agent \times RunTerm$. A buffer is a multiset of such messages: $Buffer = \mathcal{M}(MSG)$. Abusing notation, we will use the set MSG as if $MSG \subset RunTerm$, to avoid redefining existing operators on $RunTerm$ for the set MSG.

The full network model also contains elements of the intruder model, which have not been discussed at this point. In particular, to define the notion of state in our transition system, we need to include a set of terms from $\mathcal{P}(RunTerm)$, which will be used in Section 2.4 to represent the dynamic intruder knowledge.

Definition 2.25 (Network state). *The state of a network of agents executing roles in a security protocol is defined by*

$$State = \mathcal{P}(RunTerm) \times Buffer \times Buffer \times \mathcal{P}(Run)$$

and thus a state contains the intruder knowledge, the contents of the output buffer, the contents of the input buffer, and the (remainders of the) runs that still have to be executed.

Messages from the buffer are accepted by agents if they match a certain pattern, specified in the read event. We introduce a predicate *Match* that expresses that a message matches the pattern for some instantiation of the variables. The definition of this predicate is a parameter of our system, but we will give an example of a straightforward match.

For example, we define for each variable a set of run terms which are allowed values. We introduce an auxiliary function $type : Var \to \mathcal{P}(RunTerm)$ that defines the set of run terms that are valid values for a variable. We give three possible examples for defining *type*, which each result in a slightly different semantics for the read events.

Example 2.26 (Type matching: no type flaws). *In most cases we assume that the set of run terms is partitioned into a finite number of sets $S_1, \ldots S_n$. For example, these can represent nonces, agent names, session keys, encryptions, tuples, etc. For the typed matching model we require that*

$$\forall V \in Var : \exists i : type(V) = S_i$$

Intuitively, this means that if a variable is of type "Nonce", it will only match with run terms that are nonces.

Example 2.27 (Simple type matching: basic type flaws). *A second option is a slightly relaxed instance of the type matching model, with only three partitions,*

where S_1 represents all encrypted terms, S_2 all tuples, and S_3 all other terms. In this case, the match predicate cannot distinguish between an agent name or a nonce, as both are elements of S_3. It is thus possible that a read event that expects a nonce term matches with a message that contains an agent name. When this occurs, we speak of a basic type flaw*.*

Example 2.28 (No type matching: all type flaws). *In the previous setting, there is still a distinction between the three partitions. Another possibility is no type matching, in which we have that*

$$\forall V \in Var : type(V) = RunTerm$$

Here, a variable can match with any term. It can therefore happen that a variable is instantiated with a tuple.

Throughout this thesis, we assume the type matching model, which distinguishes e.g. agent names from nonces as in Example 2.26, unless stated otherwise.

Definition 2.29 (Well-typedness of instantiation). *We define the predicate Welltyped on $(Var \to \mathcal{P}(RunTerm))$ that expresses that a substitution is well-typed:*

$$Welltyped(\sigma) = \forall v \in dom(\sigma) : \sigma(v) \in type(v)$$

Using this predicate, we define the typed matching predicate $Match : Inst \times RoleTerm \times RunTerm \times Inst$. The purpose of this predicate is to match an incoming message (the third argument) to a pattern specified by a role term (the second argument). This pattern is already instantiated (the first argument), but may still contain free variables. The idea is to assign values to the free variables such that the incoming message equals the instantiated role term. The old instantiation extended with these new assignments provides the resulting instantiation (the fourth argument).

Definition 2.30 (Match). *For all $inst, inst' \in Inst$, $pt \in RoleTerm$, $m \in RunTerm$, the match predicate is defined as:*

$$\begin{aligned} Match(inst, pt, m, inst') \iff\ & inst = (\theta, \rho, \sigma) \land inst' = (\theta, \rho, \sigma') \land \\ & \sigma \subseteq \sigma' \land dom(\sigma') = dom(\sigma) \cup vars(pt) \land \\ & Welltyped(\sigma') \land \langle \theta, \rho, \sigma' \rangle(pt) = m \end{aligned}$$

Example 2.31 (Match). *Assume $\rho = \{i \mapsto A, r \mapsto B\}$, and $Const \sharp RID \subseteq type(X)$. Then, some examples for which the predicate is true are:*

	$inst$	pt	m	$inst'$
$Match($	$(1, \rho, \emptyset)$,	X,	$nr\sharp 2$,	$(1, \rho, \{X \mapsto nr\sharp 2\})$)
$Match($	$(1, \rho, \emptyset)$,	$\{\!\|\, r, ni \,\|\!\}_{pk(i)}$,	$\{\!\|\, B, ni\sharp 1 \,\|\!\}_{pk(A)}$,	$(1, \rho, \emptyset)$)

In the first example above, the pattern is simply a single variable X, with a type that contains the set of local constants. From the first parameter we find that X has not

been instantiated yet. Thus, the agent expects to receive any value of this type. The message is $nr\sharp 2$*, which is an element of the type of* X*. Therefore, the message fits the pattern, and upon reading would assign to the variable* X *the read term* $nr\sharp 2$*.*

The second example above shows a more complicated pattern and a message that can be accepted. The pattern consists of a role name r *and a local constant* ni*, encrypted with the public key of a role* i*. From the instantiation with* ρ *we find that in this particular run, the* r *role is performed by* B*, and the* i *role by* A*. Furthermore, the local constant is unique to this run, of which the run identifier is* 1*. Thus, the agent reading this pattern gets exactly what he expects. There are no uninstantiated variables in the pattern. He expects exactly the term* $\{\!|\, B, ni\sharp 1 \,|\!\}_{pk(A)}$*, and any other message will not match this pattern.*

Some examples where the predicate does not *hold, can be produced if we assume type matching, and the type of* X *is the set* $Agent \cup Const\sharp RID \cup IntruderConst$*. In the last column* $inst'$ *we write* $\_$ *to denote that the predicate is false for any value of* $inst'$*.*

	$inst$	pt	m	$inst'$	
$\neg Match($	$(1, \rho, \emptyset)$,	nr,	$nr\sharp 2$,	$\_$	$)$
$\neg Match($	$(1, \rho, \emptyset)$,	X,	$(nr\sharp 2, ni\sharp 1)$,	$\_$	$)$
$\neg Match($	$(1, \rho, \emptyset)$,	$\{\!\| i, ni \|\!\}_{pk(i)}$,	$\{\!\| B, ni\sharp 1 \|\!\}_{pk(A)}$,	$\_$	$)$

In the first example above, the run with run identifier 1 expects to read his own local constant $nr\sharp 1$. In the second example, the message does not fit the type of the variable X. The third example the encryption key is the correct one, but the agent was expecting the name A inside, not B.

In order to define the behaviour of the system, a connection is made between the protocol descriptions (i. e. roles and their events) and their execution (i. e. when they are instantiated). In particular, when an instantiation function is combined with a role event, we call the result a run event.

Definition 2.32 (Run event). *A run event is a tuple* $(inst, \varepsilon)$*, where* $inst \in Inst$ *and* $\varepsilon \in RoleEvent$*. We use* $RunEvent$ *to denote the set of all run events.*

We write *ReadRunEv* for the set of run events corresponding to read events, *SendRunEv* for those corresponding to send events, and *ClaimRunEv* for claim events. Together, these three sets combined form the set *RunEvent*.

The behaviour of the system is defined as a transition relation between system states.

Definition 2.33 (Transition label). *A transition is labeled with an element of the set*

$$Transitionlabel ::= RunEvent \mid create(Run) \mid Networkrulename$$

The set of network/intruder rules *Networkrulename* is a parameter of the system, and we will discuss some of its possibilities in Section 2.4.1. When there is no

intruder, we simply take $\mathit{Networkrulename} ::= \mathit{transmit}(\mathit{RunTerm})$, referring to the transmit rule of Table 2.3 on the next page.

For transition labels corresponding to instances of read or send role events, we define a *content* extraction function. This function is defined as the run term that is being sent or read.

Definition 2.34 (Content of event). *We define a function* $\mathit{cont} : \mathit{ReadRunEv} \cup \mathit{SendRunEv} \to \mathit{RunTerm}$*, where we have*

$$\mathit{cont}((\mathit{inst}, \mathit{send}_\ell(R, R', m))) = \langle \mathit{inst} \rangle (R, R', m)$$
$$\mathit{cont}((\mathit{inst}, \mathit{read}_\ell(R, R', m))) = \langle \mathit{inst} \rangle (R, R', m)$$

A protocol description allows for the creation of runs. The runs that can be created are defined by the function *runsof*.

Definition 2.35 (Possible runs). *The runs that can be created by a protocol* P *are defined by the function* $\mathit{runsof} : \mathit{Protocol} \to \mathcal{P}(\mathit{Run})$ *:*

$$\mathit{runsof}(P) = \left\{ ((\theta, \rho, \emptyset), P(R)) \;\middle|\; \theta \in \mathit{RID} \wedge \rho \in \mathit{roles}(P(R)) \times \mathit{Agent} \wedge R \in \mathit{dom}(P) \right\}$$

Definition 2.36 (Active run identifiers). *Given a set of runs* F*, we define the set of active run identifiers as*

$$\mathit{runIDs}(F) = \left\{ \theta \;\middle|\; \big((\theta, \rho, \sigma), ev\big) \in F \right\}$$

For the operational semantics, we define substitution:

Definition 2.37 (Substitution). *For* $F \in \mathcal{P}(\mathit{Run})$ *we use* $F[x'/x]$ *to denote the substituting* x' *for* x *in all terms occurring in the runs in* F*.*

Let $P \in \mathit{Protocol}$. Then the basic derivation rules for the system are given in Table 2.3 on the facing page. The *create* rule expresses that a new run can only be created if its run identifier has not been used yet. The *send* rule states that if a run executes a send event, the sent message is added to the output buffer and the executing run proceeds to the next event. The *read* rule determines when an input event can be executed. It requires that the (partially) instantiated pattern specified in the read event should match any of the messages from the input buffer. Upon execution of the read event, this message is removed from the input buffer and the executing run advances to the next event. If the input buffer, contains more than one instance of the message, only one is removed. The *claim* rule expresses that an enabled claim event can always be executed. The *transmit* rule describes transmission of a message from the output buffer to the input buffer without interference from the intruder. Notice that in all these cases the intruder knowledge is not affected. The dynamical behaviour of the intruder knowledge will be defined by the network/intruder rules in Section 2.4.1.

$$[create]\frac{run = ((\theta, \rho, \sigma), s) \in runsof(P), \theta \notin runIDs(F)}{\langle M, BS, BR, F\rangle \xrightarrow{create(run)} \langle M, BS, BR, F \cup \{run\}\rangle}$$

$$[send]\frac{run = (inst, send_\ell(m) \cdot s) \in F}{\langle M, BS, BR, F\rangle \xrightarrow{(inst, send_\ell(m))} \langle M, BS \cup \{\langle inst\rangle(m)\}, BR, F[(inst, s)/run]\rangle}$$

$$[read]\frac{run = (inst, read_\ell(pt) \cdot s) \in F, m \in BR, Match(inst, pt, m, inst')}{\langle M, BS, BR, F\rangle \xrightarrow{(inst', read_\ell(pt))} \langle M, BS, BR \setminus \{m\}, F[(inst', s)/run]\rangle}$$

$$[claim]\frac{run = (inst, claim_\ell(R, c, t) \cdot s) \in F}{\langle M, BS, BR, F\rangle \xrightarrow{(inst, claim_\ell(R,c,t))} \langle M, BS, BR, F[(inst, s)/run]\rangle}$$

$$[transmit]\frac{m \in BS}{\langle M, BS, BR, F\rangle \xrightarrow{transmit(m)} \langle M, BS \setminus \{m\}, BR \cup \{m\}, F\rangle}$$

Table 2.3: Operational semantics rules.

Definition 2.38 (Initial network state). *In the initial state of the system both buffers are empty, and no runs have been created yet. Thus the initial state of the system is given by*

$$s_0 = \langle M_0, \emptyset, \emptyset, \emptyset\rangle$$

where M_0 refers to the initial intruder knowledge, which we define in the Section 2.4.2.

A state transition is the conclusion of finitely many applications of these rules, starting from the initial state. In this way, we can derive all possible behaviours of a system executing security protocol P. This is what we consider the operational semantics of P, and we write $traces(P)$ to denote the set of *traces*, or behaviours, of a protocol P. Each such a trace is a sequence of run events, and thus we have $traces(P) \in RunEvent^*$.

2.4 Threat model

In this section we address the threat model parameter. We discern two different elements of the threat model. First, we incorporate the possibility that the network

is partially or completely under control of an intruder. Second, there can be agents that are conspiring with, or compromised by an intruder. We address these elements in the next sections.

2.4.1 Network threat model

In the context of security protocol verification the Dolev-Yao intruder model is commonplace (see [79]). In this model, the intruder has complete control over the network. Messages can be learnt, deflected, and created by such an intruder. However, often this intruder model is too powerful, for example when an intruder can only eavesdrop on the network, or in the context of wireless communications. In such cases, it is desirable to develop lightweight protocols that are correct with respect to a weaker intruder model. Therefore, we parameterise over the intruder model, which is defined as a set of capabilities. Each intruder rule defines a capability by explaining the effect of the intruder action on the output buffer, the input buffer and the intruder knowledge (in other words, on the state, except the remaining runs.) In Table 2.4 we give some examples of intruder rules. If the intruder has eavesdropping capabilities, as stated in the *eavesdrop* rule, he can learn the message during transmission. The *take* rule states that an intruder with deflection capabilities can delete any message from the output buffer. The difference with the *jam* rule is that the intruder can read the deflected message and add it to its knowledge. The *fake* rule describes the injection of any message inferable from the intruder knowledge (conform Definition 2.12) into the input buffer.

$$[take]\frac{m \in BS}{\langle M, BS, BR, F\rangle \xrightarrow{take(m)} \langle M \cup \{m\}, BS \setminus \{m\}, BR, F\rangle}$$

$$[fake]\frac{M \vdash m}{\langle M, BS, BR, F\rangle \xrightarrow{fake(m)} \langle M, BS, BR \cup \{m\}, F\rangle}$$

$$[eavesdrop]\frac{m \in BS}{\langle M, BS, BR, F\rangle \xrightarrow{eavesdrop(m)} \langle M \cup \{m\}, BS \setminus \{m\}, BR \cup \{m\}, F\rangle}$$

$$[jam]\frac{m \in BS}{\langle M, BS, BR, F\rangle \xrightarrow{jam(m)} \langle M, BS \setminus \{m\}, BR, F\rangle}$$

Table 2.4: Network/intruder rules.

Next, we define some interesting intruders.

Example 2.39 (No intruder). *In a network without an intruder we only have the transmit rule as in Table 2.3 on page 27:*

$$Networkrulename ::= transmit(RunTerm)$$

Example 2.40 (Dolev-Yao intruder). *In the Dolev-Yao model the intruder has full control over the network. Hence, there is no direct transmit rule. Every message is read and analysed, and anything that can be constructed can be inserted into the network:*

$$Networkrulename ::= take(RunTerm) \mid fake(RunTerm)$$

Example 2.41 (Wireless communications intruder). *A wireless communication network is weaker than Dolev-Yao, because it does not allow learning from a message and preventing its arrival at the same time. Thus we define*

$$Networkrulename ::= eavesdrop(RunTerm) \mid jam(RunTerm) \mid fake(RunTerm)$$

Example 2.42 (Passive intruder). *If the intruder can only eavesdrop, we have*

$$Networkrulename ::= eavesdrop(RunTerm)$$

It is possible to construct more intruder rules, for intruder capabilities such as rerouting of messages or the modification of messages.

2.4.2 Agent threat model

In the agent threat model, we model the fact that a number of agents may be compromised by the intruder. Assuming that the intruder can insert messages into the network and read from it, the only thing that distinguishes the intruder from a real agent, is some particular piece of knowledge representing the identity of the agent. When an agent is compromised by the intruder, the intruder learns all the knowledge of this agent for all the roles. This allows the intruder to act as if it were one of the compromised agents in each of these roles.

The set *Agent* is partitioned into sets $Agent_T$ (denoting the *trusted agents*) and $Agent_U$ (denoting the *untrusted agents*).

Definition 2.43 (Untrusted agents). *An untrusted agent, typically named E, with $E \in Agent_U$, is an agent that has been compromised by the intruder. The initial role knowledge of this agent is part of the initial intruder knowledge.*

Note that in our set-up, the agents do not know which of the agents are trusted and which are not. Thus, trusted agents might still start protocol sessions with untrusted agents, or accept requests from untrusted agents.

Within our model, it is possible that any number of agents are compromised. In other security protocol models, this is often reduced to a single untrusted agent,

typically E. Whether or not this reduction is valid depends on the details of the (protocol)model and the specific security property that is tested.[2]

We assume agents are immediately compromised, and we therefore model them by adding their knowledge to the initial intruder knowledge. The intruder learns all possible instantiations of the initial knowledge of a role.

This assumes the intruder can generate local constants just as the agents can. However, in order to correctly model the fact that local constants (such as nonces) are uniquely generated, we must ensure these constants are syntactically different from the constants generated by any other runs in the system. In other words, all nonces generated by the intruder are distinct from a nonce $ni\sharp 1$ generated in run 1. We could express this by introducing a distinctness constraint for the nonces, effectively reserving certain run identifiers for the intruder generated constants. Instead, we choose to model the fresh constants generated by the intruder as elements of the set *IntruderConst*. Thus, the intruder can create any number of constants of each type, defined as the set *IntruderConst*, and we require that the intruder can generate values of each type. More formally, we have that $\forall V \in Var : \exists t : t \in type(V) \ \wedge \ t \in IntruderConst$.

The initial intruder knowledge will consist of e.g. the names and public keys of all agents, and the secret keys of the untrusted agents.

To instantiate the role knowledge, we only need to know how the role names are mapped to agent names, and information about a run or instantiation of the variables is not needed. For a protocol P, an untrusted agent E in a role R, and a term rt that is part of the initial role knowledge of R that does not contain local constants, the knowledge that is passed to the intruder is defined as all possible instantiations of rt for which R is instantiated to E. The full initial intruder knowledge consists of these terms for all roles and all untrusted agents.

Definition 2.44 (Initial intruder knowledge). *For a protocol P, we define the initial intruder knowledge as the union of this knowledge of all untrusted agents in all roles:*

$$M_0 = IntruderConst \cup \bigcup_{\substack{R \in Role \\ \rho \in Role \to Agent \\ \rho(R) \in Agent_U}} \left\{ \langle \_, \rho, \_ \rangle(rt) \;\middle|\; rt \in MR(R) \wedge \forall rt' \sqsubseteq rt : rt' \notin Const \right\}$$

Example 2.45 (Initial intruder knowledge for the *NS* protocol). *For the*

[2] To give an idea of an example in which this reduction is not valid, consider a protocol of three roles $R1, R2, R3$, for which there is a requirement for each run that $\rho(R1) \neq \rho(R2) \neq \rho(R3)$. Now, if there is an attack on the protocol (along the lines of the Needham-Schroeder attack presented in the next chapter) that requires an agent (e. g. in role $R1$) to communicate with untrusted agents (in roles $R2, R3$), then the attack requires that there is more than one untrusted agent. In such a case the reduction of $Agent_T = \{E\}$ would not be valid, as this would cause attacks on the protocol to be missed.

NS protocol, we have that

$$M_0 = \mathit{IntruderConst} \cup \mathit{Agent} \cup \\ \{\mathit{pk}(\mathit{Ag}) \mid \mathit{Ag} \in \mathit{Agent}\} \cup \{\mathit{sk}(\mathit{Ag}) \mid \mathit{Ag} \in \mathit{Agent}_U\}$$

In this particular case, the initial role knowledge of both roles contributes exactly the same information to M_0. We consider role i, with $MR(i) = \{i, r, sk(i), pk(i), pk(r)\}$. When such a role is executed by an untrusted agent, we have that $\rho(i) \notin \mathit{Agent}_T$. No such restriction holds for the communication partner, and thus $\rho(r) \in \mathit{Agent}$. Because the initial role knowledge contains r, a compromised run can therefore contain any agent name, yielding $\mathit{Agent} \subseteq M_0$. Second, because the role knowledge contains $pk(r)$, we find that compromised agents also reveal the public keys of all the agents. Finally, because $sk(i) \in MR(i)$, we have that the secret keys of the untrusted agents are known to the intruder.

Before any events have occurred in a trace, the intruder knowledge is denoted by M_0. Some intruder events (such as take or eavesdrop) cause the intruder knowledge to change. Informally stated, we will write M_i^t to denote the intruder knowledge just before execution of the event t_i.

Definition 2.46 (Intruder knowledge after execution of events). *Let t be a trace, and let i be a trace index with $0 < i \leq |t|$. We write M_i^t to denote the intruder knowledge after execution of the events $t_0, t_1, \ldots, t_{i-1}$.*

Where the trace t is clear from the context, we will simply write M_i.

Note that if we include the *take* or *eavesdrop* intruder rules from the previous section in the model (as with the Dolev-Yao model), the intruder can potentially eavesdrop on any message that is being sent. However, the intruder cannot learn more than that which can be inferred from the sent messages.

Lemma 2.47 (Maximum intruder knowledge after execution of events). *For any trace t and a trace index i with $i \leq |t|$, we have that*

$$M_i^t \subseteq \{\langle \mathit{inst} \rangle(m) \mid t_j = (\mathit{inst}, \mathit{send}_\ell(R1, R2, m)) \wedge j < i \wedge \\ j \in N \wedge \mathit{inst} \in \mathit{Inst} \wedge \ell \in \mathit{Label} \wedge R1, R2 \in \mathit{Role} \wedge m \in \mathit{RoleTerm}\}$$

Proof. For all intruder rules, we have that the intruder knowledge set M is changed only by adding elements from the send buffer BS. □

2.5 Conclusions

We have developed a generic model for fundamental analysis of security protocols. Some characteristics of this model are that we give explicit static requirements for valid protocols, and that the model is parametrized over the matching function and

intruder network capabilities. Multi-protocol analysis, by which we mean the analysis of running several different protocols or protocol roles concurrently, is handled in an intuitive way by simply adding more role descriptions to the model. In line with this, security properties are defined as local claims. Furthermore, local constants are bound to runs, which can assist in the construction of proofs.

Given the (inherent) complexity of such a model, one might wonder whether the model is *correct*. Here, we have chosen to start from basic (intuitive) concepts. Furthermore, the operational semantics make it possible to directly compare the model with the implementation. The consequences of both these decisions allow for some degree of validation. Further validation of the model will come from the results of the corresponding tool (in Chapter 4) and case studies (in Chapters 5 and 6). A strict formalisation of the model in a theorem proving environment would allow for further validation, but this is beyond the scope of this thesis. Formalizing the model within such an environment is therefore listed as future work in Chapter 8.

This completes our description of the base model for security protocol description. We proceed by defining and explaining security claims in the next chapter.

3

Security Properties

In the previous chapter we presented a model for the description of security protocols and their executions in terms of traces. In this chapter we will formally introduce security properties into our model. We define here only forms of authentication and secrecy. This does not mean that other properties cannot be expressed in the model. Rather, it reflects our choice of base properties that can serve as a starting point from which other trace properties can be easily defined.

We start off by defining the notion of *claim events* in Section 3.1. Then we address two classes of properties: secrecy in Section 3.2 and authentication in Section 3.3. We proceed by establishing in Section 3.4 a hierarchy on various strong forms of authentication, and discuss some relative merits of the properties. We conclude by establishing a useful result on the strongest form of authentication (synchronisation) with respect to so-called injectivity in Section 3.5.

3.1 Security properties as claim events

For our model as introduced thus far, we have several options for formalizing security properties. As a first choice, we can decide to integrate the properties into the protocol specification, or have the properties as separate entities, e.g. using (temporal) logic formulas. We choose to integrate the security properties into the specification of the protocol. The main reason for this is that we consider security properties to be an essential part of a security protocol, and a security protocol should not be considered without knowing the exact properties it is supposed to satisfy.

We integrate the security properties into our model by introducing a special type of role event: the *claim event*. The main idea behind the concept of a claim event is locality: agents have a local view on the state of the system, based on the messages they receive. The protocol should guarantee that based on the local view, the agent can be sure about some properties of the global state of the system, e.g. that something is not in the intruder knowledge, or that a certain agent is active.

To provide some intuition about local claim events, consider the protocol in Figure 3.1 on the next page. We will give a formal definition of the claim events later. Just for this first example, we will rely on the intuition of the reader as to what "secrecy of a term" means: if an agent communicates with non-compromised agents,

the term in question should be secret in every possible trace of the protocol.

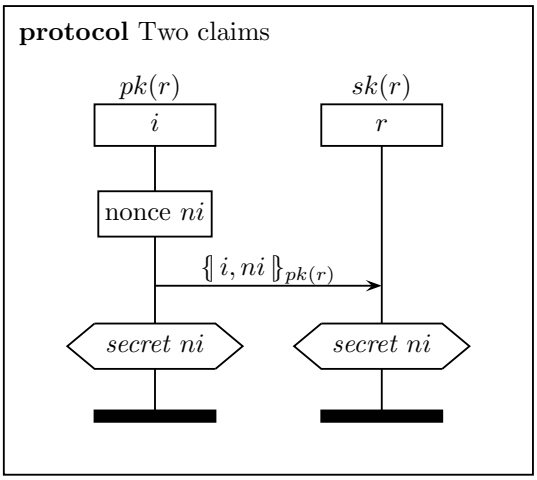

Figure 3.1: A protocol with two claim events

If we analyse the protocol in Figure 3.1, we find that if an agent completes the initiator role whilst communicating with a trusted agent, he can be sure that the term ni he created is secret. As the nonce is encrypted with the public key of the intended recipient, which is trusted, then only this agent can decrypt it. Whenever a run of the i role is completed with trusted communication partners, the nonce ni generated in the run will not become known to the intruder: We say that the claim *secret ni* of the role i holds.

However, this does not mean the protocol guarantees some form of global secrecy. In particular, if an agent completes the responder role, he cannot be sure that the received term is secret. In Figure 3.2 on the next page we show an attack on the protocol in 3.1. The intruder knows the public key of A, and thus he can simply generate a message containing a nonce ne himself. The agent A will accept the message, as there is no authentication performed on the origin of the message, which seems to come from B. We say that the claim *secret ni* of the role r does not hold. Such a falsification of a claim constitutes an attack.

The difference between the local views on security by the different roles is captured by the local claim events. In this particular case, the claim of i is true, but the claim of r is obviously false.

Example 3.1 (Structure of a claim definition). *Let γ be a claim role event of a protocol P. For the security properties defined in this chapter, we will require that some predicate Q on $traces(P) \times ClaimRunEv$ holds for each instance of γ in all traces of P. We will say that the property is true for the protocol if and only if*

$$\forall t \in traces(P)\ \forall (inst, \gamma) \in t : Q(t, (inst, \gamma))$$

where we use the notation $e \in t$ as an abbreviation for $\exists i : t_i = e$.

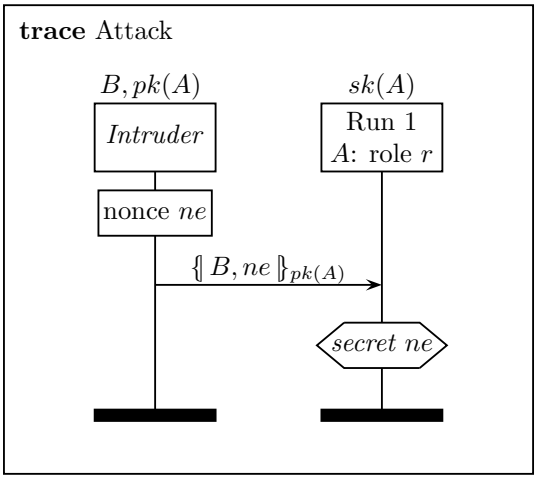

Figure 3.2: An attack

In the next sections we will see how secrecy and authentication can be defined in terms of claim events.

3.2 Secrecy

As a first example of a security property we define secrecy in our model. Secrecy expresses that certain information is not revealed to an intruder, even though we are communicating this data over an untrusted network.

It is possible to define various forms of secrecy, with subtle differences between them. We start off with a basic form of secrecy. We introduce a secrecy claim event, written as $claim(R, secret, S)$, executed in role R that takes a term S as a parameter. Roughly stated, for all executions of the protocol role, the term S should be secret, i.e. not known to the intruder.

Of course, given the trust relations sketched in the previous chapter, agents communicating (secret) data with untrusted agents are effectively sharing their information with the intruder. Although the communicated terms are not secret anymore, this does not mean that the protocol is broken. Rather, we want a secrecy claim to mean that if an agent communicates with trusted agents, then we want the data to be secret.

Definition 3.2 ($SECRET$). *A protocol P with a role R with claim protocol event $\gamma = claim(R, secret, S)$ satisfies* secrecy *of S, notation $SECRET(P, \gamma)$, if and only if*

$$\forall t \in traces(P)\ \forall((\theta, \rho, \sigma), \gamma) \in t\ :\ rng(\rho) \subseteq Agent_T \Rightarrow \forall i : M_i^t \not\vdash \langle\theta, \rho, \sigma\rangle(S)$$

The definition states that a secrecy claim γ in a protocol is true, if and only if we have the following for all traces: for each run in which roles are mapped to

trusted agents only, and secrecy of a term is claimed, the term should never be in the knowledge of the intruder.

This definition allows us to express secrecy of terms. This also includes variables (as in the responder role in the example). The responder does not generate the nonce, but if the protocol is correct, an agent completing the responder role should be ensured that the term he received is secret.

In the example attack from Figure 3.2 on the preceding page we can see that the claim does not hold. In the attack the variable ni of the responder role is instantiated with a term that is under control of the intruder.

In Section 3.6 we illustrate this definition by sketching a proof of a secrecy claim of the Needham-Schroeder-Lowe protocol.

3.3 Authentication

The security property studied the most in the field of security protocol analysis is *authentication*. However, contrary to the requirement of *secrecy*, there is no general consensus on the meaning of authentication. In fact, as indicated by Lowe [122], there is a hierarchy of authentication properties. We will return to this in Section 3.4.

In its most basic form, authentication is a simple existential statement about a communication partner. A protocol description, especially when written in the form of a Message Sequence Chart, suggests that at least two agents are communicating. However, because the network can be under the control of an intruder, not every role execution guarantees that there actually has been a communication partner, as e. g. in the attack in Figure 3.2.

Authentication focusses on establishing that executing a protocol role actually guarantees that there is at least a communication partner in the network. In most cases we want to establish something stronger, e.g. that the intended partner is aware he is communicating with us, and that some protocol is run, and that messages have been exchanged as expected.

In the next section we will start off by defining a class of simple authentication properties, which is often called *Aliveness*.

3.3.1 Aliveness

We define the notion of generic aliveness as a security claim with a parameter R. The definition includes a predicate AL which will be used below to define several specific forms of aliveness.

Definition 3.3 (Generic Aliveness). *A protocol P with a claim protocol event $\gamma = claim(R, alive, R')$ in role R with parameter R', is said to satisfy* generic

aliveness of role R' *if and only if*

$$\forall t \in traces(P)\ \forall (inst, \gamma) \in t : \{\langle inst\rangle(R), \langle inst\rangle(R')\} \subseteq Agent_T \Rightarrow \\ \exists (inst', \varepsilon') \in t : \langle inst'\rangle((role(\varepsilon'))) = \langle inst\rangle(R') \wedge AL$$

where AL is some predicate, denoting a specific type of aliveness.

The definition tells us that when an agent executes a role specification up to the claim event, and he thinks he is talking to an agent that is trusted, then the intended communication partner has actually executed an event.

Using the predicate AL, we can refine this definition in several ways. We give a few examples (in accordance with e.g. [122]): weak aliveness, weak aliveness in the correct role, and recent aliveness.

Weak Aliveness The notion of weak aliveness corresponds to the weakest possible form of the generic aliveness class sketched above, i.e.

$$AL \equiv True$$

It simply states the intended partner is at least alive, hence the name of the class.

Example 3.4. *A protocol where an agent sends the text "Hello, I am Alice." in plaintext to Bob, where there is only a passive (eavesdropping) intruder, satisfies weak aliveness: when Bob receives the message, he can be sure Alice has been active. However, for the Dolev-Yao intruder model, it does not guarantee weak aliveness. The intruder can generate such a message, even if there is no Alice. Thus, the protocol would need to be strengthened, e.g. by having Alice sign the message with her secret key.*

Weak Aliveness in the correct role Knowing that the communication partner is alive is often not enough. In most cases we at least require he is executing some particular role, as could be expected from the protocol description.

$$AL \equiv role(\varepsilon') = R'$$

Recent Aliveness Although our model does not include a notion of time, we can give an interpretation of recentness. The previous notions of aliveness guaranteed that the communication partner is alive, but it does not tell us whether this was before, after, or during the run that makes the claim. Recent aliveness guarantees that an event took place during our run.

$$AL \equiv \exists inst'', \varepsilon'' : \\ (inst'', \varepsilon'') < (inst', \varepsilon') < (inst, \gamma)\ \wedge\ runidof(inst'') = runidof(inst)$$

3.3.2 Synchronisation

The notions of aliveness above have one main thing in common. They only consider a role R' (which is a parameter of the claim event) from the perspective of the role R (in which the claim occurs), and require various properties of the agent performing this particular role R'. If a specification contains two roles, then upon completion of one role, we expect there to be an agent performing a run in the other role. Similarly, the protocol specification suggests that there are runs for each role in the protocol, and messages are exchanged between these runs. The specification furthermore suggests an ordering on the events, and unmodified delivery of messages. If an agent executes a role up to some point, the protocol description suggests that some events have occurred as expected.

In the presence of an active intruder, other behaviour (not defined by the protocol description) might occur. For example, if there is no agent performing a certain role, this contradicts the protocol description. We consider any behaviour that is not specified, to be unauthenticated. This leads to a natural definition of strong authentication, which we will call *synchronisation*.

In order to define synchronisation in a concrete trace, we need some mechanism to pinpoint which agents are performing which roles. As a protocol description can consist of a number of roles, which can be instantiated any number of times, we need some way to express which run is (supposedly) communicating with which other runs. Therefore, we introduce the notion of a cast, borrowing intuition from a theatre play that is performed several times. The cast for a particular performance of the play relates activity in the performance to particular roles. Likewise, the concrete activities in a protocol instance at the trace level, are assigned to the roles in the protocol. In the theater case, in different performances an actor can play different roles. Also, the same role can be taken up, for different performances of the play, by different actors. Likewise, run events associated with different roles may belong to the same agent and run events that are instances for the same role may belong to different agents. For our purposes the coupling of roles and run events is more important, than the coupling of roles and actors. Therefore, we have the following definition.

Abusing notation, we write $role(\theta)$ to denote the role that the run θ is an instance of, and $runidof(c)$ to denote the run identifier of a run event.

Definition 3.5 (*Cast*). *A mapping* $\Gamma\colon RunEvent \to Role \to RID$ *is called a* cast function *for a trace* t *of the protocol* P *if*

$$\Gamma(c)(R) = \theta \implies role(\theta) = R \tag{3.1}$$

for all $\theta \in RID$, $c \in RunEvent$, *where* c *is a claim event,* $R \in dom(P)$, *and*

$$\Gamma(c)(role(c)) = runidof(c) \tag{3.2}$$

for all $c \in ClaimRunEv$.

A cast function $\Gamma\colon RunEvent \to Role \to RID$ *is called an* injective *cast function, if*

$$\Gamma(c)(R) = \Gamma(c')(R') \implies c = c' \wedge R = R'$$

for every two claim run events c, c' and every two roles R, R'. We use $Cast(P,t)$ to denote the collection of all cast functions for a trace t of the protocol P.

The idea behind the notion of a cast function is the assignment of roles in the context of a concrete claim event c. We require that the run identifier assigned to a certain role is actually executing that role, as captured by condition (3.1). Furthermore, we require in condition (3.2) that for a claim in run θ, the claiming role is assigned the run identifier θ. At this point, we do not require that every role will be performed completely, leaving room for unfinished role executions.

An injective cast function is, with abuse of language, an injective mapping when considered to be of functionality $RunEvent \times Role \to RID$, rather than injective as a function of type $RunEvent \to Role \to RID$. The main reason of sticking to the latter function type is the underlying intuition of a cast. The mapping $\Gamma(c)$ captures the perspective of the agent executing the instance of the role of the claim run event c.

Example 3.6. *As a running example to illustrate the synchronisation property, we will use a classical example: the short version of the Needham-Schroeder protocol from [145], depicted in Figure 2.1 on page 12.*

We have $Role = \{ i,\ r \}$ as the set of roles with initiator i and responder r, and for the set of protocol events we put

$$\begin{aligned} RoleEvent = \{\ & send_1(i,r,\{\!|\, i, ni \,|\!\}_{pk(r)}),\ read_1(i,r,\{\!|\, i, W \,|\!\}_{pk(r)}), \\ & send_2(r,i,\{\!|\, W, nr \,|\!\}_{pk(i)}),\ read_2(r,i,\{\!|\, ni, V \,|\!\}_{pk(i)}), \\ & send_3(i,r,\{\!|\, V \,|\!\}_{pk(r)}),\ read_3(i,r,\{\!|\, nr \,|\!\}_{pk(r)}), \\ & claim_4(i, ni\text{-}synch),\ claim_5(r, ni\text{-}synch)\ \} \end{aligned}$$

with the following role assignment

$$\begin{aligned} role^{-1}(i) = \{\ & send_1(i,r,\{\!|\, i, ni \,|\!\}_{pk(r)}), \\ & read_2(r,i,\{\!|\, ni, V \,|\!\}_{pk(i)}), send_3(i,r,\{\!|\, V \,|\!\}_{pk(r)}), claim_4(i, ni\text{-}synch)\ \} \\ role^{-1}(r) = \{\ & read_1(i,r,\{\!|\, i, W \,|\!\}_{pk(r)}), \\ & send_2(r,i,\{\!|\, W, nr \,|\!\}_{pk(i)}), read_3(i,r,\{\!|\, nr \,|\!\}_{pk(r)}), claim_5(r, ni\text{-}synch)\ \}. \end{aligned}$$

Observe that the nonce ni of the initiator is stored in a variable W by the responder side. Likewise, the responder nonce nr is stored by the initiator in the variable V.

Note that on the one hand, we have for the protocol order $\preccurlyeq_P$ of NS that

$$send_3(i,r,\{\!|\, V \,|\!\}_{pk(r)}) \preccurlyeq_P read_3(i,r,\{\!|\, nr \,|\!\}_{pk(r)}) \preccurlyeq_P claim_5(r, ni\text{-}synch)$$

but, on the other hand,

$$send_3(i,r,\{\!|\, V \,|\!\}_{pk(r)}) \preccurlyeq_P read_3(i,r,\{\!|\, nr \,|\!\}_{pk(r)}) \not\preccurlyeq_P claim_4(i, ni\text{-}synch)$$

Such a situation is relevant to the notion of synchronisation. For the claim event $claim_4(i, ni\text{-}synch)$ of the initiator role there are only two preceding pairs of send and read events, viz. those of the first two messages. For the claim $claim_5(r, ni\text{-}synch)$ there are three preceding communication pairs, viz. all three messages.

For the Needham-Schroeder protocol discussed above, the trace representing the well-known Lowe attack [119] can have the following form. Let $\rho 1, \rho 2 : \mathit{Role} \to \mathit{Agent}$ *be defined as*

$$\rho 1 = \{i \mapsto A, r \mapsto E\}$$
$$\rho 2 = \{i \mapsto A, r \mapsto B\}$$

Then, the attack is given as the trace

$$\begin{array}{l}((1, \rho 1, \emptyset), \mathrm{send}_1(i, r, \{\!|\, i, ni\sharp 1 \,|\!\}_{pk(r)})) \cdot \\ \quad \mathrm{take}(A, E, \{\!|\, A, ni\sharp 1 \,|\!\}_{pk(E)}) \cdot \mathrm{fake}(A, B, \{\!|\, A, ni\sharp 1 \,|\!\}_{pk(B)}) \cdot \\ \quad\quad ((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{read}_1(i, r, \{\!|\, i, W \,|\!\}_{pk(r)})) \cdot \\ \quad\quad ((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{send}_2(r, i, \{\!|\, W, nr\sharp 2 \,|\!\}_{pk(i)})) \cdot \\ \quad\quad\quad \mathrm{take}(B, A, \{\!|\, ni\sharp 1, nr\sharp 2 \,|\!\}_{pk(A)}) \cdot \mathrm{fake}(E, A, \{\!|\, ni\sharp 1, nr\sharp 2 \,|\!\}_{pk(A)}) \cdot \\ \quad\quad\quad\quad ((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{read}_2(r, i, \{\!|\, ni\sharp 1, V \,|\!\}_{pk(i)})) \cdot \\ \quad\quad\quad\quad ((1, \rho 1, \{V \mapsto nr\sharp 2\}), \mathrm{send}_3(i, r, \{\!|\, V \,|\!\}_{pk(r)})) \cdot \\ \quad\quad\quad\quad ((1, \rho 1, \{V \mapsto nr\sharp 2\}), \mathrm{claim}_4(i, \textit{ni-synch})) \cdot \\ \quad\quad\quad\quad\quad \mathrm{take}(A, E, \{\!|\, nr\sharp 2 \,|\!\}_{pk(E)}) \cdot \mathrm{fake}(A, B, \{\!|\, nr\sharp 2 \,|\!\}_{pk(B)}) \cdot \\ \quad\quad\quad\quad\quad\quad ((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{read}_3(i, r, \{\!|\, nr\sharp 2 \,|\!\}_{pk(r)})) \cdot \\ \quad\quad\quad\quad\quad\quad ((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{claim}_5(r, \textit{ni-synch}))\end{array}$$

The attack is also shown in a graphical form in Figure 3.3 on the facing page. This attack violates the ni-synch claim of the r role. It uses the take *and* fake *intruder rules. In the attack, an agent A executes the initiator role, trying to communicate to an agent E. Unfortunately, E has been compromised. The intruder takes the message, and decrypts it which is possible because it knows the private key of E. The intruder encrypts the message again, but now using the public key of B, and fakes a messages from A to B. Now B replies to A. This message is encrypted with the public key of A, so the intruder cannot decrypt this message, but it can change the original sender address to E. A thinks E has replied, and thus it decrypts the second challenge* $nr\sharp 2$*, and encrypts it with the public key of E. Thus, the intruder learns the value* $nr\sharp 2$*, and can complete the protocol with B. This trace violates a form of authentication, because B thinks he is talking to A, but in fact A is talking to somebody else. Although neither A nor B are compromised, the intruder can impersonate as A to B.*

Example 3.7. *In the context of the Lowe attack trace introduced above, we have, e.g., as a cast the mapping* Γ *such that*

$$\begin{array}{l}\Gamma((\mathit{inst}1, \mathrm{claim}_4(i, \textit{ni-synch})))(i) = 1 \\ \Gamma((\mathit{inst}1, \mathrm{claim}_4(i, \textit{ni-synch})))(r) = 2 \\ \Gamma((\mathit{inst}2, \mathrm{claim}_5(r, \textit{ni-synch})))(i) = 1 \\ \Gamma((\mathit{inst}2, \mathrm{claim}_5(r, \textit{ni-synch})))(r) = 2\end{array}$$

where $\mathit{inst}1 = (1, \rho 1, \{V \mapsto nr\sharp 2\})$ *and* $\mathit{inst}2 = (2, \rho 2, \{W \mapsto ni\sharp 1\})$.

Note that the cast Γ *is bounded by the general restrictions of Definition 3.5 on page 38. In particular, in this case we use the first restriction on the cast function,*

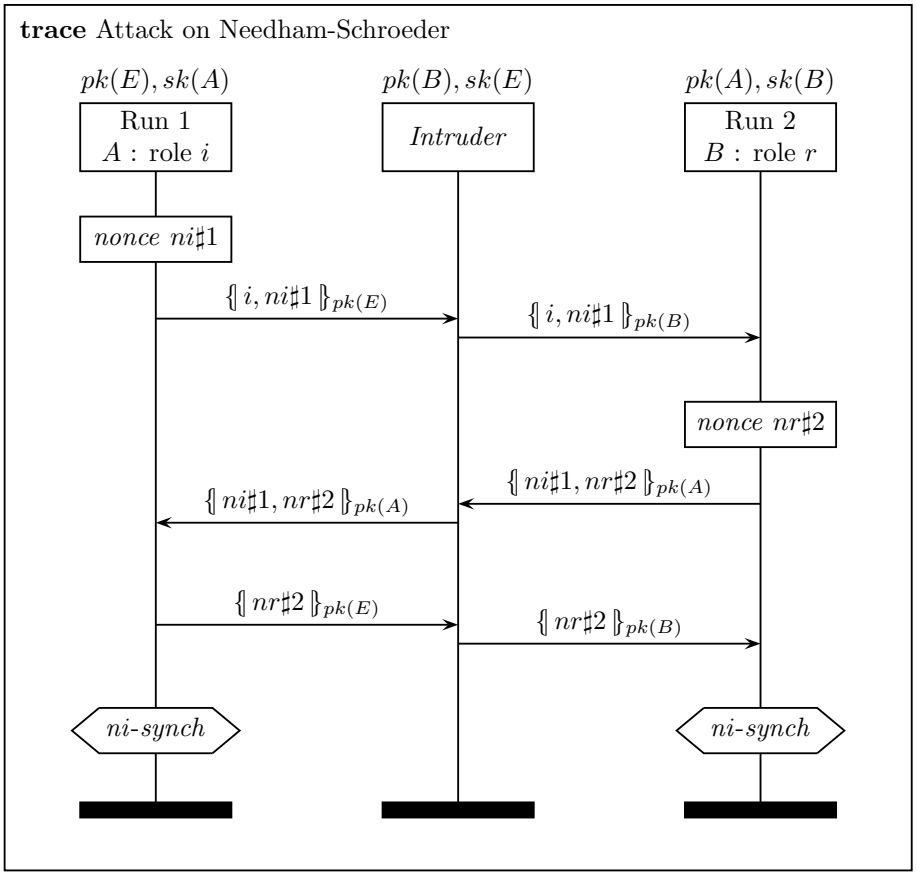

Figure 3.3: Attack on Needham-Schroeder

stating that for any claim event, a role must map to a run which is an instance of that same role. Because of the limited trace we consider, there is in fact no choice for the cast function. However, in general, traces can contain many interleavings of runs, in which case there would be several alternatives.

Regarding the first two clauses above, the completely genuine run events of agent A are matched by activity of the intruder (as E is a untrusted agent). Just bad luck for agent A, but no guarantees can be made for agents communicating with untrusted agents such as E. The situation for the latter two clauses is more seriously wrong. Here, the role of responder of a trusted agent B, who assumes he is talking to another trusted agent A, is matched by activity of agent A engaged in an protocol session with agent E.

Using the cast function, we introduce an authentication property that captures the following correspondence: at the trace level, we require that the same structures

occur as the ones found at the protocol description level. Informally put, we require that everything we intended to happen in the protocol description also actually happens in the trace.

We first give the definition of the authentication property that we call non-injective synchronisation, and explain it in detail below.

Recall that the function $cont(e)$ extracts the instantiated contents (of the form (a, b, m)) from a (send or read) run event e.

Definition 3.8 (*NI-SYNCH*). *A protocol P with a claim role event γ is called* non-injectively synchronising*, notation $NI\text{-}SYNCH(P, \gamma)$, if*

$$\begin{aligned}
&\forall t \in traces(P)\ \exists \Gamma \in Cast(P,t) \\
&\quad \forall c \in t,\ c = ((\theta, \rho, \sigma), \gamma),\ rng(\rho) \subseteq Agent_T, \\
&\qquad \forall \varsigma, \varrho \colon \varsigma \rightsquigarrow \varrho \preccurlyeq_P \gamma\ \exists s, r \colon s <_t r <_t c \\
&\qquad\quad roleevent(s) = \varsigma\ \wedge\ runidof(s) = \Gamma(c)(role(\varsigma)) \wedge \\
&\qquad\quad roleevent(r) = \varrho\ \wedge\ runidof(r) = \Gamma(c)(role(\varrho)) \wedge \\
&\qquad\quad cont(s) = cont(r).
\end{aligned}$$

where we write $s <_t r$ to denote that the run event r is preceded by the run event s in the trace t.

Non-injective synchronisation is a trace property for a protocol P and a claim role event γ. In particular, each trace t of the protocol P can contain a number of instances of the claim event γ. We only consider the valid instances of these claims, i.e. the claims of agents that communicate with agents that have not been compromised. For each of these instances of the claim, we require that there are actual communication partners. Thus, for all of these claims, there must exist runs that fulfill the roles of the protocol. This is expressed by the existence of the Γ function, which assigns the communication partner runs for each claim instance and role of the protocol.

Given an assignment of communication partners by the cast function Γ, we require, for each claim instance c that the communications have occurred as expected. This requirement must hold for each communication pair (s, r) that precedes the claim role event, expressed as $\forall \varsigma, \varrho : \varsigma \rightsquigarrow \varrho \preccurlyeq_P \gamma$.

In order for the communication $\varsigma \rightsquigarrow \varrho$ to have occurred correctly from the viewpoint of the claim run event c, there must exist actual send and read events s and r, such that the following three conditions for correct communications hold: the order of the events must be correct, the events are instantiations of the right role events by the runs as defined in Γ, and the message must be communicated correctly.

For the first condition regarding the ordering in the trace, we require that just as in the protocol description, $s <_t r <_t c$ holds. The second condition requires that the run events are indeed the events corresponding to the correct read and send events from the protocol, and that they are part of the runs as assigned by the Γ function. For the third condition, we simply require that the contents of the read message must be identical to the content of the message sent.

Example 3.9. *Concretely, for the case of the Needham-Schroeder protocol NS, we have that NI-SYNCH does not hold for the claim event* $claim_5(r, ni\text{-}synch)$ *of the responder. Consider again the trace below, where* $\rho 1 = \{i \mapsto A, r \mapsto E\}$ *and* $\rho 2 = \{i \mapsto A, r \mapsto B\}$*:*

$$
\begin{array}{l}
((1, \rho 1, \emptyset), \mathrm{send}_1(i, r, \{\!|\, i, ni\sharp 1 \,|\!\}_{pk(r)})) \cdot \\
\quad \mathrm{take}(A, E, \{\!|\, A, ni\sharp 1 \,|\!\}_{pk(E)}) \cdot \mathrm{fake}(A, B, \{\!|\, A, ni\sharp 1 \,|\!\}_{pk(B)}) \cdot \\
\qquad ((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{read}_1(i, r, \{\!|\, i, W \,|\!\}_{pk(r)})) \cdot \\
\qquad ((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{send}_2(r, i, \{\!|\, W, nr\sharp 2 \,|\!\}_{pk(i)})) \cdot \\
\qquad\quad \mathrm{take}(B, A, \{\!|\, ni\sharp 1, nr\sharp 2 \,|\!\}_{pk(A)}) \cdot \mathrm{fake}(E, A, \{\!|\, ni\sharp 1, nr\sharp 2 \,|\!\}_{pk(A)}) \cdot \\
\qquad\qquad ((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{read}_2(r, i, \{\!|\, ni\sharp 1, V \,|\!\}_{pk(i)})) \cdot \\
\qquad\qquad ((1, \rho 1, \{V \mapsto nr\sharp 2\}), \mathrm{send}_3(i, r, \{\!|\, V \,|\!\}_{pk(r)})) \cdot \\
\qquad\qquad ((1, \rho 1, \{V \mapsto nr\sharp 2\}), \mathrm{claim}_4(i, ni\text{-}synch)) \cdot \\
\qquad\qquad\quad \mathrm{take}(A, E, \{\!|\, nr\sharp 2 \,|\!\}_{pk(E)}) \cdot \mathrm{fake}(A, B, \{\!|\, nr\sharp 2 \,|\!\}_{pk(B)}) \cdot \\
\qquad\qquad\qquad ((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{read}_3(i, r, \{\!|\, nr\sharp 2 \,|\!\}_{pk(r)})) \cdot \\
\qquad\qquad\qquad ((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{claim}_5(r, ni\text{-}synch))
\end{array}
$$

representing the Lowe attack considered before. We want to show that for this trace, there exists no cast for the claim run event $((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{claim}_5(r, ni\text{-}synch))$ *that satisfies NI-SYNCH. Given the requirement (3.2) of Definition 3.5 on page 38, we have*

$$\Gamma(((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{claim}_5(r, ni\text{-}synch)))(r) = 2$$

We have as single possibility of $\Gamma(((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{claim}_5(r, ni\text{-}synch)))(i)$*, by condition (3.1) of Definition 3.5 on page 38,*

$$\Gamma(((2, \rho 2, \{W \mapsto ni\sharp 1\}), \mathrm{claim}_5(r, ni\text{-}synch)))(i) = 1$$

However, none of the two read run events will have counterparts in the run 1 *that match in content (as the public keys* $pk(B)$ *and* $pk(E)$ *are different) as is required by the fact that*

$$\begin{array}{l}
send_1(i, r, \{\!|\, i, ni \,|\!\}_{pk(r)}) \rightsquigarrow read_1(i, r, \{\!|\, i, W \,|\!\}_{pk(r)}) \preccurlyeq_P claim_5(r, ni\text{-}synch) \wedge \\
send_3(i, r, \{\!|\, V \,|\!\}_{pk(r)}) \rightsquigarrow read_3(i, r, \{\!|\, nr \,|\!\}_{pk(r)}) \preccurlyeq_P claim_5(r, ni\text{-}synch).
\end{array}$$

On the other hand, NI-SYNCH$(NS, claim_4(i, ni\text{-}synch))$ *holds for this protocol, but this cannot be determined from the example trace. In this trace, the run containing this claim is communication with an untrusted agent E, and thus it can be disregarded.*

Protocols satisfying non-injective synchronisation may still be vulnerable to so-called *replay attacks*. In a replay attack the intruder replays a message taken from a different context, thereby fooling the honest participants into thinking they have successfully completed the protocol run. See [127].

Example 3.10 (Injectivity). *The protocol in Figure 3.4 on the following page shows an example of a protocol that satisfies synchronisation, while Figure 3.5 on page 45 shows a replay attack on this protocol.*

The intruder can eavesdrop the message sent and can fool i in a future run to think that r has sent this message.

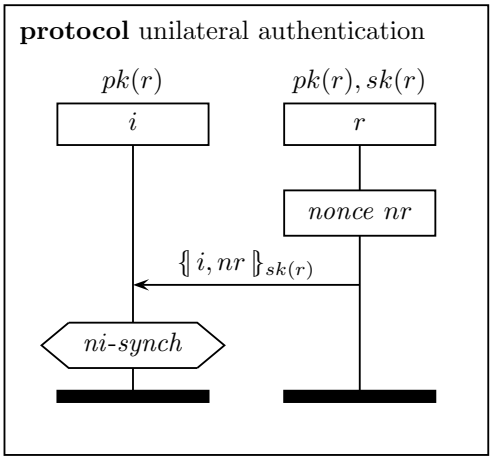

Figure 3.4: A protocol vulnerable to a replay attack

In order to rule out such flawed protocols, the additional property of *injectivity* is required. For each instance of a claim, we expect there to be a *unique* set of runs that execute the other roles as expected.

In order to capture the requirement that each claim instance corresponds to a unique set of runs fulfilling the other roles, we modify *NI-SYNCH* slightly and require that the cast function is injective, yielding the notion of injective synchronisation.

Definition 3.11 (*I-SYNCH*). *A protocol P with a claim protocol event γ is called* injectively synchronising*, notation $I\text{-}SYNCH(P, \gamma)$, if*

$$\begin{aligned}
&\forall t \in \mathit{traces}(P)\ \exists \Gamma \in \mathit{Cast}(P,t),\ \mathit{injective}\\
&\quad \forall c \in t,\ c = ((\theta, \rho, \sigma), \gamma),\ \mathit{rng}(\rho) \subseteq \mathit{Agent}_T,\\
&\qquad \forall \varsigma, \varrho\colon \varsigma \rightsquigarrow \varrho \preccurlyeq_P \gamma\ \exists s, r\colon s <_t r <_t c\\
&\qquad\quad \mathit{roleevent}(s) = \varsigma\ \wedge\ \mathit{runidof}(s) = \Gamma(c)(\mathit{role}(\varsigma)) \wedge\\
&\qquad\quad \mathit{roleevent}(r) = \varrho\ \wedge\ \mathit{runidof}(r) = \Gamma(c)(\mathit{role}(\varrho)) \wedge\\
&\qquad\quad \mathit{cont}(s) = \mathit{cont}(r).
\end{aligned}$$

Example 3.12. *For the Needham-Schroeder protocol in Figure 2.1 on page 12, I-SYNCH holds for the role i: for each instance of the claim of role i in a trace, there must also be a unique instance of the role r to synchronise with.*

3.3.3 Message agreement

Of all forms of authentication, the *agreement* family as defined by Lowe in [122] is the most well-known. All the definitions presented by Lowe focus not on the expectations from the protocol specification, but on agreement over some terms between two agents.

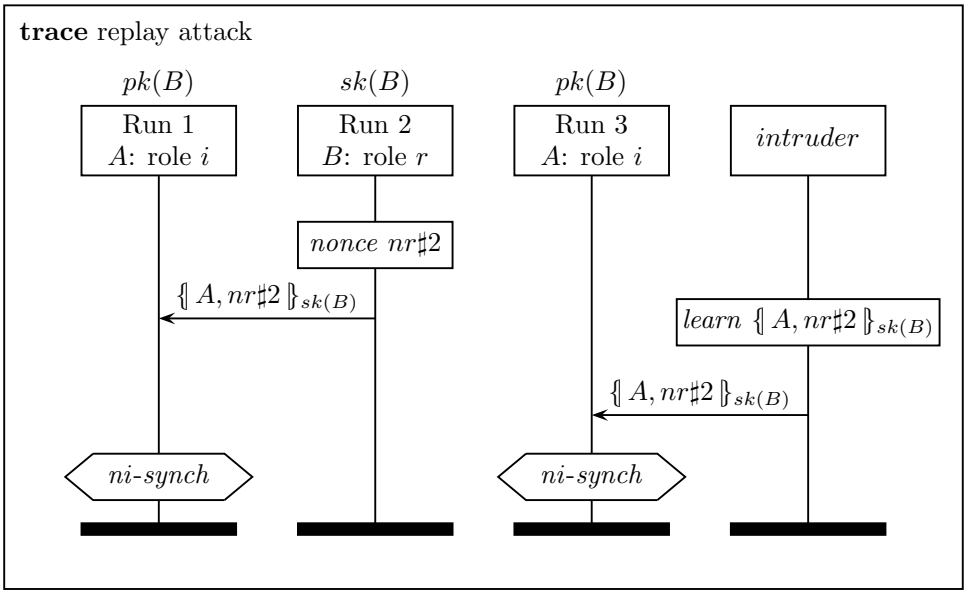

Figure 3.5: A replay attack

Using terminology from Roscoe [161] agreement is a so-called *extensional* security property, which means that it takes into account the effect the protocol achieves. In contrast, the previously defined notions of Aliveness and Synchronisation are *intensional* security properties. Intensional security properties are induced by the form or structure of the protocol, whereas extensional security properties are related to the effect the protocol achieves. For instance, agreement expresses that after successful completion of the protocol the parties agree on the values of all (or some) variables. In order to be able to compare this to synchronisation, we tune the definitions of [122] here, to provide an intensional characterisation of agreement, to arrange for such a comparison. From these definitions it easily follows that injective synchronisation is stronger than injective agreement over all variables, and thus forms a new top element in the authentication hierarchy of Lowe. Using these insights, we are able to show the difference between the several forms of authentication by means of some simple examples.

The starting point for providing an intensional characterisation of agreement is the following definition of *injective agreement* by Lowe [122].

> Initiator i is in agreement with responder r, whenever i as initiator completes a run of the protocol with r, then r as responder has been running the protocol with i. Moreover, i and r agree on all data variables, and each run of i corresponds to a unique run of r.

Although this definition is conceptually clear, it is still informal. Therefore, we will analyse this definition and translate the given concepts into our framework.

The main issue to be clarified is the notion of a *data variable*, which we did not use to define synchronisation. Since the values of the variables are determined by the contents of the messages sent and received, we can reformulate the correspondence between the variables as a requirement on the contents of the sent and received messages. In a two-party protocol, it is clearly the case that if two parties agree on the values of *all* variables, then they agree on the contents of all messages exchanged, and vice versa.

Agreement in a multi-party protocol means that only the initiator and the responder agree on their shared variables. There is no such requirement for the variables maintained by the other roles in the protocol. In order to be able to provide an intensional definition of agreement, we will have to extend the agreement relation to all parties involved in the protocol. Therefore, we will require that upon completion of the protocol *all* parties agree on *all* variables. This is somewhat stronger than the extensional definition provided by Lowe, but for many multi-party protocols this seems to be a natural extension. Summarising, we see that the agreement requirement translates to the demand that corresponding sends and receives have the same contents.

Given this interpretation of agreement, it is easy to see the correspondence with synchronisation. Like agreement, synchronisation requires correspondence on the contents of all messages, but in addition it requires that a message is sent before it can be received. The definition of Lowe does not bother about this send/read order. Thus, we arrive at the following definition of non-injective agreement, which is adapted from Definition 3.8 on page 42 by removing the requirement that send events occur before their corresponding read event.

Definition 3.13 (*NI-AGREE*). *Given a protocol P with a claim protocol event γ, non-injective agreement holds, notation NI-AGREE(P,γ), if*

$$
\begin{array}{l}
\forall t \in \mathit{traces}(P)\ \exists \Gamma \in \mathit{Cast}(P,t) \\
\quad \forall c \in t : c = ((\theta,\rho,\sigma),\gamma)\mathit{rng}(\rho) \subseteq \mathit{Agent}_T \\
\quad\quad \forall \varsigma,\varrho\colon \varsigma \rightsquigarrow \varrho \preccurlyeq_P \gamma\ \exists s,r\colon s <_t c \wedge r <_t c \\
\quad\quad\quad \mathit{roleevent}(s) = \varsigma \ \wedge\ \mathit{runidof}(s) = \Gamma(c)(\mathit{role}(\varsigma)) \wedge \\
\quad\quad\quad \mathit{roleevent}(r) = \varrho \ \wedge\ \mathit{runidof}(r) = \Gamma(c)(\mathit{role}(\varrho)) \wedge \\
\quad\quad\quad \mathit{cont}(s) = \mathit{cont}(r)
\end{array}
$$

The agreement predicate expresses that for all instantiated claims in any trace of a given security protocol, there exist runs for the other roles in the protocol, such that all communication events causally preceding the claim must have occurred before the claim.

Injective agreement is defined in the same way as injective synchronisation is obtained from non-injective synchronisation.

Definition 3.14 (*I-AGREE*). *Given a protocol P with a claim protocol event γ,*

injective agreement holds, notation I-AGREE(P,γ), if

$$\begin{array}{l}\forall t \in \mathit{traces}(P)\ \exists \Gamma \in \mathit{Cast}(P,t),\ \mathit{injective}\\ \quad \forall c \in t,\ c = ((\theta,\rho,\sigma),\gamma),\ \mathit{rng}(\rho) \subseteq \mathit{Agent}_T,\\ \quad\quad \forall \varsigma, \varrho \colon \varsigma \rightsquigarrow \varrho \preccurlyeq_P \gamma\ \exists s, r \colon s <_t c \wedge r <_t c\\ \quad\quad\quad \mathit{roleevent}(s) = \varsigma\ \wedge\ \mathit{runidof}(s) = \Gamma(c)(\mathit{role}(\varsigma)) \wedge\\ \quad\quad\quad \mathit{roleevent}(r) = \varrho\ \wedge\ \mathit{runidof}(r) = \Gamma(c)(\mathit{role}(\varrho)) \wedge\\ \quad\quad\quad \mathit{cont}(s) = \mathit{cont}(r)\end{array}$$

It expresses that for any trace and for any run of any role in the protocol there exist unique runs for the other roles of the protocol such that for all claims occurring in the trace all communications preceding the claim must have occurred correctly within these runs.

The definition of *I-AGREE* does not involve *all* communications, but only the set of events that causally precede a claim. However, it turns out that the way in which agreement is made precise in terms of CSP, as can be checked by compiling Casper-code into CSP, it also takes only preceding communications into account. For this, running-commit signals (see [164]) are introduced in the protocol. For each role, a running signal is added to the last communication preceding the agreement claim. In the role that includes the claim, a commit signal is added to the last communication. Injective agreement over all roles requires that the running signals of each role precede the commit signal. This corresponds to the order requirements of *I-AGREE*.

3.4 Authentication hierarchy

In [122], Lowe defined a number of authentication properties and positioned them in a hierarchy. In this section we study the relation between these properties and our notion of (injective) synchronisation. Since time is not considered in our model, we will restrict our attention to authentication properties not involving time.

The definitions of the four security properties above clearly reveal their relative strengths in excluding attacks. Every injective protocol is also non-injective and if a protocol satisfies synchronisation then it satisfies agreement too. Figure 3.6 on the following page shows this hierarchy. An arrow from property X to property Y means that every protocol satisfying X also satisfies Y. Phrased differently, the class of protocols satisfying X is included in the class satisfying Y.

The correctness of the hierarchy is captured by the following theorem.

Theorem 3.15. *The security properties I-SYNCH, NI-SYNCH, I-AGREE, and NI-AGREE satisfy the inclusion relation as depicted in Figure 3.6 on the next page.*

Proof. Straightforward from the definitions. □

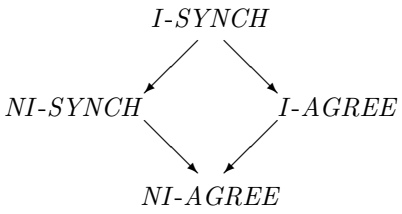

Figure 3.6: Hierarchy of security properties.

The question of whether the inclusions in Figure 3.6 are strict is harder to answer. In part, this is due to the generic approach of our model. Since our approach is parameterised over the intruder model, we cannot determine for a given protocol to which class it belongs. Therefore, strictness of the inclusions can only be answered relative to a given intruder model. Consequently, the following reasoning will be at a conceptual level only.

If we take, e.g., a model where all agents simply follow their roles and the intruder has no capabilities at all, then the diamond in Figure 3.6 collapses into a single class. The same holds if the intruder can only eavesdrop on the communications. However, in the Dolev-Yao model, all inclusions are strict, as we will see below.

The case of injectivity vs. non-injectivity has been studied extensively before. The MSC on in Figure 3.7 shows a protocol that satisfies *NI-SYNCH* and *NI-AGREE*, but neither *I-SYNCH*, nor *I-AGREE*.

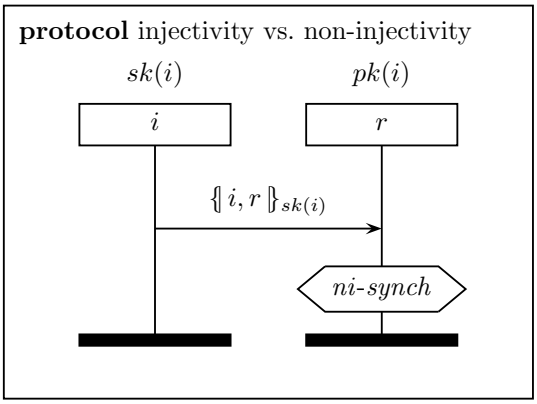

Figure 3.7: A protocol that is not injective

The intruder will only be able to construct message $\{\!|\, i, r \,|\!\}_{sk(i)}$ after having eavesdropped this message from a previous run. Therefore every read event of this message is preceded by a corresponding send event, so the protocol is both *NI-SYNCH* and *NI-AGREE*. However, once the intruder has learnt this message, he can replay it as often as desired, so the protocol is not injective.

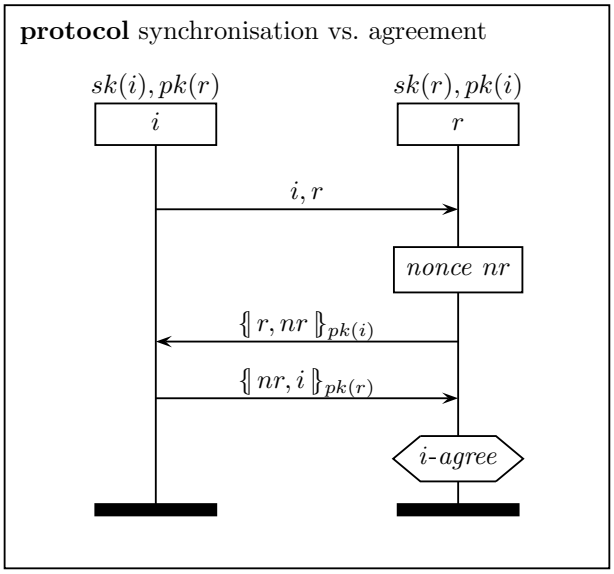

Figure 3.8: A protocol that does not synchronise

A distinguishing example between synchronisation and agreement is depicted in Figure 3.8. As confirmed by the Casper/FDR tool set (developed by Lowe), this protocol satisfies unilateral authentication in the sense of agreement (both injective and non-injective). However, the protocol does not satisfy synchronisation (both variants). This is the case, because the intruder can send message i, r long before i actually initiates the protocol, making r to believe that i has requested the start of a session before he actually did.

The two examples show that the inclusions of the diamond in Figure 3.6 on the facing page are strict if the intruder has the capabilities to eavesdrop, deflect and inject messages. Both examples also imply that there are no arrows between *NI-SYNCH* and *I-AGREE*.

Synchronisation vs. agreement As stated before, the difference between synchronisation and agreement is rather subtle and, indeed, most authentication protocols in practice satisfy both properties. The distinction is that synchronisation requires that corresponding send and receive messages have to be executed in the expected order, while for agreement a message may be received before it is sent. This can, for instance, be caused by a message injection of the intruder. An attack in which the intruder injects a message before its actual creation is called a *preplay* attack. Whether such a protocol weakness can be exploited by the attacker depends on the intention of the protocol. Below we will sketch three examples of possible weaknesses.

Example 3.16 (Predictable nonce). *In the first example we consider the notion*

of predictable nonces. *There may be several reasons for such predictability, such as a bad pseudo-random generator, or the fact that the nonce is implemented as a counter. Consider, for instance, the protocol in Figure 3.9. The purpose of this protocol is unilateral authentication of responder r towards initiator i. This is established by using nonce ni as a challenge. However, if the value of this nonce is predictable by the intruder, the protocol has a major shortcoming. This is shown in the trace in Figure 3.10 on the facing page, which displays that the run of r has finished even before the run of i started. The consequences of this preplay attack are similar to many well-known replay attacks. The protocol satisfies (injective and non-injective) agreement, but does not satisfy synchronisation (both variants).*

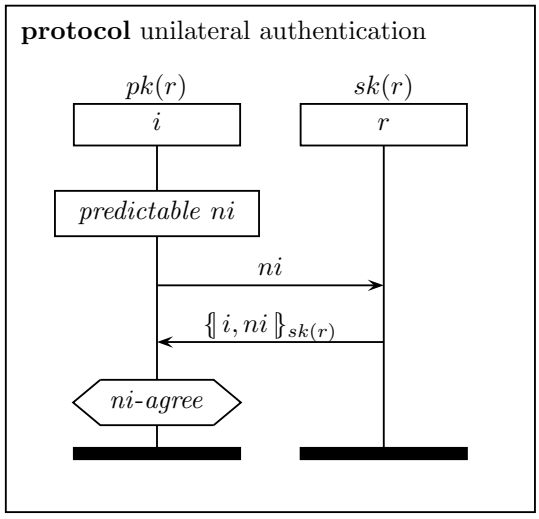

Figure 3.9: A protocol with a predictable nonce

This type of preplay attacks is also called *suppress-replay* attacks [91, 142]. As pointed out by Chen and Mitchell [142], practical protocols such as the S/KEY user authentication scheme suffer from this kind of attack because they use predictable challenges. Roscoe [161] found a similar problem for the Needham-Schroeder Secret Key protocol in the case that the initiator's nonce is predictable.

Example 3.17 (Responder controls nonce). *The second example concerns the protocol in Figure 3.11 on page 52, in which we assume that an agent keeps a state which is shared by all its instances of the protocol. The purpose of this protocol is again unilateral authentication, but now the responder is in control of the nonce. After receiving a request from the initiator, the responder sends his nonce to the initiator. The initiator keeps a set Z in which he stores all nonces from previous instantiations of the protocol. This is to prevent replay attacks and, thus, to ensure injectivity. If the nonce is accepted as fresh, the initiator challenges the responder to prove his identity, which the responder does by replying the signed nonce. This may seem a reasonable authentication protocol, and indeed it satisfies injective agreement.*

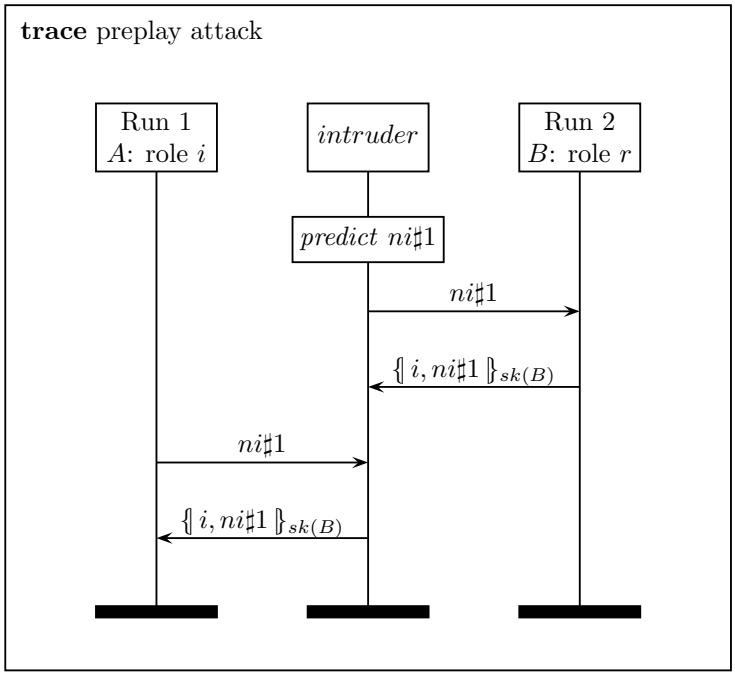

Figure 3.10: An attack

However, the preplay attack shown in Figure 3.12 on page 53 indicates a weakness of the protocol. An initiator can successfully execute his side of the protocol, while the responder was not even alive when the initiator started the protocol. This example shows that even a complex message interaction can be preplayed. Since the protocol does not satisfy synchronisation, this weakness can be detected by verifying synchronisation.

Two remarks apply to this example. The first remark is that testing whether the nonce is already in Z and the extension of Z with the nonce should be implemented as an atomic action (a *test-and-set* action) to achieve the desired result. The second remark is that this protocol still has interesting properties when leaving out the validation of nr's freshness by the initiator (i.e. if we remove the set Z and its operations). The resulting protocol is not injective anymore, since the intruder may replay the responder behaviour of an earlier run of the protocol. However, this reduced protocol still satisfies non-injective agreement. Since the displayed attack remains possible, the protocol does not satisfy synchronisation. Thus, we have a stateless protocol suffering from a preplay attack. It satisfies non-injective agreement but it does not satisfy non-injective synchronisation.

In the previous two examples we have seen how the intruder can preplay a complete protocol session and use it later to fool the initiator into thinking that the responder is still alive. In the third example, we see that it can already be harmful if only a

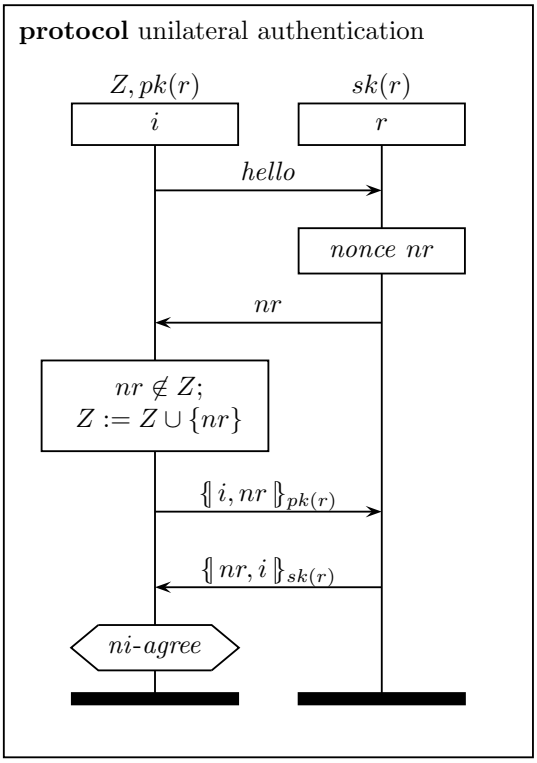

Figure 3.11: A protocol where the responder controls the nonce

single message is preplayed by the intruder.

In Figure 3.13 on page 54, r is an Internet Service Provider, used by i. Assume that i pays r for the time he is connected. When i wants to connect, r retrieves the certificate of i from the trusted server S and uses this to authenticate i. After a successful session, i is billed from the moment the first message was received by r.

This protocol is a slightly modified version of the Needham-Schroeder-Lowe protocol. It can be exploited as follows. An intruder can send the first message pre-emptively, causing r to initiate a session with what it believes is i. If at some later time i decides to initiate a session with r and finishes it successfully, i will receive a bill that is too high. In fact, although, this protocol satisfies agreement for r, the first message is not authenticated at all. In contrast, this protocol does not satisfy synchronisation. The protocol can be easily modified to satisfy *NI-SYNCH* and thus to be resilient against the sketched type of timing attacks.

This type of attack may seem of little relevance, but it depends on the interpretation of the messages whether such unexpected behaviour can cause harm or not. In the rather contrived example above, a typical interpretation of the messages concerning billing time allows the intruder to exploit this unexpected behaviour. Following

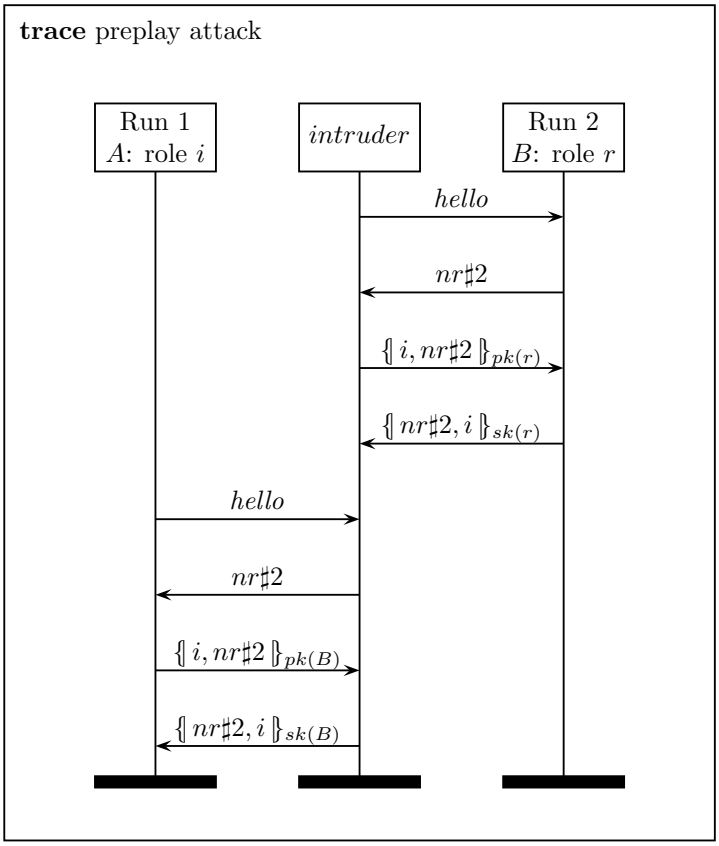

Figure 3.12: An attack

the observations of Roscoe [161], this sort of behaviour, while seemingly innocent was certainly unexpected by the protocol designer. After finding such unexpected behaviour, the designer has two options. First, he may decide that this behaviour is acceptable, but then he has to take the implications of this behaviour into account and extend his mental model of the protocol. He should make sure that this behaviour does not interfere with any other analysis which is based on the intended order of the protocol events. Still according to Roscoe, the second option is to strengthen the protocol as to make it compatible with the mental model again. As pointed out by Roscoe, the TMN protocol, and a seemingly correct strengthening thereof, suffer from the same weakness as the protocol in Figure 3.13 on the following page.

We conclude by stating that failure of a protocol to respect synchronisation does not always indicate an exploitable weakness of the protocol. However, such unexpected behaviour should always receive extra attention and should at least lead to adjusting the mental model of the protocol.

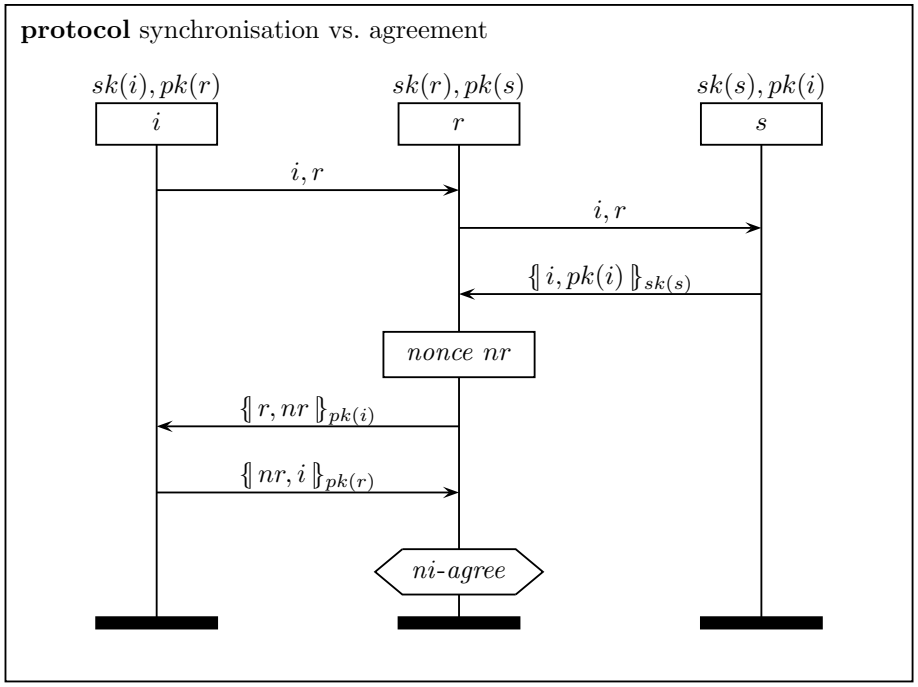

Figure 3.13: A second protocol that does not satisfy synchronisation

3.5 Verifying injective synchronisation

Several tools exist to verify whether a protocol satisfies some form of agreement, e.g. [121, 7, 173]. Because synchronisation is very similar to agreement, we expect that it will be feasible to adapt most of the verification tools to be able to handle at least non-injective synchronisation.

In refinement or forward model-checking approaches, agreement is commonly verified by inserting *running* and *commit* or similar signals in the protocol [121, 7]. When somebody commits to some values, the other party needs to have emitted a running signal. The commit signal corresponds to the claim in our framework, whereas the running signal denotes the last communication of the other role that causally precedes the claim. These signals are introduced to ease verification: instead of having to inspect the trace leading up to the claim, only the set of emitted signals needs to be inspected. In our framework, agreement is a property of the trace prefix ending in a claim. By introducing running and commit signals, agreement can be verified by inspecting the set of signals. In much the same way, it is possible to verify synchronisation by introducing such signals for each communication that precedes the claim.

In order to verify injective agreement, many tools rely on a counting argument: if a trace prefix contains n commit signals, there should be at least n preceding

running signals in the trace prefix. If enough commit signals are considered in this way, this should ensure injectivity. The main drawback of this method is that the verification complexity is exponential in the length of the traces, and thus in the number of running claims considered. Another approach to verifying injectivity uses detailed knowledge of the data model: if the agreement includes data items that are guaranteed to be unique for each instance of the claim, injectivity can be derived. However, not all injective protocols include such unique data items. In some cases it can also be non-trivial to establish the required uniqueness property, as we show in the next section.

This section will focus on the verification of injectivity for protocols that satisfy non-injective synchronisation. We propose and study a property, the *LOOP* property that can be syntactically verified. We prove a theorem that shows that *LOOP* is sufficient to guarantee injectivity. Our result is generic in the sense that it holds for a wide range of security protocol models, and does not depend on the details of message content or nonce freshness.

The remainder of this section will proceed as follows. First we elaborate on the difficulties posed by injectivity, and informally describe our theorem. Then, we define a class of intruder models for which our theorem holds. Then, in Section 3.5.2 we propose and study the *LOOP* property, and prove the main theorem.

3.5.1 Injectivity of synchronisation

As discussed in Section 3.3.2 protocols that satisfy *NI-SYNCH* can still be vulnerable to so-called replay attacks. Looking back at example 3.10 on page 43 in particular, we saw that the unilateral authentication protocol from Figure 3.4 clearly does not satisfy injectivity, as is shown by the replay attack in Figure 3.5. A simple fix would be to have the initiator determine the value of the nonce, as in Figure 3.14.

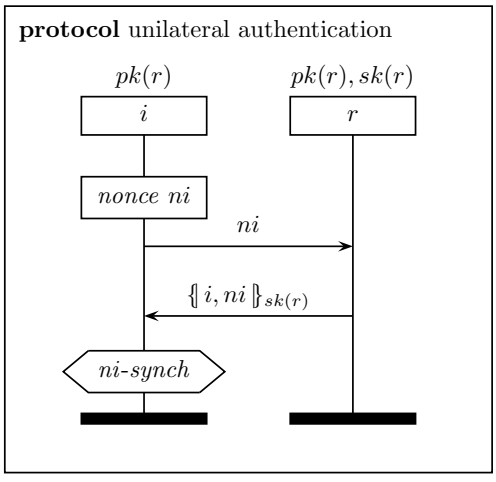

Figure 3.14: A protocol that is not vulnerable to replay attacks

The introduction of a causal chain of messages from the initiator to the responder and back to the initiator seems to do the trick. We will call such a chain a *loop*. The presence of such a loop plays a key role in the discussion on injectivity below.

It is folklore that a nonce handshake is sufficient to ensure injectivity. Here we identify a more abstract property, viz. the occurrence of a loop, which is independent of the data model, and thus applicable to a wide range of security protocol models. To give an indication of the limitations of the data based approach, consider a protocol where a nonce n is created, and some function is applied to it. The result $f(n)$ is sent to the responder, who applies another function and replies with $g(f(n))$. Now, to check whether such a protocol can be injective based on the freshness of n in a data-based model, we need to know some details of f and g. If for example $f(x) = x \bmod 2$, the protocol will not be injective. Therefore, we propose to only reason about loops, which does not require any information about the contents of messages.

We show that in our model (but also others, as we show in more detail in [70]), and given that the intruder is at least capable of duplicating messages, the *LOOP* property guarantees that a synchronising protocol is also injective. For this result to hold, we require a specific property to hold for the intruder model. We can informally characterize this by saying that the intruder must at least be able to duplicate messages. To be more precise, the protocol model (including the intruder model) must satisfy closure of the set of execution traces under swapping of events. This class contains, e.g., the model presented in the previous chapter with a duplicating (or stronger) intruder. Since the *LOOP* property can easily be verified by means of static analysis of the security protocol description, we provide, in fact, a practical syntactic criterion for verifying injectivity.

We extend the use of the $\cdot$ operator, and introduce the notation $t = u1 \cdot u2$ to denote that t is a concatenation of run event sequences $u1$ and $u2$.

For the remainder of this chapter, we assume that traces contain only role events, and intruder events are left implicit.

Definition 3.18 (Swap). *A security protocol semantics satisfies the SWAP property if the following two conditions hold:*

(i) The trace set $traces(P)$ is closed with respect to non-read swaps, i.e., for all run events e' such that $roleevent(e')$ is not a read event, it holds that

$$t \cdot e \cdot e' \cdot t' \in traces(P) \ \wedge \ runidof(e) \neq runidof(e') \Rightarrow \ t \cdot e' \cdot e \cdot t' \in traces(P)$$

for all run events e and traces t, t'.

(ii) The trace set $traces(P)$ is closed with respect to read swaps, i.e., for s, r, e with $roleevent(s)$ is a send event, $roleevent(r)$ is a read event, we have that

$$t \cdot s \cdot t' \cdot e \cdot r \cdot t'' \in traces(P) \ \wedge \ runidof(e) \neq runidof(e') \wedge \ cont(s) = cont(r) \ \Rightarrow \ t \cdot s \cdot t' \cdot r \cdot e \cdot t'' \in traces(P)$$

for all traces t, t', t''.

These properties state that we can shift a non-read event to the left as long as it does not cross any other events of the same role instance. For the read event we have an additional constraint: we can only shift it to the left if there remains an earlier send of the same message.

Lemma 3.19 (Duplicating intruder implies SWAP). *If the intruder is at least able to duplicate messages, the SWAP property holds for our model.*

Proof. The first condition, swapping non-reads, is related to the fact that independent runs share no memory, and trivially holds based on the semantics. The second condition, swapping reads, is met based on the duplication property. The precondition of the read rule of the semantics (i. e. that the appropriate message is in the read buffer) can be satisfied at any time by a duplicating intruder, once the message has been sent once. □

For the remainder of this section we assume the protocol P contains a claim γ. We introduce a predicate χ and a set χ' for protocols that satisfy non-injective agreement for this claim γ. Given a trace t, a claim event c, and a cast Γ that maps the roles to runs, we express the auxiliary predicate χ on the domain $\mathit{traces}(P) \times \mathit{Cast}(P,t) \times \mathit{RunEvent}$ by

$$\begin{aligned}\chi(t,\Gamma,c) \iff\ & \mathit{roleevent}(c) = \gamma \wedge \\ & \forall \varsigma, \varrho : \varsigma \rightsquigarrow \varrho \wedge \varrho \preccurlyeq_P \gamma\ \exists s, r : s <_t r <_t c \\ & \quad \mathit{roleevent}(s) = \varsigma \ \wedge\ \mathit{runidof}(s) = \Gamma(c)(\mathit{role}(\varsigma)) \wedge \\ & \quad \mathit{roleevent}(r) = \varrho \ \wedge\ \mathit{runidof}(r) = \Gamma(c)(\mathit{role}(\varrho)) \wedge \\ & \quad \mathit{cont}(s) = \mathit{cont}(r)\end{aligned}$$

The first conjunct of this predicate expresses the fact that the run executing the claim role is determined by the parameter c. The second conjunct expresses that in the trace t, the claim c is valid with respect to the specific cast Γ, i. e. that the partners have executed all communications as expected. In the formula this is expressed by the fact that send and read events are executed by the expected runs, viz. $\Gamma(c)(\mathit{role}(\mathit{roleevent}(s)))$ and $\Gamma(c)(\mathit{role}(\mathit{roleevent}(r)))$, respectively, with identical contents, and in the right order.

Given a valid synchronisation claim c in a trace t, there exists a role instantiation function Γ such that $\chi(t,\Gamma,c)$ holds. The predicate χ tells us that certain events exist in the trace. Because we want to reason about these events in the following, we decide to make this set of events explicit. We define the set of events $\chi'(t,\Gamma,c)$ by

$$\chi'(t,\Gamma,c) = \{\ e \in t \mid \mathit{runidof}(e) = \Gamma(c)(\mathit{role}(e)) \ \wedge\ \mathit{roleevent}(e) \preccurlyeq_P \mathit{roleevent}(c)\ \}$$

Assuming that χ holds, its set of events χ' has two interesting properties. If there is a read in this set, there is also a matching send in the set. Furthermore, given an event of a role in the set, all preceding events of the same role are also in the set.

To prove our main result, the *SWAP* property from Definition 3.18 on page 56 suffices. However, to ease the explanation of the proof, we introduce two additional lemmas. These lemmas are implied by the model and the two swap conditions.

The first lemma generalises the swapping of two events to the swapping of a set of events. The lemma does not hold for any set of events: we now use results obtained for a set of events defined by χ, that are involved in a synchronisation claim. Based on the two swap properties, we can shift these events (in their original order) to the beginning of the trace. This trace transformation function $\mathit{shift}\colon \mathcal{P}(TE) \times TE^* \to TE^*$ is defined by

$$\mathit{shift}(E, t) = \begin{cases} t & \text{if } t \cap E = \emptyset \\ e \cdot \mathit{shift}(E, u1 \cdot u2) & \text{if } t = u1 \cdot e \cdot u2 \ \wedge \ u1 \cap E = \emptyset \ \wedge \ e \in E \end{cases}$$

Here, the intersection of a trace and a set yields the collection of elements of the set occurring in the trace. This function effectively reorders a trace. The next lemma formulates conditions assuring that the reordering of a trace in $\mathit{traces}(P)$ is in $\mathit{traces}(P)$ as well.

Lemma 3.20. *Given a protocol P and a trace $t \in \mathit{traces}(P)$, claim event c and role instantiation function Γ:*

$$\chi(t, \Gamma, c) \ \wedge \ t' = \mathit{shift}(\chi'(t, \Gamma, c), t) \ \Rightarrow t' \in \mathit{traces}(P) \ \wedge \ \chi(t', \Gamma, c).$$

Proof. Induction on the size of the finite set $\chi'(t, \Gamma, c)$, because $\chi(t, \Gamma, c)$ implies that the read events can be swapped. Recall that, by convention, each event occurs at most once in a trace. □

The lemma directly generalises to more claim instances (of the same claim). Thus, instead of a single claim run, we can consider sets of claim runs.

Lemma 3.21. *Given a trace t, a set of claim events $C \subseteq t$ and cast $\Gamma \in \mathit{Cast}(P, t)$:*

$$(\forall c \in C\colon \chi(t, \Gamma, c)) \ \wedge \ t' = \mathit{shift}(\bigcup_{c \in C} \chi'(t, \Gamma, c), t) \ \Rightarrow \\ t' \in \mathit{traces}(P) \ \wedge \ (\forall c \in C\colon \chi(t', \Gamma, c))$$

Proof. Similar to the proof of Lemma 3.20. □

If we apply the *shift* function to a trace of the system, and the conditions of the lemma are met, we get a reordered trace that is also in $\mathit{traces}(P)$. The new trace consists of two segments: in the first segment there are only the preceding events of the claim events in C, and all other events are in the second segment.

Intuitively, these lemmas express that the events involved in a valid synchronisation claim are independent of the other events in the trace. A valid synchronisation can occur at any point in the trace, because it does not require the involvement of other runs, or of the intruder. However, other events in the trace might depend on events

involved in the synchronisation. Although we cannot shift the synchronising events to the right, we can shift them to the left, which ensures that any dependencies will not be broken.

We use Lemma 3.20 on the facing page and Lemma 3.21 on the preceding page in the injectivity proof in the next section.

3.5.2 The *LOOP* property

We define a property of protocols, which we call the *LOOP* property. For protocols with only two roles, it resembles a ping-pong property. First the claim role executes an event, then the other role, and then the claim role again. For example, the *LOOP* property does not hold for the protocol in Figure 3.5 on page 45, but it does hold for the protocols in Figures 3.14 on page 55 and 2.1 on page 12.

We generalise this for multi-party protocols with any number of roles. We require that the partner roles have an event that must occur after the start of the claim run, but before the claim event itself.

Definition 3.22 (*LOOP*). *A security protocol P has the LOOP property with respect to a claim γ if*

$$\begin{aligned}&\forall \varepsilon \preccurlyeq_P \gamma, \mathit{role}(\varepsilon) \neq \mathit{role}(\gamma)\\ &\qquad \exists \varepsilon', \varepsilon'' \colon \varepsilon' \preccurlyeq_P \varepsilon'' \preccurlyeq_P \gamma \ \wedge\ \mathit{role}(\varepsilon') = \mathit{role}(\gamma) \ \wedge\ \mathit{role}(\varepsilon'') = \mathit{role}(\varepsilon)\end{aligned} \tag{3.3}$$

The property tells us that for each role that has an event ε that precedes the claim γ, there exists a loop from the claim role to the role and back. This structure is identified in the formula by $\varepsilon' \preccurlyeq_P \varepsilon'' \preccurlyeq_P \gamma$.

Lemma 3.23. *Given a security protocol P with a claim γ: If all roles $R \neq \mathit{role}(\gamma)$ that have events preceding γ, start with a read event, then we have that $LOOP(P, \gamma)$.*

Proof. The proof of this lemma follows from the definition of the protocol order $\preccurlyeq_P$. Let P be a protocol with a claim γ. Let $R \neq \mathit{role}(\gamma)$ be a role with an event ε that precedes the claim. Based on the precondition of the lemma, it starts with a read, and thus there must be a preceding event with the same label on a different role R'. Suppose $R' = \mathit{role}(\gamma)$: in this case we have established a loop. On the other hand, if we have $R' \neq \mathit{role}(\gamma)$, we again have that this role must have a preceding event on another role. As the set of role events is finite, we ultimately end up at an event of the claiming role. Thus we can conclude that $LOOP(P, \gamma)$ holds. □

In practice, this lemma tells us that the *LOOP* property always holds for the initiating role of a protocol. Thus, we only have to check whether the *LOOP* property holds for responder roles.

Now we can state a theorem, which provides a syntactic condition for the injectivity of a protocol that synchronises.

Theorem 3.24 (LOOP). *Let P be a protocol with claim eent γ. Then we have that*

$$NI\text{-}SYNCH(P,\gamma) \wedge LOOP(P,\gamma) \Rightarrow I\text{-}SYNCH(P,\gamma)$$

Proof. By contradiction. Assume that the implication does not hold. Thus we have

$$NI\text{-}SYNCH(P,\gamma) \wedge LOOP(P,\gamma) \wedge \neg I\text{-}SYNCH(P,\gamma). \tag{3.4}$$

The remainder of the proof is done in two steps. The first step of the proof establishes a trace t of the protocol, in which there are two runs that synchronise with the same run. In the second step we use the shifting lemmas to transform t into another trace of the protocol. For this new trace, we will show that *NI-SYNCH* cannot hold, which contradicts the assumptions.

From now on, we will omit the type information for t and Γ in the quantifiers and assume that $t \in traces(P)$.

Given that the protocol synchronises, but is not injective, we derive from definition 3.8 on page 42 and 3.11 and formula (3.4) that

$$\begin{aligned}\forall t\ \exists \Gamma\ \forall c \in t : roleevent(c) = \gamma \Rightarrow \chi(t,\Gamma,c) \wedge \\ \neg \forall t\ \exists \Gamma \text{ injective } \forall c \in t : roleevent(c) = \gamma \Rightarrow \chi(t,\Gamma,c)\end{aligned} \tag{3.5}$$

We push the negation on the right through the quantifiers, yielding

$$\begin{aligned}\forall t\ \exists \Gamma\ \forall c \in t : roleevent(c) = \gamma \Rightarrow \chi(t,\Gamma,c) \wedge \\ \exists t\ \forall \Gamma \neg(\Gamma \text{ injective} \wedge\ \forall c \in t : roleevent(c) = \gamma \Rightarrow \chi(t,\Gamma,c)).\end{aligned} \tag{3.6}$$

Based on the existential quantifiers in (3.6), we choose a trace t and instantiation function Γ such that

$$\begin{aligned}\forall c \in t : roleevent(c) = \gamma \Rightarrow \chi(t,\Gamma,c) \wedge \\ \neg(\Gamma \text{ injective} \wedge\ \forall c \in t : roleevent(c) = \gamma \Rightarrow \chi(t,\Gamma,c)).\end{aligned} \tag{3.7}$$

Note that in (3.7) the left conjunct also occurs as a sub-formula in the right conjunct. Rewriting yields

$$\forall c \in t : roleevent(c) = \gamma \Rightarrow \chi(t,\Gamma,c)\ \wedge\ \neg(\Gamma \text{ injective}). \tag{3.8}$$

Making the non-injectivity for the function Γ explicit as explained in Definition 3.5 on page 38, there must exist two claim events, for which χ holds:

$$\begin{aligned}\exists c1, c2, R1, R2 : \chi(t,\Gamma,c1)\ \wedge\ \chi(t,\Gamma,c2) \\ \wedge\ \Gamma(c1)(R1) = \Gamma(c2)(R2)\ \wedge\ (c1 \neq c2\ \vee\ R1 \neq R2)\end{aligned} \tag{3.9}$$

From the predicate χ and formula (3.9), we have that the run $\Gamma(c1)(R1)$ must be executing the role $R1$. Because $\Gamma(c1)(R1) = \Gamma(c2)(R2)$ it is also executing role $R2$.

Runs only execute a single role, and thus we derive that $R1 = R2$. The fourth conjunct now reduces to $c1 \neq c2$.

Put $R = R1 = R2$. We choose two claim events $c1, c2$ such that Formula (3.9) holds for R. Now there exists a run identifier θ such that

$$\Gamma(c1)(R) = \Gamma(c2)(R) = \theta$$

From the definition of χ, we obtain that if R would be equal to $role(\gamma)$, we would have $\theta = c1$ and $\theta = c2$, implying $c1 = c2$ and contradicting Equation (3.9). Thus, we have $R \neq role(\gamma)$.

We have now established that the trace t contains events from at least three role instances. Two of these, $[c1]_\pi$ and $[c2]_\pi$, are executing the claim role, while the third, θ is executing a different role R. Furthermore, we have that the claims $c1$ and $c2$ synchronise with θ.

This completes the first step of the proof. We will now proceed by transforming t into a trace for which *NI-SYNCH* cannot hold, for the second part of the proof.

Because we have $\chi(t, \Gamma, c1)$ and $\chi(t, \Gamma, c2)$, on the basis of Lemma 3.21 on page 58 we can apply *shift* using $c1$ and $c2$ to get a trace $t' \in traces(P)$

$$t' = shift(\chi'(t, \Gamma, c1) \cup \chi'(t, \Gamma, c2), t)$$

In the trace t' we now have two distinct segments. All events involved with the synchronisation of $c1$ and $c2$ are now in the initial segment of t'. This includes the events of θ that precede the claim. The second segment of t' contains all other events that are not involved in the preceding events of $c1$ and $c2$.

We will now reorder the initial segment of t'. To this end, we apply the *shift* function a second time, now only for $c1$. This will also yield a trace of the protocol, because the conditions of Lemma 3.21 on page 58 hold for t', as the application of *shift* to t maintained the order of the events in the shifted set, which implies that $\chi(t', \Gamma, c1)$ holds. Thus, we also know that the trace t'' is an element of $traces(P)$, where

$$t'' = shift(\chi'(t', \Gamma, c1), t')$$

Because the *shift* function maintains the order of the involved events, we have that $t'' = u1 \cdot u2 \cdot u3$, where

$$\begin{aligned} set(u1) =& \chi'(t', \Gamma, c1) \\ set(u2) =& \chi'(t, \Gamma, c2) \setminus \chi'(t', \Gamma, c1) \end{aligned}$$

All events that are not involved with the synchronisation claims $c1$ and $c2$, are in $u3$.

Observe that $u1$ includes all events of θ that are involved with the claim of the run $c1$. As all events are unique, these are not part of $u2$. From the construction of the involved events set, we know that all involved events of role R are also in θ, because all other role instances are executing other roles (as indicated by Γ). This implies that there are no events of role R in the $u2$ segment at all: these are all in $u1$.

Now we have arrived at a contradiction. t'' is in the set $traces(P)$. The loop property combined with *NI-SYNCH* requires that for each role, there is an event after the first event of the claim role that occurs before the claim. For the run $c2$ all events are in $u2$ (including the start and the claim), but in this segment there is no event of role R. Thus, there can be no Γ for t'' such that $\chi(t'', \Gamma, c2)$ holds. This implies that *NI-SYNCH* does not hold for the protocol, which contradicts the assumptions. □

Thus, we have established that *LOOP* is a sufficient condition to guarantee injectivity for protocols that satisfy *NI-SYNCH*.

We conclude this chapter by illustrating how the definitions of some of the security properties described here can be used for manually proving a protocol correct.

3.6 Proving security properties of the NS/NSL protocol

In this section we will take a closer look at the Needham-Schroeder protocol from Figure 2.1 on page 12 and illustrate our definitions. The protocol goal is to ensure mutual authentication and as a side effect secrecy of the involved nonces. Starting point of the protocol is a public key infrastructure. This is depicted by the initial knowledge above each of the roles in the protocol. The initiator starts the protocol by sending an encrypted initialisation request to the responder. The nonce is used to prevent replay attacks. Only the responder is able to unpack this message and replies by sending the initiator's nonce together with his own fresh nonce. Then the initiator proves his identity by replying the responder's nonce.

The man-in-the-middle attack in Figure 3.3 on page 41 only requires two runs. One of trusted agent A performing the initiator role in a session with untrusted agent E and one of trusted agent B performing the responder role in a session with agent A. The intruder impersonates both E and A and in this way uses A as an oracle to unpack message from B. At the end he has fooled B into thinking that he is talking to A, while he is talking to the intruder.

Knowing this attack, it is straightforward to reconstruct it formally with our semantics. Our experience shows that when trying to prove a flawed protocol correct, the way in which the proof fails often indicates the attack. Rather than showing the details here, we will prove correctness of the fixed Needham-Schroeder protocol, which is called the Needham-Schroeder-Lowe protocol. The protocol is hardened by extending message 2 with the responder name. It is specified as follows.

$$
\begin{aligned}
nsl(i) = (\{&i, r, ni, pk(r), pk(i), sk(i)\},\\
&send_1(i, r, \{\!|\, i, ni \,|\!\}_{pk(r)})\cdot\\
&read_2(r, i, \{\!|\, ni, V, r \,|\!\}_{pk(i)})\cdot\\
&send_3(i, r, \{\!|\, V \,|\!\}_{pk(r)}))\cdot\\
&claim_4(i, secret, ni)\cdot\\
&claim_5(i, secret, V)\cdot\\
&claim_6(i, nisynch)\ )
\end{aligned}
\qquad
\begin{aligned}
nsl(r) = (\{&i, r, nr, pk(i), pk(r), sk(r)\},\\
&read_1(i, r, \{\!|\, i, W \,|\!\}_{pk(r)})\cdot\\
&send_2(r, i, \{\!|\, W, nr, r \,|\!\}_{pk(i)})\cdot\\
&read_3(i, r, \{\!|\, nr \,|\!\}_{pk(r)}))\cdot\\
&claim_7(r, secret, nr)\cdot\\
&claim_8(r, secret, W)\cdot\\
&claim_9(r, nisynch)\ )
\end{aligned}
$$

We assume that there are no trusted roles. For this protocol, the initial intruder knowledge (cf. Definition 2.44) is given by

$$M_0 = \mathit{IntruderConst} \cup \bigcup_{a \in Agent} \{a, pk(a)\} \cup \bigcup_{e \in Agent_U} \{sk(e)\}$$

First we introduce some notation and present results which support verification. We define $msgs(P)$ as the set of all role messages sent in the protocol. The first lemma helps to infer that secret information which is never transmitted, remains secret forever.

Lemma 3.25. *Let P be a protocol, inst an instantiation and t a term that is not a tuple or an encryption. If t is not a subterm of any message that is ever sent, and $\langle inst \rangle(t)$ is not a subterm of the initial intruder knowledge, then $\langle inst \rangle(t)$ will never be known by the intruder. Formally:*

$$\forall_{t' \in msgs(P)} t \not\sqsubseteq t' \ \wedge \ \forall_{m : M_0 \vdash m} \langle inst \rangle(t) \not\sqsubseteq m \ \Rightarrow \ \forall_{\alpha \in Trace(P), 0 \leq j \leq |\alpha|} M_j^\alpha \not\vdash \langle inst \rangle(t)$$

The correctness of this lemma follows from the OS-rules.

The next lemma expresses that roles are executed from the beginning to the end. The predicate $e \prec_R e'$ means that event e precedes event e' in the specification of role R.

Lemma 3.26. *Let α be a trace of a protocol, let (θ, ρ, σ) be an instantiation and e', e events, such that $e' \prec_R e$ for some role R. If for some i $(0 \leq i < |\alpha|)$ $\alpha_i = ((\theta, \rho, \sigma), e)$ then there exists j $(0 \leq j < i)$ and $\sigma' \subseteq \sigma$ such that $\alpha_j = ((\theta, \rho, \sigma'), e')$.*

The correctness of this lemma follows from Table 2.3 by observing that every run is "peeled off" from the beginning, while taking into account that the *Match* predicate is defined such that it only extends the valuation of the variables.

The next lemma is used to infer from an encrypted message reception that the message must have been sent by an agent if it contains a component which is not known to the intruder. In most applications of this lemma we can infer l' by inspection of the role specification and we have $\langle inst \rangle(\{\!|\, m \,|\!\}_k) = \langle inst' \rangle(m')$, rather than a subterm relation.

Lemma 3.27. *Let α be a trace and let i be an index of α. If $\alpha_i = (inst, read_\ell(x, y, \{\!|\, m \,|\!\}_k))$ and $M_0 \not\vdash \langle inst \rangle(\{\!|\, m \,|\!\}_k)$, and $M_i^\alpha \not\vdash \langle inst \rangle(m)$, then there exists index $j < i$ such that $\alpha_j = (inst', send_{\ell'}(x', y', m'))$ and $\langle inst \rangle(\{\!|\, m \,|\!\}_k) \sqsubseteq \langle inst' \rangle(m')$.*

The correctness of this lemma follows from the fact that if the intruder does not know m when the message containing $\{m\}_k$ is read, he could not have constructed the encryption. Thus, it must have been sent as a subterm earlier.

The final lemma is characteristic for our model. It expresses that when two instantiations of a constant (such as a nonce or session key) are equal, they were created in the same run.

Lemma 3.28. *Let (θ, ρ, σ) and $(\theta', \rho', \sigma')$ be instantiations, and let $n \in Const$. If $\langle\theta, \rho, \sigma\rangle(n) = \langle\theta', \rho', \sigma'\rangle(n)$ we have $\theta = \theta'$.*

Theorem 3.29. *The Needham-Schroeder-Lowe protocol is correct in the Dolev-Yao intruder model with conspiring agents and without type flaws.*

Proof. We will sketch the proofs for $claim_7$ and $claim_9$. The other claims are proven analogously.

First observe that the intruder will never learn secret keys of trusted agents. This follows directly from Lemma 3.25, since none of the messages contain an encryption key in the message text. Since the set of keys known to the intruder is constant, it must be the case that if the intruder learns a basic term he learns it from unpacking an intercepted message which was encrypted with the key of an untrusted agent.

Proof outline We construct proofs for the Needham-Schroeder-Lowe protocol. The proof construction would fail for the Needham-Schroeder protocol, and we will use a marker † to indicate where the difference occurs. After the proof of $claim_7$, we briefly discuss this difference.

Both proofs will roughly follow the same structure. We examine the occurrence of a claim event in a trace of the system. Based on the rules of the semantics, we gradually derive more information about the trace, until we can conclude that the required property holds.

Proof of $claim_7$. In order to prove $claim_7$ we assume that α is a trace with index $r7$, such that $\alpha_{r7} = ((\theta_{r7}, \rho_{r7}, \sigma_{r7}), claim_7(r, secret, nr))$ and $rng(\rho_{r7}) \subseteq Agent_T$. Now we assume that the intruder learns nr and we will derive a contradiction. Let k be the smallest index such that $\langle\theta_{r7}, \rho_{r7}, \sigma_{r7}\rangle(nr) \in M_{k+1}$, and thus $\langle\theta_{r7}, \rho_{r7}, \sigma_{r7}\rangle(nr) \notin M_k$. Inspection of the derivation rules reveals that this increase in knowledge is due to an application of the send rule, followed by an application of the deflect rule. Therefore, there must be a smallest index $p < k$ such that $\alpha_p = ((\theta', \rho', \sigma'), send_\ell(m))$ and $\langle\theta_{r7}, \rho_{r7}, \sigma_{r7}\rangle(nr) \sqsubseteq \langle\theta', \rho', \sigma'\rangle(m)$. Since we have three possible send events in the NSL protocol, we have three cases: $\ell = 1, 2$, or 3.

$[\ell = 1]$ In the first case we have $\alpha_p = ((\theta', \rho', \sigma'), send_1(i, r, \{\!|\, i, ni \,|\!\}_{pk(r)}))$. Since constants i and ni both differ from nr, the intruder cannot learn $\langle\theta_{r7}, \rho_{r7}, \sigma_{r7}\rangle(nr)$ from $\langle\theta', \rho', \sigma'\rangle(i, r, \{\!|\, i, ni \,|\!\}_{pk(r)})$, which yields a contradiction.

$[\ell = 2]$ In the second case $\alpha_p = ((\theta', \rho', \sigma'), send_2(r, i, \{\!|\, W, nr, r \,|\!\}_{pk(i)}))$. The intruder can learn nr because $\rho'(i)$ is an untrusted agent and either $\langle\theta_{r7}, \rho_{r7}, \sigma_{r7}\rangle(nr) = \langle\theta', \rho', \sigma'\rangle(W)$ or $\langle\theta_{r7}, \rho_{r7}, \sigma_{r7}\rangle(nr) = \langle\theta', \rho', \sigma'\rangle(nr)$. We discuss both options separately.

(i) For the former equality we derive that $\langle\theta', \rho', \sigma'\rangle(W) \notin M_p$, so we can apply Lemmas 3.26 and 3.27 to find $i1$ with $\alpha_{i1} = ((\theta_{i1}, \rho_{i1}, \sigma_{i1}), send_1(i, r, \{\!|\, i, ni \,|\!\}_{pk(r)}))$. This gives $\langle\theta_{i1}, \rho_{i1}, \sigma_{i1}\rangle(ni) = \langle\theta', \rho', \sigma'\rangle(W) = \langle\theta_{r7}, \rho_{r7}, \sigma_{r7}\rangle(nr)$, which cannot be the case since ni and nr are distinct constants.

(ii) That the latter equality yields a contradiction is easy to show. Using Lemma 3.28 we derive $\theta_{r7} = \theta'$ and since run identifiers are unique, we have $\rho_{r7} = \rho'$. So $\rho_{r7}(i) = \rho'(i)$, which contradicts the assumption that $\rho_{r7}(i)$ is a trusted agent.

[$\ell = 3$] In the third case we have $\alpha_p = ((\theta', \rho', \sigma'), send_3(i, r, \{\!|\, V \,|\!\}_{pk(r)}))$. In order to learn $\langle \theta_{r7}, \rho_{r7}, \sigma_{r7} \rangle(nr)$ from $\langle \theta', \rho', \sigma' \rangle(i, r, \{\!|\, V \,|\!\}_{pk(r)})$ we must have that $\langle \theta', \rho', \sigma' \rangle(V) = \langle \theta_{r7}, \rho_{r7}, \sigma_{r7} \rangle(nr)$ and that $\rho'(r)$ is an untrusted agent. Using Lemma 3.26 we find index $i2$ such that $\alpha_{i2} = ((\theta', \rho', \sigma'), read_2(r, i, \{\!|\, ni, V, r \,|\!\}_{pk(i)})$. Because $\langle \theta', \rho', \sigma' \rangle(V) \notin M_p$ we can apply Lemma 3.27 to find index $r2$ with $\alpha_{r2} = ((\theta_{r2}, \rho_{r2}, \sigma_{r2}), send_2(r, i, \{\!|\, W, nr, r \,|\!\}_{pk(i)})$. This gives $\rho'(r) = \rho_{r2}(r)$. (†)

Next, we derive $\langle \theta_{r2}, \rho_{r2}, \sigma_{r2} \rangle(nr) = \langle \theta', \rho', \sigma' \rangle(V) = \langle \theta_{r7}, \rho_{r7}, \sigma_{r7} \rangle(nr)$. Applying Lemma 3.28 yields $\theta_{r2} = \theta_{r7}$ and thus $\rho_{r2} = \rho_{r7}$, so $\rho'(r) = \rho_{r2}(r) = \rho_{r7}(r)$. Because $\rho'(r)$ is an untrusted agent while $\rho_{r7}(r)$ is trusted, we obtain a contradiction. This finishes the proof of $claim_7$.

Note †*:* Please notice that the step in the proof marked with † fails for the Needham-Schroeder protocol, which gives an indication of why the hardening of the second message exchange is required.

Proof of $claim_9$. Let $\alpha \in Trace(nsl)$ be a trace of the system. Suppose that for some $r9$ and $(\theta_r, \rho_r, \sigma_{r9}) \in Inst$, with $rng(\rho_r) \subseteq Agent_T$, we have $\alpha_{r9} = ((\theta_r, \rho_r, \sigma_{r9}), claim_9(r, nisynch))$. In order to prove this synchronisation claim correct, we must find a run executing the initiator role which synchronises on the events labeled 1, 2, and 3, since $prec(nsl, 9) = \{1, 2, 3\}$. By applying Lemma 3.26, we find $r1$, $r2$, $r3$ ($0 \leq r1 < r2 < r3 < r9$) and $\sigma_{r1} \subseteq \sigma_{r2} \subseteq \sigma_{r3} \subseteq \sigma_{r9}$, such that

$$\begin{aligned}
\alpha_{r1} &= ((\theta_r, \rho_r, \sigma_{r1}), read_1(i, r, \{\!|\, i, W \,|\!\}_{pk(r)})) \\
\alpha_{r2} &= ((\theta_r, \rho_r, \sigma_{r2}), send_2(r, i, \{\!|\, W, nr, r \,|\!\}_{pk(i)})) \\
\alpha_{r3} &= ((\theta_r, \rho_r, \sigma_{r3}), read_3(i, r, \{\!|\, nr \,|\!\}_{pk(r)})).
\end{aligned}$$

We have already proved that nr remains secret, so we can apply Lemma 3.27 and find index $i3$ and $(\theta_i, \rho_i, \sigma_{i3})$ such that $i3 < r3$ and $\alpha_{i3} = ((\theta_i, \rho_i, \sigma_{i3}), send_3(i, r, \{\!|\, V \,|\!\}_{pk(r)})) \ \wedge \ \langle \theta_r, \rho_r, \sigma_{r3} \rangle(nr) = (\theta_i, \rho_i, \sigma_{i3}(V)$. By applying Lemma 3.26 we obtain $i1 < i2 < i3$ such that

$$\begin{aligned}
\alpha_{i1} &= ((\theta_i, \rho_i, \sigma_{i1}), send_1(i, r, \{\!|\, i, ni \,|\!\}_{pk(r)})) \\
\alpha_{i2} &= ((\theta_i, \rho_i, \sigma_{i2}), read_2(r, i, \{\!|\, ni, V, r \,|\!\}_{pk(i)})) \\
\alpha_{i3} &= ((\theta_i, \rho_i, \sigma_{i3}), send_3(i, r, \{\!|\, V \,|\!\}_{pk(r)})).
\end{aligned}$$

Now that we have found out that run θ_i is a candidate, we only have to prove that it synchronises with run θ_r. Therefore, we have to establish $r2 < i2$, $i1 < r1$ and that the corresponding send and read events match each other.

First, we observe α_{i2}. Since $\langle \theta_r, \rho_r, \sigma_{r3} \rangle(nr)$ is secret, $\langle \theta_i, \rho_i, \sigma_{i2} \rangle(V)$ is secret too and we can apply Lemma 3.27, obtaining index $r2' < i2$ such that

$\alpha_{r2'} = ((\theta_{r'}, \rho_{r'}, \sigma_{r2'}), send_2(r, i, \{\!|\, W, nr, r \,|\!\}_{pk(i)}))$ such that we have $\langle \theta_i, \rho_i, \sigma_{i2} \rangle(\{\!|\, ni, V, r \,|\!\}_{pk(i)}) = \langle \theta_{r'}, \rho_{r'}, \sigma_{r2'} \rangle(\{\!|\, W, nr, r \,|\!\}_{pk(i)})$. This implies that we have $\langle \theta_r, \rho_r, \sigma_{r3} \rangle(nr) = (\theta_i, \rho_i, \sigma_{i3}(V) = \langle \theta_{r'}, \rho_{r'}, \sigma_{r2'} \rangle(nr)$, so from Lemma 3.28 we have $\theta_r = \theta_{r'}$, and thus $r2 = r2'$. This establishes synchronisation of events α_{i2} and α_{r2}.

Next, we look at α_{r1}. Because $\langle \theta_r, \rho_r, \sigma_{r1} \rangle(W)$ is secret (cf. claim 8), we can apply Lemma 3.27, which gives index $i1' < r1$ such that $\alpha_{i1'} = ((\theta_{i'}, \rho_{i'}, \sigma_{i1'}), send_1(i, r, \{\!|\, i, ni \,|\!\}_{pk(r)}))$ and $\langle \theta_r, \rho_r, \sigma_{r1} \rangle(\{\!|\, i, W \,|\!\}_{pk(r)})) = \langle \theta_{i'}, \rho_{i'}, \sigma_{i1'} \rangle(\{\!|\, i, ni \,|\!\}_{pk(r)})$. Correspondence of α_{i2} and α_{r2} gives $\langle \theta_i, \rho_i, \sigma_{i2} \rangle(ni) = \langle \theta_r, \rho_r, \sigma_{r2} \rangle(W) = \langle \theta_r, \rho_r, \sigma_{r1} \rangle(W) = \langle \theta_{i'}, \rho_{i'}, \sigma_{i1'} \rangle(ni)$. By lemma 3.28 θ_i and $\theta_{i'}$ are equal, which establishes synchronisation of events α_{r1} and α_{i1}. This finishes the synchronisation proof of $claim_9$.

□

The proof method will form the outline of the verification method described in Chapter 4. Given the existence of certain events, we draw conclusions (directly based on the semantics) about previous events, branching options as they arise. The theorems used are generic and apply to all protocols. In contrast, e. g. Paulsons inductive proofs [152] require the establishment of specific theorems (e. g. unicity, secrecy) for each protocol.

3.7 Conclusions

In this chapter we introduced security properties to our model, defined in terms of local claim events. These local claim events denote the supposed properties of a security protocol, and are part of the protocol description. The main idea behind claim event is locality: agents have a local view on the state of the system, based on the messages they receive. The protocol should guarantee that based on the local view, the agent can be sure about some properties of the global state of the system, e.g. that something is not in the intruder knowledge, or that a certain agent is active.

We defined secrecy and several notions of authentication, and introduced a new strong form of authentication called synchronisation. In order to compare two notions of authentication, synchronisation and agreement, we formalised two forms of agreement: injective and non-injective agreement over all variables and all roles. Due to the uniform phrasing, the two notions of authentication can be distinguished easily: agreement allows that an intruder injects a (correct and expected) message before it is sent by the originator of the message. As for agreement, we provide both an injective and a non-injective variant of synchronisation.

Our definitions of synchronisation and agreement abstract away from the protocol and the semantic model as much as possible, e.g. they do not refer to the details of the message elements. Given a trace, we only need to have an equality relation between the contents of send and read events, and to know the ordering of the events

in a protocol description, in order to be able to verify every form of authentication defined here. This contrasts with other definitions of authentication, where often much more information about the protocol and its semantics is required to verify authentication for a given trace. In fact, for our approach, the definitions do not even require a trace semantics or a full ordering on the events within a role. The definitions will also work with the partially-ordered structures of the Strand Spaces model of [182], but also with the preorder on the events of a role of the AVISPA model in [104]; the only requirement on the role event order is that each event must have a finite set of preceding events.
From the definitions of synchronisation and agreement, we construct a hierarchy of authentication properties depicted in Figure 3.6. We show that with respect to the Dolev-Yao intruder model, injective synchronisation is strictly stronger than injective agreement.

Theorem 3.24 states that, for a large class of security protocol models including the model defined in this thesis, injectivity of authentication protocols is easy to verify, once synchronisation has been established. Until now, injectivity and authentication have been strongly connected. Our new results establish that it suffices to verify the non-injective variant of synchronisation. Verifying injectivity is a simple and separate task, which does not depend on any specific (data) model.
For our injectivity result, we did not choose a specific security protocol model. Instead, as already mentioned, we have characterised a class of models in which Theorem 3.24 holds. This class contains nearly all models found in the literature, such as the Strand Spaces model, Casper/FDR without time, and term rewrite systems [182, 121, 89], as well as many models that allow for non-linear (branching) protocol specifications. These models share the following properties:

(i) Multiple instances of the protocol are truly independent. They do not share variables, memory, or time.

(ii) The intruder has the ability to duplicate messages, as holds, for example, in the standard Dolev-Yao intruder model.

The question arises whether the theorem also holds in an intruder-less model. This is in fact the case, but of less interest, because injectivity always holds for synchronising or agreeing protocols when there is no intruder.
Automated verification of the *LOOP* property can be implemented easily. The algorithm is an instance of the reachability problem in a finite acyclic graph, and therefore has linear complexity.

Almost all correct authentication protocols in the literature satisfy *NI-SYNCH* as well as *LOOP*. It seems that *LOOP* is a necessary condition for injectivity, in particular for the Dolev-Yao intruder model. However, for peculiar intruder models, *LOOP* is not a necessary condition for injectivity.
In the models where *LOOP* is also a necessary condition for injectivity, our results imply a minimum number of messages in a multi-party authentication protocol. We will investigate this in future work.
The *LOOP* property guarantees injectivity for synchronising protocols. This raises

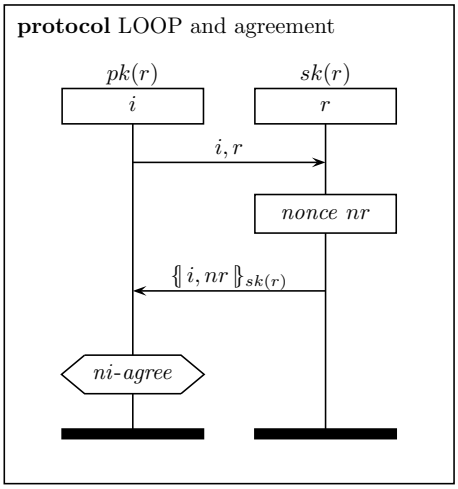

Figure 3.15: A unilateral agreement protocol, with *LOOP*, but not injective.

the question whether there is a similar property to show injectivity of agreeing protocols. It can be seen from the example in Figure 3.15 that *LOOP* does not suffice to guarantee injectivity. The protocol satisfies the loop property for the claim role, and the protocol satisfies non-injective agreement, but not injective agreement.

In this chapter we defined notions of secrecy and authentication, but it is possible to express other security properties, such as e. g. non-repudiation, in a similar way.

We have also illustrated how the security protocol model from Chapter 2 can be used to prove security properties correct. We used the model and the definitions of secrecy and synchronisation to prove two properties of the Needham-Schroeder-Lowe protocol.

The advantage of correctness proofs is that they guarantee that within the model, no attacks on the security protocol exist. However, there are several disadvantages to constructing such proofs by hand. For each claim in each protocol, a new proof is required. Each minor modification to the messages of a protocol can change the proof significantly. Also, if we cannot find a proof, we cannot make any statement about the protocol: it may still be correct, or flawed.

Ideally, we would like to establish either correctness of a security claim of protocol, or find a concrete attack against a claim. In the next chapter we address automated verification (establishing correctness) and falsification (finding attacks) of security protocol claims.

4

Verification

In the previous two chapters we introduced a model for security protocols and various security properties. In this chapter we address verification of security properties, and develop an automated verification method for security properties defined in our model.

The verification procedure developed here is based on verification of security properties by analysing patterns. Instead of analysing all individual traces, analysis involves classes of traces, defined by trace patterns. Trace patterns are defined by partially ordered, symbolic sets of events. We start off by explaining trace patterns in Section 4.1.

Trace patterns allow us to capture the class of all attack traces for a given protocol and security property: in other words, all attacks on a protocol property must exhibit a particular pattern, induced by the security property. If there exist traces of a protocol that exhibit a certain attack pattern, each such trace violates the security property, and we can conclude that the security property is false. If no trace of the protocol exhibits the attack pattern, there exists no attack, and the property is true.

In order to verify a security property of a given protocol, we want to determine whether or not the attack pattern occurs in any trace. This is where the verification procedure comes into the picture. In Section 4.2 we develop an iterative algorithm that given a protocol and a trace pattern, decides whether or not traces of the protocol exist in which the pattern occurs. There are two possible results of the algorithm, (1) no such traces exist in the protocol, or (2) there exist traces of the protocol that include the pattern. In this case, the algorithm returns a special kind of patterns, called explicit trace patterns, which capture all the possible ways in which the original pattern can occur in traces of the protocol.

The developed algorithm is not guaranteed to terminate. We address this issue in Section 4.2.5, and present a modified version of the algorithm that is guaranteed to terminate, by bounding the number of runs. This bounded version of the algorithm has four possible outcomes: two as in the unbounded version, and two bounded variants. The two new outcomes are (3) there exist no traces of the protocol that include the pattern within the bound, or (4) there exist traces that include pattern within the bound (for which explicit trace patterns are produced), but there might be other ways in which the pattern can occur. Even though a bound is introduced,

the algorithm can decide security properties for the majority of protocols, for any number of runs.

In Section 4.3 we show how this algorithm can be used to effectively verify security properties. In particular, we sketch how the algorithm can be used to verify secrecy and authentication properties such as synchronisation.

In Section 4.4 we discuss a prototype implementation of the verification method called *Scyther* and perform empirical analysis using the prototype. We investigate the influence of the heuristics of the algorithm, and the choice of the bound, for a large set of protocols.

We finish this chapter in Section 4.5 by drawing some conclusions.

4.1 Trace patterns

The algorithm we develop in the next section is based on reasoning about classes of traces, as opposed to individual traces. We therefore start off by defining trace patterns, which will be used to represent classes of traces.

4.1.1 Representing classes of traces by trace patterns

In general, a protocol has many different possible behaviours (traces), particularly in the presence of an active intruder. However, in the light of property verification, many of these traces are just different interleavings or renamings, of similar behaviours. In order to capture the concept of similar traces, we introduce a way to represent classes of traces, by means of trace patterns, defined as *partially ordered*, and *symbolic* sets of events.

We consider symbolic events (and terms occurring in them) as patterns, which can be instantiated in many ways. Relating to the concepts of Chapter 2, symbolic terms and events are nothing more than events with *incomplete* instantiations. To accommodate for incomplete instantiations, we extend the role and variable mapping rules from the definition of term instantiation (Definition 2.22 on page 22), which results in:

Definition 4.1 (Symbolic term instantiation). *For instantiation $(\theta, \rho, \sigma) \in Inst$, $f \in Func$ and terms $rt, rt1, \ldots, rtn \in RoleTerm$, we define symbolic instantiation by:*

$$\langle \theta, \rho, \sigma \rangle (rt) = \begin{cases} \rho(R) & \text{if } rt \equiv R \in Role \cap dom(\rho) \\ f((\theta, \rho, \sigma)(rt1), \ldots, (\theta, \rho, \sigma)(rtn)) & \text{if } rt \equiv f(rt1, \ldots, rtn) \\ c\sharp\theta & \text{if } rt \equiv c \in Const \\ \sigma(v) & \text{if } rt \equiv v \in Var \cap dom(\sigma) \\ (\langle \theta, \rho, \sigma \rangle (rt1), \langle \theta, \rho, \sigma \rangle (rt2)) & \text{if } rt \equiv (rt1, rt2) \\ \{\!| \langle \theta, \rho, \sigma \rangle (rt1) |\!\}_{\langle \theta, \rho, \sigma \rangle (rt2)} & \text{if } rt \equiv \{\!| rt1 |\!\}_{rt2} \\ R\sharp\theta & \text{if } rt \equiv R \in Role \setminus dom(\rho) \\ v\sharp\theta & \text{if } rt \equiv v \in Var \setminus dom(\sigma) \end{cases}$$

The difference between this definition and the old one, is that the last two rules are new. Symbolic instantiation does not require that ρ and σ are defined for all roles and variables that occur in the term. We have added two clauses to cover these cases. Thus, throughout this chapter we will use $\langle inst \rangle(rt)$ to denote *symbolic instantiation*. Correspondingly, the set of run terms is extended with basic run terms of the form $R\sharp\theta$ (for uninstantiated roles) and $V\sharp\theta$ (for uninstantiated variables).

The upshot of this is that we can now have run term patterns: a run term $(A, R\sharp\theta)$ effectively represents a class of run terms, of which $(A, A), (A, B)$ etc., are members.

We extend this to instantiating role events, yielding instantiated role events.

Definition 4.2 (Instantiated role events). *For all instantiations (θ, ρ, σ), role events ε, role terms rt and labels ℓ, we define instantiated role events as*

$$\langle(\theta,\rho,\sigma)\rangle(\varepsilon) = \begin{cases} send_\ell(\langle(\theta,\rho,\sigma)\rangle(rt))\sharp\theta & \text{if } \varepsilon = send_\ell(rt) \\ read_\ell(\langle(\theta,\rho,\sigma)\rangle(rt))\sharp\theta & \text{if } \varepsilon = read_\ell(rt) \\ claim_\ell(\langle(\theta,\rho,\sigma)\rangle(rt))\sharp\theta & \text{if } \varepsilon = claim_\ell(rt) \end{cases}$$

The set of trace pattern events *PatternEvent* contains instantiated role events:[1]

$$PatternEvent \supseteq \left\{ \langle inst \rangle(\varepsilon) \;\middle|\; inst \in Inst, \varepsilon \in RoleEvent \right\}$$

Definition 4.3 (Trace pattern). *A trace pattern is defined as $(PatternEvent, \rightarrow)$, with a finite set of edges from $\rightarrow$ connecting a finite number of vertices from PatternEvent. Edges may optionally be labeled with run terms, as in $e1 \xrightarrow{rt} e2$. The set of all trace patterns is denoted by Pattern.*

A trace pattern is a partially ordered, symbolic representation of a set of traces.

We define $\rightarrow^*$ as the smallest transitive relation containing $\rightarrow$ and $\xrightarrow{rt}$ for any term rt. Its purpose is to capture the ordering on events induced by the combination of the labeled and unlabeled versions of $\rightarrow$.

In the remainder of this chapter the notation $e1 \rightarrow e2$ is used for both unlabeled and labeled edges.

The trace patterns correspond to sets of concrete traces. To translate a pattern back into a concrete trace, all variables occuring in the trace pattern must be instantiated. Furthermore, when considering patterns, we abstract away from the actual choice of concrete run identifiers.

Definition 4.4 (Pattern instantiation function). *We call $f : PatternEvent \nrightarrow RunEvent$ a pattern instantiation function for a pattern $(TPE, \rightarrow)$ if and only if f is a substitution function, mapping (a) all variables occurring in TPE to run terms, and (b) all role names occurring in TPE to agents, and (c) injectively maps all run identifiers to run identifiers. The set of pattern instantiation functions for a pattern TP is denoted by $PatternInst_{TP}$.*

[1] The complete set of trace pattern events will be given in Definition 4.18.

Contrary to the previously used instantiation functions, a pattern instantiation function substitutes variables and roles globally, as opposed to locally, because variables of a run can occur in events of other runs.

Example 4.5 (Pattern instantiation). *Given the trace pattern*

$$(\{\ \mathrm{send}_1(B, R\sharp 2, v\sharp 1)\sharp 3\ ,\ \mathrm{send}_1(R\sharp 2, C, (A, v\sharp 1))\sharp 2\ \}, \rightarrow)$$

we have that $\{v\sharp 1 \mapsto ni\sharp 2, R\sharp 2 \mapsto A, 1 \mapsto 1, 2 \mapsto 2\}$ *is a pattern instantiation function.*

Next we define the concrete traces that match the pattern, using the pattern instantiation function.

Definition 4.6 (Trace class defined by a trace pattern). *Let* $(TPE, \rightarrow) \in$ *Pattern be a trace pattern. The set of traces it represents is defined as*

$$traces((TPE, \rightarrow)) = \Big\{ t \;\Big|\; t \in RunEvent^* \wedge f \in PatternInst_{(TPE,\rightarrow)} \wedge (e \in TPE \Rightarrow f(e) \in t)\ \wedge\ (e1 \rightarrow e2 \Rightarrow f(e1) <_t f(e2)) \Big\}$$

where f *is a pattern instantiation function conform the previous definition.*

In other words: a concrete trace exhibits the pattern, if there exists an instantiation function f, such that the instantiations of all pattern events occur in the trace, in the order prescribed by the pattern.

Definition 4.7 (Protocol traces exhibiting a trace pattern). *Let* P *be a protocol, and let* $(TPE, \rightarrow)$ *be a trace pattern. We define the set of traces of the protocol that exhibit the pattern as*

$$traces(P, (TPE, \rightarrow)) = traces(P) \cap traces((TPE, \rightarrow))$$

Example 4.8 (Trace pattern). *Consider the events of the Needham-Schroeder protocol as defined on page 16. All traces that include an instance of the synchronisation claim of the responder are defined by the pattern*

$$(\{e\}, \emptyset)$$

with

$$e = claim_5(r\sharp 1, ni\text{-}synch))\sharp 1$$

The traces of the Needham-Schroeder protocol that exhibit this pattern, include the attack on the protocol, and the traces that we consider to be the expected execution of the protocol.

Not all trace patterns can occur in the traces of a protocol, as the following example shows.

Example 4.9 (Trace pattern). *Let P be a protocol with two events $\varepsilon 1, \varepsilon 2$ such that $\varepsilon 1 \prec_r \varepsilon 2$. Let $e1 = \langle inst \rangle(\varepsilon 1)$ and $e2 = \langle inst \rangle(\varepsilon 2)$ for some instantiation function inst. Then we have that*

$$traces(P, (\{e1, e2\}, \{e2 \rightarrow e1\})) = \emptyset$$

This pattern does not occur in the traces of any protocol, because according to the operational semantics, the events of a single run of any protocol conform to the protocol role order.

Definition 4.10 (Trace pattern refinement). *Let TP and TP' be trace patterns. We write $TP' \sqsubseteq TP$ to denote that TP' is a refinement of TP, if and only if*

$$traces(TP') \subseteq traces(TP)$$

We say that TP' is a refinement of TP, if all traces of TP' are also trace of TP. In particular, we have that TP' contains more structure, or less choice, than TP.

Example 4.11 (Trace pattern refinement). *We give three examples of operations that refine a trace pattern. First, consider adding an event to a pattern. Let $(\rightarrow, TPE)$ be a trace pattern, and let e be a trace event. Then we have that*

$$(TPE \cup \{e\}, \rightarrow) \sqsubseteq (TPE, \rightarrow)$$

A second form of refinement is adding an edge to a trace pattern. Let $(\rightarrow, TPE)$ be a trace pattern, let $\{e1, e2\} \subseteq TPE$, and rt a term. Then we have that

$$(TPE, \rightarrow \cup (e1 \xrightarrow{rt} e2)) \sqsubseteq (TPE, \rightarrow)$$

A third form is substitution of uninstantiated variables. Let $(\rightarrow, TPE)$ be a trace pattern, let $V\sharp\theta$ be an uninstantiated variable occurring in TPE, and let rt be a term. Then we have that

$$(TPE, \rightarrow)[rt/V\sharp\theta] \sqsubseteq (TPE, \rightarrow)$$

The resulting trace set contains less choice: where we first had the option of choosing any value for the variable $V\sharp\theta$ in concrete traces, we now fix the choice to a particular term rt.

That the above three operations constitute refinements is a direct result of definition 4.6.

4.1.2 Realizable trace patterns

Now that we have defined trace patterns, we investigate when a trace pattern is *realizable* with respect to a given protocol, i.e. when there exist traces of the protocol that include the pattern. We define three predicates on protocols and trace patterns. The predicates find their origin in the rules of the operational semantics, and have been tailored specifically to allow us to conclude that the intersection of the traces of

the protocol and the pattern is not empty, i. e. that there exist traces of the protocol that exhibit the pattern.

Because labels of role events are unique within a protocol definition, we define an auxiliary function *roleevent* that maps trace pattern events to their unique role events.

Definition 4.12 (Unique runs in a trace pattern). *The first predicate UniqueRuns on trace patterns expresses that role events should not occur twice with the same run identifier.*

$$\begin{aligned} &\mathit{UniqueRuns}((\mathit{TPE}, \rightarrow)) = \forall e, e' \in \mathit{TPE} : \\ &\qquad \mathit{roleevent}(e) = \mathit{roleevent}(e') \wedge \mathit{runidof}(e) = \mathit{runidof}(e') \Rightarrow e = e' \end{aligned}$$

This predicate does not refer to any protocol. Rather, it is a necessary requirement of a pattern to correspond to any traces of the semantics in general. As a result we have the following theorem.

Theorem 4.13 (Unique runs). *Let TP be a trace pattern, and let P be a protocol. Then*

$$\neg \mathit{UniqueRuns}(\mathit{TP}) \Rightarrow \mathit{traces}(P, \mathit{TP}) = \emptyset$$

The proof is a direct result of the rules of the operational semantics, as each run executes each role event only once, and role events are unique within role definitions.

Definition 4.14 (Role consistent trace pattern). *The predicate RoleConsistent expresses consistency of a trace pattern $(\mathit{TPE}, \rightarrow)$ according to the individual role definitions of a protocol. Let P be a protocol, with the role event order $\prec_r$ for each role r.*

$$\begin{aligned} &\mathit{RoleConsistent}(P, (\mathit{TPE}, \rightarrow)) = \forall e \in \mathit{TPE} : \\ &\qquad \exists \mathit{inst} \in \mathit{Inst}, \varepsilon \in \mathit{RoleEvent} : e = \langle \mathit{inst} \rangle(\varepsilon) \wedge \\ &\qquad\qquad (\forall \varepsilon' \prec_r \varepsilon : \exists \mathit{inst}' : \mathit{inst}' \subseteq \mathit{inst} \wedge \langle \mathit{inst}' \rangle(\varepsilon') \rightarrow^* e) \end{aligned}$$

This predicate expresses that each event in the trace pattern must be an instantiation of a role event. Furthermore it must be preceded by instantiated role events in the order prescribed by the role definition.

Contrary to the *UniqueRuns* predicate, the *RoleConsistent* predicate is not necessary for a pattern in order to represent any traces of a protocol . As an example, consider the pattern *TP* that includes only the second event of a role of a well-formed protocol P. Clearly, *TP* does not satisfy *RoleConsistent*. However, there exist traces in which this pattern occurs, such as in traces in which the protocol P is executed exactly as prescribed by the role definitions.

For the remainder of this chapter, we restrict the discussion to the strongest intruder model, for which we have

$$\mathit{Networkrulename} ::= \mathit{take}(\mathit{RunTerm}) \mid \mathit{fake}(\mathit{RunTerm})$$

The notion of a trace pattern and the resulting theorems can easily be modified for other intruder models, but this would introduce a large amount of case distinctions. For clarity, we choose only to discuss the strongest intruder model. In this intruder model, read events are enabled if and only if a term matching their pattern can be inferred from the intruder knowledge, which consists of all sent terms, conform Lemma 2.47. This is captured by the third predicate on trace patterns.

Definition 4.15 (Read enabled trace pattern). *The predicate $\mathit{ReadEnabled}(P, TP)$ expresses that read events of the pattern TP of the protocol P are enabled, which corresponds to the intruder being able to generate the correct message.*

$$\mathit{ReadEnabled}(P, (TPE, \rightarrow)) = \forall \ell, \theta, rt : \mathit{read}_\ell(rt)\sharp\theta \in TPE \Rightarrow \\ M_0 \cup \left\{ rt' \;\middle|\; \exists \ell', \theta' : \mathit{send}_{\ell'}(rt')\sharp\theta' \rightarrow^* \mathit{read}_\ell(rt)\sharp\theta \right\} \vdash rt$$

A trace pattern of a protocol for which the above three properties hold, is called a realizable trace pattern of the protocol.

Definition 4.16 (Realizable trace pattern). *Let TP be a trace pattern, and let P be a well-formed protocol, such that all events in the pattern are instantiations of events of P. If we have that $\mathit{UniqueRuns}(TP)$, $\mathit{RoleConsistent}(P, TP)$ as well as $\mathit{ReadEnabled}(TP)$, then we say that trace pattern TP is realizable with respect to P.*

A trace pattern for which the above three properties holds is called realizable because there exist traces of the protocol that exhibit the pattern.

Theorem 4.17 (Realizable trace pattern occurs in traces of the protocol). *Let TP be a realizable trace pattern of a well-formed protocol P. Then we have that $\mathit{traces}(P, TP) \neq \emptyset$.*

Proof. Let $(TPE, \rightarrow)$ be a trace pattern and P a protocol, such that the above conditions hold. We will construct a sequence of transitions starting from the initial state of the semantics, corresponding to the rules in Table 2.3 on page 27, thus creating a trace that is a trace of the protocol by construction. Initially, we have that the trace is empty, $t = [\,]$, the network state $s = \langle M, BS, BR, F\rangle$ is the initial state, i. e. $M = M_0$, and $BS = BR = F = \emptyset$.

As an invariant, we have that at each step in the construction, each combination of role event with a run identifier (which is unique based on *UniqueRuns*) occurs either in TPE or in t, but not in both. In each step, such an event is removed from TPE and added to t.

We repeat the following procedure until TPE is empty.

Let e be an element TPE, such that $\neg\exists e' \in TPE : e' \rightarrow e$. Because TPE is finite and $\rightarrow$ is acyclic, such an event exists. Let $\theta = \mathit{runidof}(e)$, and let $\varepsilon = \mathit{roleevent}(e)$. If θ does not occur in e, the run θ is not part of the state s yet, and we apply the create rule. Because *RoleConsistent* holds, there exists a (partial) instantiation function *inst* such that all events of the run θ in TPE are instantiations of role

events with this instantiation function. Let $inst_\theta = (\theta, \sigma_\theta, \rho_\theta)$ be any instantiation function such that (1) $inst \subseteq inst_\theta$, (2) $inst_\theta$ is not partial, in the sense that all role names and variables occurring in role $role(e)$ are mapped to terms. Let $run = ((\theta, \rho_\theta, \emptyset), P(role(e)))$. We extend t by this create event.

$$t := t \cdot create(run)$$

and consequently we extend the set of runs F with this run.

Next we apply case distinction on the type of event of e: claim, send or read.

- If e is a claim event, we observe that by construction, there is a create event in t for this run. Furthermore, exactly those events that precede it in the run are already part of t based on the invariant and *RoleConsistent*. Thus, the set of runs F contains an element $run = (inst, \varepsilon \cdot s)$, for some $inst$ and some s. Thus, we meet the requirements for applying the [claim] rule, and add the event to the trace
$$t := t \cdot (inst, \varepsilon)$$
and we update F according to the rule.

- If e is a send event, we follow the same procedure as for the claim event, except that we additionally add the sent message m to BS, and add a *take* event to the claim, expressing that the intruder learns m.
$$t := t \cdot (inst, \varepsilon) \cdot take(m)$$
Effectively, the message is copied from the send buffer to the intruder knowledge, and we extend M accordingly: $M := M \cup \{m\}$.

- If e is a read event, note that the precondition for the rule is significantly more involved than the other ones. Additionally, it requires that the message that is to be read is in the read buffer, and that this matches with the pattern m. Observe that based on *ReadEnabled*, we have that the message m, as read by e can be inferred from the preceding send events. By construction, these are already in the trace, and the messages are in the intruder knowledge, because of the added *take* events. Note that the instantiation of variables by concrete terms preserves the inference relation: if we have that $T \vdash t$, and we substitute role names and variables in T and t by concrete terms, the relation still holds. Let $inst$ be the instantiation function that occurs last in t for the run θ. The pattern m can contain variables or role names not yet defined in $inst$. For these, we extend $inst$ and define them to be as in $inst_\theta$, ensuring that the instantiation of the pattern is a concrete term that does not contain variables. However, although the resulting concrete message $\langle inst \rangle(m)$ can be inferred from M, the messages that can be inferred are not in the read buffer. Thus, we first insert a *fake* event for the message m. After that we can add the read event to the trace as for the claim event.
$$t := t \cdot fake(m) \cdot (inst, \varepsilon)$$

Finally we remove e from TPE, and repeat this procedure until TPE is empty.

The trace constructed in this way is a trace of the protocol by construction. □

4.1.3 Explicit trace patterns

In order to give an algorithm that can determine whether a trace pattern contains only traces of the protocol, we construct a variation on the notion of trace pattern, which we call an *explicit trace pattern*.

The main difference of an explicit trace pattern when compared to a trace pattern is that the construction (involving the use of the encrypt or decrypt rules of the knowledge inference operator) of the intruder knowledge is made explicit. We use the labels on the ordering relation, which will now have the following interpretation: $e1 \xrightarrow{rt} e2$ denotes that the term rt, where rt is not a tuple, occurs first in the intruder knowledge after event $e1$, and is needed to enable event $e2$. However, some non-tuple terms such as e.g. $\{\!|\, rt1 \,|\!\}_{rt2}$ might be constructed from two terms, and we would not be able to determine a single point where they both occur first. To express the event after which such a composed term occurs first, we introduce two new events: the *encr* and *decr* events. We can now give a complete definition of the set of trace pattern events.

Definition 4.18 (Trace pattern events). *The set PatternEvent of all trace pattern events is defined as*

$$\begin{aligned} PatternEvent = &\{\langle inst\rangle(\varepsilon) \mid inst \in Inst, \varepsilon \in RoleEvent\} \cup \\ &\{encr(rt1, rt2, \{\!|\, rt1 \,|\!\}_{rt2}) \mid rt1, rt2 \in RoleTerm\} \cup \\ &\{decr(\{\!|\, rt1 \,|\!\}_{rt2}, rt2^{-1}, rt1) \mid rt1, rt2 \in RoleTerm\} \end{aligned}$$

Note that whereas traces of a protocol contain events as tuples $(inst, \varepsilon)$, trace patterns contain concrete events $\langle inst\rangle(\varepsilon)$ to which the substitutions prescribed by $inst$ have been applied.

For the remainder of this chapter, we abstract away from the initial intruder knowledge M_0. The reason for this is that including it would lead to a large number of (trivial) case distinctions, and furthermore it can also be easily modeled as a single role of the protocol that contains a send event which contains the base terms from M_0, from which all others can be constructed.

In order to formally express explicit trace patterns, we need to introduce some auxiliary notation. These correspond directly to the intuitive notions of (1) the parts of a term after projecting tuples, (2) the non-tuple terms that are added to the intruder knowledge after an event, and (3) the non-tuple terms that are required to be in the intruder knowledge in order to enable an event.

For all terms we define the function $parts : RunTerm \to \mathcal{P}(RunTerm)$ such that

$$parts(rt) = \begin{cases} parts(rt1) \cup parts(rt2) & \text{if } rt = (rt1, rt2) \\ \emptyset & \text{if } rt = V\sharp\theta \\ \{rt\} & \text{otherwise} \end{cases}$$

Thus, we consider all non-tuple terms that are not (uninstantiated) variables to be a *part* of the term. Uninstantiated variables are omitted as the initial intruder knowledge contains intruder constants of all types (*IntruderConst*).

For all labels ℓ and terms $rt1, rt2, rt3$, the parts that are considered the output of an event e are defined as $out : RunEvent \to \mathcal{P}(RunTerm)$, where

$$out(e) = \begin{cases} parts(rt1) & \text{if } e = send_\ell(rt1) \\ parts(rt3) & \text{if } e = encr(rt1, rt2, rt3) \\ parts(rt3) & \text{if } e = decr(rt1, rt2, rt3) \\ \emptyset & \text{otherwise} \end{cases}$$

For all labels ℓ and terms $rt1, rt2, rt3$, the parts that are considered the input of an event e are defined as $in : RunEvent \to \mathcal{P}(RunTerm)$

$$in(e) = \begin{cases} parts(rt1) & \text{if } e = read_\ell(rt1) \\ parts(rt1) \cup parts(rt2) & \text{if } e = encr(rt1, rt2, rt3) \\ parts(rt1) \cup parts(rt2) & \text{if } e = decr(rt1, rt2, rt3) \\ \emptyset & \text{otherwise} \end{cases}$$

Using these notions, we can define explicit trace patterns.

Definition 4.19 (Explicit trace pattern). *We call a trace pattern $(TPE, \to)$ explicit if and only if the following four properties hold:*

(i) If a term occurs first after an event e, it is an element of its out(e) set:

$$\forall e1, e2 \in TPE, rt : e1 \xrightarrow{rt} e2 \Rightarrow rt \in out(e1)$$

(ii) The intruder can only learn a term once first

$$\forall e\, TPE, rt : e \xrightarrow{rt} \ldots \Rightarrow \neg\exists e' \in TPE : e' \to^* e \ \wedge \ rt \in parts(out(e'))$$

(iii) The intruder learns a term first after a unique event

$$\forall e1, e2 \in TPE, rt : e1 \xrightarrow{rt} \ldots \wedge e2 \xrightarrow{rt} \ldots \Rightarrow e1 = e2$$

(iv) All events are enabled: the terms that are required for an event e (defined as elements of the set in(e)) occur first before the event e.

$$\forall e \in TPE : in(e) = \left\{ rt \;\middle|\; e2 \in TPE \ \wedge \ e2 \xrightarrow{rt} e \right\}$$

Three examples of explicit trace patterns (with additional annotations) can be found in Figures 4.1, 4.2 and 4.3.

Lemma 4.20 (Explicit trace patterns satisfy ReadEnabled). *Every explicit trace pattern $(TPE, \to)$ of a protocol P satisfies ReadEnabled.*

Proof. The proof proceeds by iterating over the labeled edges, marking them as done as they are addressed.

As an invariant we take that all terms occurring as labels of edges that are marked as done, can be inferred from the preceding set of send events. Any uninstantiated variables can safely be ignored, as they can be instantiated by intruder constants, as the set M_0 includes intruder constants of any occurring type.

Let $e1 \xrightarrow{rt} e2$ be an unmarked labeled edge such that there is no unmarked edge that precedes it in the graph. We apply case distinction on the type of event of $e1$:

- $e1$ is a claim or read event. By definition, we have that $out(e1) = \emptyset$, so this case cannot occur.
- $e1$ is a send event. By definition of explicit trace pattern, rt is an element of the parts of the sent term. Thus, rt can be inferred from the sent message by (repeatedly) applying projection, as $(rt1, rt2) \vdash rt1$ and $(rt1, rt2) \vdash rt2$.
- $e1$ is a decrypt event. For an explicit trace pattern, we have that there exists m,k, such that $rt \in parts(m)$. Furthermore, we have that $\{\!|\, m \,|\!\}_k$,$k^{-1} \in in(e1)$. Based on the invariant and the fact that $(TPE, \rightarrow)$ is explicit, we have that $\{\!|\, m \,|\!\}_k$ and k^{-1} are inferable from the preceding events. Consequently, we have that rt is inferable (because of $\{\{\!|\, m \,|\!\}_k, k^{-1}\} \vdash m$, and projection).
- $e1$ is an encrypt event. This situation is similar to decryption.

Afterwards, we mark the edge as done.

We apply this procedure until all labeled edges (of which there are finitely many) are marked as done. □

Corollary 4.21 (Satisfiability of explicit trace patterns). *Every explicit trace pattern $(TPE, \rightarrow)$ of a protocol P that satisfies RoleConsistent$(P, (TPE, \rightarrow))$ as well as UniqueRuns$(TPE, \rightarrow)$, is also realizable.*

This corollary is a consequence of Theorem 4.17 and Lemma 4.20.

4.1.4 Complete characterization

Given a trace pattern and a protocol, we are interested in the existence of traces of the protocol in which the pattern occurs. In particular, we are interested in expressing the class of traces of the protocol that exhibit the pattern as a set of explicit trace patterns. We refer to the process of capturing such a trace class by means of a set of explicit trace patterns, as *complete characterization*. We owe this terminology to the (independently developed) work of [78], in which a different approach is taken to arrive at such characterizations. We will return to this work later in Chapter 7.

Definition 4.22 (Complete characterization). *Let P be a protocol, and let TP be a trace pattern. Let ES be a set of* explicit *trace patterns. We call ES a* complete characterization *of the set $traces(P, TP)$, if and only if*

$$\bigcup_{E \in ES} traces(P, E) = traces(P, TP)$$

Example 4.23 (Complete characterization of Needham-Schroeder). *In order to give an idea of what a complete characterization looks like, we give a complete characterization of the intersection of two trace patterns with the Needham-Schroeder protocol: the trace pattern that corresponds to an instance of the initiator role of the Needham-Schroeder protocol, and the trace pattern that corresponds to an instance of the responder role of the protocol.*

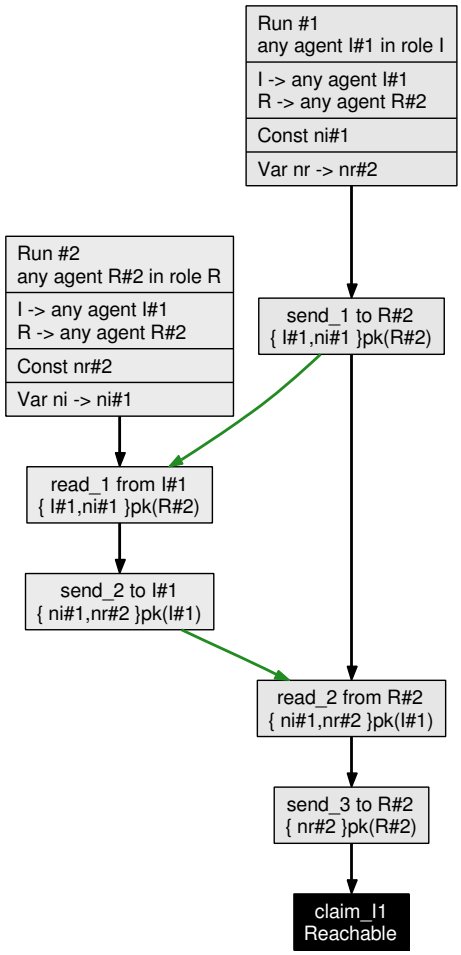

Figure 4.1: Needham-Schroeder, role I: pattern 1/1

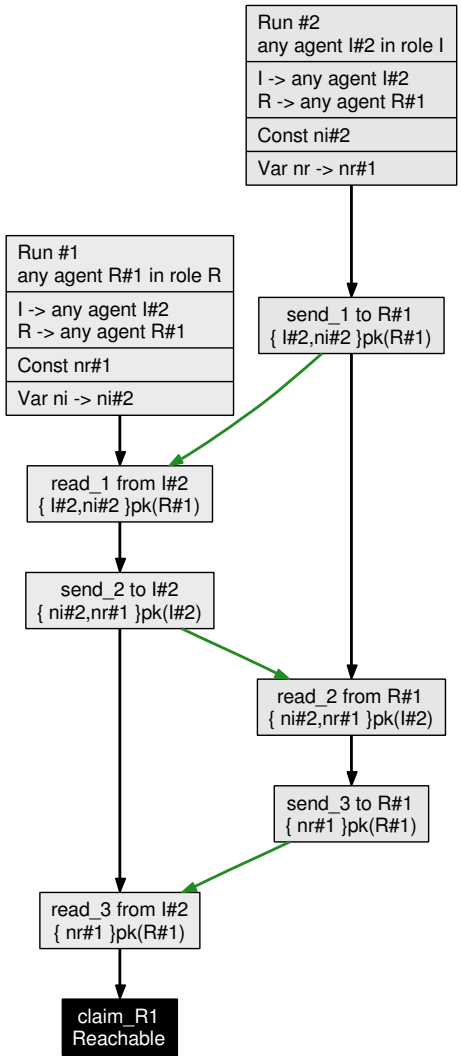

Figure 4.2: Needham-Schroeder, role R: pattern 1/2

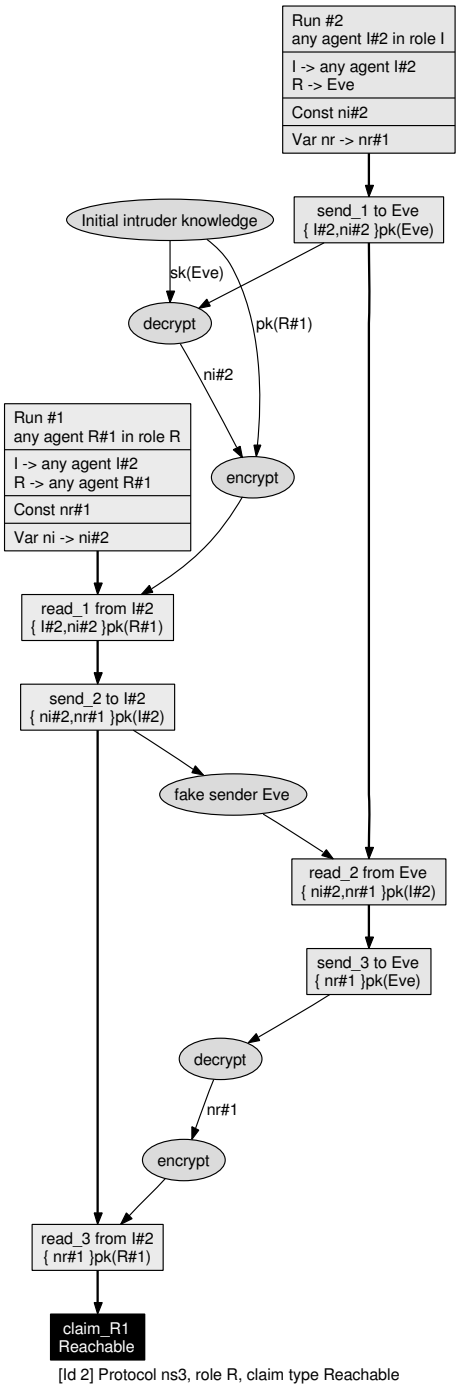

Figure 4.3: Needham-Schroeder, role R: pattern 2/2

For the initiator role of the Needham-Schroeder protocol, there is only one explicit trace pattern, shown in Figure 4.1 on page 80. Thus, all traces that include the initiator role, must also include the structure in the graph, which exactly corresponds to a valid protocol execution. Thus, any synchronisation claim at the end of the initiator role is correct.

For the responder role, there are exactly two explicit trace patterns, shown in Figures 4.2 and 4.3. The first of these corresponds to the expected protocol execution, whilst the second is exactly the man-in-the-middle attack originally found by Lowe. As a side result, this shows that every attack on the authentication properties of Needham-Schroeder includes the man-in-the middle attack, as the two graphs represent a complete characterization of the responder role. Each trace that includes an instance of the responder role either contains also a partner to synchronize with, or it contains the man-in-the-middle attack.

All three figures are unmodified output of the prototype tool discussed in Section 4.4.

4.2 Characterization algorithm

In this section we present an iterative algorithm Ψ with the following signature:

$$\Psi : \mathit{Protocol} \times \mathit{Pattern} \to \mathcal{P}(\mathit{Pattern})$$

Given a trace pattern TP and a protocol P the algorithm Ψ yields a set of *explicit* trace patterns that represent a complete characterization of the protocol, i. e.

$$\bigcup_{E \in \Psi(P,TP)} \mathit{traces}(P, E) = \mathit{traces}(P, TP)$$

The algorithm has its origins in the Athena algorithm developed for Strand Spaces, as described in [172], although the used framework (and the resulting scope) differs significantly.

4.2.1 Basic idea

The intuition behind the algorithm is that a trace pattern can only occur in a trace if it is realizable.

Recall that there were three properties that must hold for a pattern to be realizable. First, events must be uniquely assigned to a run. Second, if an event of the pattern is preceded in its role description by another event, then this must also be in the pattern. Third, all read events must be enabled. Given a pattern that meets the first two criteria, we explore all possible ways in which it can be refined, until we can either be sure that the refined pattern cannot occur in traces of the protocol, or the refined pattern is a realizable pattern, in which case we know that there are traces in which the pattern occurs.

In manual proofs exploring all refinements corresponds to stating that a message pattern m that is read, must have ended up in the read buffer somehow. Applying case distinction, we would have that it was either put there by a send event of an agent, or by the intruder, after encryption or decryption. The iterative algorithm selects a read event which is not read enabled yet, explores all three options in turn, in each case ensuring that the read event becomes enabled in each refinement.

- In the first case, we assume that the message pattern m occurs first after a send event, we explore all possible role events. In order for m (which is a pattern, and can include variables) to be part of a send message m', it must be possible to unify a part of m' with m, possibly by instantiating variables. Thus, we refine the pattern: the send event is added, variables are instantiated where necessary, and any events that precede the send in the role are also added (in order to meet the first two requirements). After adding the event and an edge labeled with m, the read enabled conditions for the event are met. Note that if read events are added because they precede the send event in the role, these will also have to be read enabled.

  As an example, if we have that $m = \{\!|\, A, r\sharp 1 \,|\!\}_k$, then this might have been sent first after a role event of a run θ that sends $m' = \{\!|\, i, r \,|\!\}_k$. We try to unify m with m'. As a result, $i\sharp\theta = A$ and $r\sharp\theta = r\sharp 1$ and thus $\theta = 1$.

- In the second case, we assume the message pattern m occurs first after an encrypt event, it must be the case that m is of the form $\{\!|\, x \,|\!\}_y$ for some patterns x and y. We unify m with $\{\!|\, x \,|\!\}_y$, and add the encrypt event. The procedure is similar to the role event, except that instead of adding any preceding role events, we must now ensure that the patterns x and y satisfy the read enabled condition.

- In the third case, we assume that the message m occurs first after a decrypt event. Thus, m must be a part of some pattern x. The intruder decrypts a message $\{\!|\, x \,|\!\}_y$ using the key y^{-1}. As a consequence, the message $\{\!|\, x \,|\!\}_y$ as well as the key y^{-1} must be read enabled.

The algorithm iterates until all read events are enabled, in which case the pattern is realizable.

4.2.2 Optimization: decryption chains

The algorithm sketched above will never terminate if there is a single read event that is not enabled. This is caused by the third case, where the option is explored in which the message was decrypted.

Assume the message pattern that is read is m. Then, for the third option, we assume it was decrypted. This introduces two new requirements for the read enabled property to hold. In particular, $\{\!|\, t1 \,|\!\}_{t2}$ and $t2^{-1}$ must be read enabled (where $m \in parts(t1)$). The algorithm iterates, and explores at some point the read enabled cases for $\{\!|\, t1 \,|\!\}_{t2}$. Again, in the third case, it is explored when this message is

the result of a decryption. Thus, there must have been a message $\{\!|\, t3 \,|\!\}_{t4}$ (where $t1 \in parts(t3)$), which must be read enabled, andsoforth. Thus, in this branch of the iteration tree, every iteration results in two new read enabled requirements, which grow in size with each iteration.

The other two cases do not necessarily suffer from this problem. For the encryption event, we have that the two patterns that must be read enabled decrease in size with each iteration, ensuring termination. For the send role events, this depends on the protocol description, and for some protocols the iteration will not terminate. We return to this issue later.

The non-termination issue caused by repeated application of the decryption case can be addressed by limiting the maximum number of levels of encryption occurring in a message. However, this (arbitrary) bound can be avoided. Observe that traces are finite sequences of events. Thus, although there can be arbitrarily many decrypt events leading to the decryption of a message m'' with many levels of encryption, ultimately the message m'' must occur first after a non-decrypt event. This can be exploited by modifying the algorithm for the decrypt event case. Instead of exploring where m occurs first after a read event, we explore all *decryption chains* that might result in m. In other words, we explore the ways in which m can be part of a larger encrypted message m'', and investigate the possibilities for the sending of m'' from other events.

This optimization results in a more complex algorithm that terminates in many cases without the need of a bound on the number of encryptions in messages.

4.2.3 Algorithm preliminaries

The two main mechanisms exploited by the algorithm are case distinction and trace pattern refinement. The algorithm explores all possible refinements of a trace pattern, until it can be concluded that such a refinement either represents a class of traces of the protocol, or represents no traces of the protocol. A given trace pattern typically does not meet the *ReadEnabled* requirement, and we need to add events, or apply suitable substitutions, in order to meet these requirements.

In order to enumerate the possible ways in which a specific read event can be enabled, the read message is traced back to its origin. Given a trace, there must be a unique event after which the message occurs first in the intruder knowledge. We examine the consequences for each type of event (after which a message m can occur first):

- m occurs first after a send event. If this is the case, all events that precede the send event in the role order, must also be part of any realizable trace pattern.
- m occurs first after an encrypt event. In this case, m must be of the form $\{\!|\, x \,|\!\}_y$.
- m occurs first after a decrypt event. In this case m is a part of x, and the decrypt event decrypts $\{\!|\, x \,|\!\}_y$ by applying the key y^{-1}.

If we assume this event is a role event or an encrypt event, there are only a finite number of ways in which this is possible, which we can enumerate and explore in turn. However, no such enumeration is possible for decrypt events: a term t can be the results of decrypting any term t' of which includes t after repeated concatenation and encryption with arbitrary terms. This suggest there are infinitely many options, which we cannot enumerate. We remedy this problem by observing that such an encrypted term t' must originally have been the result of a non-decrypt event. After this initial send of t', there are finitely many decrypt events that ultimately produce t from t'. We call such a sequence that starts with a non-decrypt event, followed by zero or more decrypt events, a decryption chain. The following lemma captures such a sequence.

Lemma 4.24 (Decryption chain). *Let $(TPE, \rightarrow)$ be an explicit trace pattern. Let $e0$ be an event and rt_0 a term, such that $e_0 \in TPE$ and $rt_0 \in in(e_0)$. Then there exist terms $rt_1, rt_2, \ldots, rt_N$ and events $e_1, e_2, \ldots, e_{N+1}$ such that*

$$e_{N+1} \xrightarrow{rt_N} e_N \xrightarrow{rt_{N-1}} \cdots \xrightarrow{rt_1} e_1 \xrightarrow{rt_0} e_0$$

where we have that e_{N+1} is not a decrypt event, and

$$\forall i : 0 < i < N+1 : \exists m, k \in RunTerm : rt_i = \{\!|\, m \,|\!\}_k \ \wedge\ rt_{i-1} \in parts(m)$$

Proof. For an explicit trace pattern, we have that if $rt_0 \in in(e_0)$, there must exist an event e_1 such that $e_1 \xrightarrow{rt_0} e_0$. We apply case distinction. If e_1 is not a decrypt event, we have that $N = 0$, $e_1 = e_0$. If e_1 is a decrypt event of the form $decr(\{\!|\, m \,|\!\}_k, k^{-1}, m)$ for some m, k, we have that $rt_0 \in parts(m)$. Furthermore, we have that $\{\!|\, m \,|\!\}_k \in in(e_1)$, and we iterate for rt_1 with this encrypted term. Because TPE is finite, this iteration terminates. □

We refer to such a sequence of events $e_{N+1}, \ldots, e_0$ that meet the criteria stated in Lemma 4.24, as a decryption chain. Intuitively, the lemma states that rt occurs first after a non-decryption event (in which case we have $N = 0$), or it occurs first after application of N decryptions.

Lemma 4.25 (Explicit trace pattern contains no alternating encrypt/decrypt). *Let m and k be terms. Let $(TPE, \rightarrow)$ be a trace pattern, with events of the form*

$$encr(m, k, \{\!|\, m \,|\!\}_k) \xrightarrow{\{\!|\, m \,|\!\}_k} decr(\{\!|\, m \,|\!\}_k, k^{-1}, m) \xrightarrow{m} e$$

for some terms m and k, and some event e, then we have that $(TPE, \rightarrow)$ is not an explicit trace pattern.

Proof. Assume $(TPE, \rightarrow)$ is explicit. By definition, we have that m is an element of $in(encr(m, k, \{\!|\, m \,|\!\}_k)$. Because $(TPE, \rightarrow)$ is explicit there exists an event e' such that $e' \xrightarrow{m} encr(m, k, \{\!|\, m \,|\!\}_k)$. Observe that we already had $decr(\{\!|\, m \,|\!\}_k, k^{-1}, m) \xrightarrow{m} e$, occurring after the encryption. This contradicts the second property of explicit trace patterns, which states that terms are only learnt once first, as m occurs first after two different events. □

Corollary 4.26 (Decryption chain does not start from an encryption). *Let $(TPE, \rightarrow)$ be an explicit trace pattern. Let $e_{N+1}, \ldots, e_0$ be a decryption chain of $(TPE, \rightarrow)$. If $N > 0$, we have that e_{N+1} is not an encryption event.*

This corollary follows from Lemma 4.25 and Lemma 4.24.

As described in the algorithm sketch, the pattern which needs to be read enabled is unified with the possible candidates. To that end, we define the notion of a most general unifier.

Definition 4.27 (Most general unifier MGU). *Let ϕ be a substitution of uninstantiated variables by run terms. We call ϕ a unifier of term $t1$ and term $t1$ if $\phi(t1) = \phi(t2)$. We call ϕ the most general unifier of two terms $t1, t2$, notation $\phi = MGU(t1, t2)$, if for any other unifier ϕ' there exists a substitution ϕ'', such that $\phi' = \phi \circ \phi''$.*

The above definition assumes variables are typeless. In the remainder, we will assume that MGU is defined in such a way that the type check constraints on variables are met.

We generalize the notion of unification to so-called decryption unification, which captures all the ways in which a term $t1$ can be unified with a (sub)term of another term $t2$ (possibly after repeated decryption and projection operations), and the terms that need to be decrypted in order to extract this unified term from $t2$.

Definition 4.28 (most general decryption unifier $MGDU$). *Let ϕ be a substitution of uninstantiated variables by run terms. We call (ϕ, L) a decryption unifier of a term $t1$ and a term $t2$, notation $(\phi, L) \in DU(t1, t2)$, if either*

- $L = [\,]$ *and* $\phi(t1) \in parts(\phi(t2))$, *or*
- $L = L' \cdot \{\!|\, m \,|\!\}_k$, $\{\!|\, m \,|\!\}_k \in parts(\phi(t2))$, *and* $(\phi, L') \in DU(t1, m)$.

We call a set of decryption unifiers S the most general decryption unifiers of $t1$,$t2$, notation $S = MGDU(t1, t2)$, if and only if

- *for all $(\phi, L) \in S$ we have that $(\phi, L) \in DU(t1, t2)$, and*
- *for any decryption unifier $(\phi, L) \in DU(t1, t2)$, there exists a decryption unifier $(\phi', L') \in MGDU$ and a substitution ϕ'', such that $\phi' = \phi \circ \phi''$.*

The set $MGDU$ captures all ways in which a term $t1$ can be the result of applying (repeated) decryptions and projections to $t2$. Each element of the set consists of a unifying substitution, and a list of terms that need to be decrypted. Each such tuple uniquely defines a decryption sequence that starts with the sending of $t2$, and results in $t1$ becoming known to the intruder.

Example 4.29. *Let V and W be variables, such their sets of allowed substitutions $type(V)$,$type(W)$ contain only basic terms, and thus no tuples or encryptions. Then*

we have that

$$MGDU(ni\sharp 1, (V\sharp 1, \{\!|\, W\sharp 2 \,|\!\}_{pk(r\sharp 2)})) = \Big\{ \big(\{V\sharp 1 \mapsto ni\sharp 1\} \ , [\,]\big), \\ \big(\{W\sharp 2 \mapsto ni\sharp 1\}\, , [\{\!|\, W\sharp 2 \,|\!\}_{pk(r\sharp 2)}]\big) \Big\}$$

Lemma 4.30 (Conditions under which the *MGDU* set is finite). *Given two terms $t1, t2$, such that for each variable V occurring in either term, type(V) contains no tuples and encryptions, the set $MGDU(t1, t2)$ is finite.*

Proof. Let $t1$ and $t2$ be terms. We use the following procedure to compute the set of most general decryption unifiers:

$$MGDU(t1,\, t2) = \\ \big\{(\phi, [\,]) \;\big|\; t' \in parts(t2) \wedge \phi = MGU(t1, t')\big\} \cup \\ \big\{(\phi, L \cdot [\{\!|\, m \,|\!\}_k]) \;\big|\; \{\!|\, m \,|\!\}_k \in parts(t2) \wedge (\phi, L) \in MGDU(t1, m))\big\}$$

The left half of the union is a finite branching of the order of the size of $t2$. The second component is defined recursively. $t2$ includes finitely many applications of the encryption operator, and each subterm contains finitely many parts. As any subsequent substitutions do no change the number of tuples or encryptions, the iteration is guaranteed to terminate. □

If $t2$ contains variables which can be instantiated with tuples or encryptions, the set *MGDU* is not guaranteed to be finite.

4.2.4 Algorithm

In essence, the algorithm takes a trace pattern, and applies case distinction to split the pattern into several refined trace patterns. Each of these patterns are split and refined further until they either (1) represent an explicit and realizable trace pattern, or (2) represent a contradictory pattern. We say a trace pattern is contradictory when it shares no traces with the protocol. We first explain the splitting and refinement part, after which we address the details of contradictory patterns, and discuss termination of the algorithm.

Let P be a protocol, and let $(TPE, \rightarrow)$ be a trace pattern that satisfies both *UniqueRuns* and *RoleConsistent*.

Let S be the set of *in* terms of the events in TPE which have no incoming edge yet:

$$S = \big\{(e, rt) \;\big|\; e \in TPE \wedge rt \in in(e) \wedge \neg\exists e2 : e2 \xrightarrow{rt} e\big\}$$

We call the elements of S the *open goals* of $(TPE, \rightarrow)$. If $(TPE, \rightarrow)$ is extended with an edge such that a goal that was previously open is removed from S, we refer to this process as *goal binding*.

If $S = \emptyset$, there are no open goals, and the trace pattern $(TPE, \rightarrow)$ satisfies all three properties required for an explicit trace pattern. As a consequence it is realizable.

It represents a non-empty set of traces, and we have

$$S = \emptyset \;\Rightarrow\; \Psi(P, (TPE, \rightarrow)) = (TPE, \rightarrow)$$

If S is not-empty, there are open goals, representing terms that are required to enable some events, of which we don't know yet the point at which they first occur in the intruder knowledge. We select a single open goal from S using a heuristic. This heuristic will be explained in detail in Section 4.4.4. For now, we just assume a single element (e_0, rt_0) is selected.

We proceed by refining $(TPE, \rightarrow)$ by applying case distinction. Based on Lemma 4.24, we have that any refinement of $(TPE, \rightarrow)$ must contain a decryption chain $e_{N+1}, \ldots, e_0$ yielding the term rt_0, which is an open goal of the event e_0. The existence of such a chain is the basis for case distinction. We outline the possible cases before addressing them in detail:

- Case 1: $\{e_{N+1}, \ldots, e_1\} \cap TPE \neq \emptyset$, covered by $\Psi_{(1)}$.
- Case 2: $\{e_{N+1}, \ldots, e_1\} \cap TPE = \emptyset$, which is further split by type of event:
  - Case 2.1: e_{N+1} is a decrypt event, covered by $\Psi_{(2.1)}$.
  - Case 2.2: e_{N+1} is an encrypt event, covered by $\Psi_{(2.2)}$.
  - Case 2.3: e_{N+1} is an instantiated role event, covered by $\Psi_{(2.3)}$.

The output of the algorithm is the union of the output of all these subcases. We explain the details of the cases below.

The main case distinction is based on whether the events of the decryption chain $e_{N+1}, \ldots, e_0$ either coincide with some events in already in TPE (case 1), or not (case 2).

Case 1: $\{e_{N+1}, \ldots, e_1\} \cap TPE \neq \emptyset$.

Because the decryption chain shares events with TPE, there exists a unique k such that $e_k \in TPE$ and $\forall i : 1 \leq i < k : e_i \notin TPE$. We explore each option: for each event $e' \in TPE$, we determine how e_0 can be the result of repeated decryptions of the output terms, captured by the set $MGDU(e_0, t)$ for each $t \in out(e')$. For each possible option (ϕ, L) of an event e', we construct a refined trace pattern $(TPE, \rightarrow)'$ as follows

- If $L = \emptyset$, we add $e' \xrightarrow{rt_0} e_0$.
- If $L \neq \emptyset$, we add a decrypt event for each $\{\!|\, m \,|\!\}_k$ in the list L. We add labeled edges, as prescribed by the decryption chain, connecting the chain of events. As a side result, any required decryption keys will be open goals of $(TPE, \rightarrow)'$.
- In the end, we apply the substitution ϕ to the resulting trace pattern.

More formally, we define

$$\Psi_{(1)}(P,(TPE,\rightarrow)) = \bigcup_{\substack{e\in TPE,\\ t\in out(e),\\ (\phi,L)\in MGDU(rt_0,t)}} \phi(chain((TPE,\rightarrow),e,L,e_0,rt_0))$$

where *chain* is a function that refines a trace pattern. Given a trace pattern, an event rt_s from which the chain starts, a list of terms that need to be decrypted, and an event e_g and a term rt_g representing an unbound goal, it adds the events and edges that are needed to establish a decryption chain. For decryption chains of length 0 it is defined as

$$chain((TPE,\rightarrow),e_s,[\,],e_g,rt_g) = (TPE,\rightarrow \cup \{e_s \xrightarrow{rt_g} e_g\})$$

and for all other decryption chains it is recursively defined as

$$chain((TPE,\rightarrow),e_s,\{\!|\, m \,|\!\}_k \cdot L,e_g,rt_g) = chain((TPE\cup\{e'\},\rightarrow\cup\{e_s \xrightarrow{\{\!|\, m \,|\!\}_k} e'\}),e',L,e_g,rt_g)$$

where $e' = decr(\{\!|\, m \,|\!\}_k, k^{-1}, m)$.

Case 2: $\{e_{N+1},\ldots,e_1\} \cap TPE = \emptyset$.

We split cases on the type of e_{N+1}.

Case 2.1: e_{N+1} is a decrypt event.

This cannot be the case, because by construction, a decryption chain does not start with a decrypt event, and thus there exist no efficient trace patterns that are refinements of $(TPE,\rightarrow)$ in this case.

$$\Psi_{(2.1)}(P,(TPE,\rightarrow)) = \emptyset$$

Case 2.2: e_{N+1} is an encrypt event.

In this case we have that $N = 0$, as a result of Corollary 4.26. Because an encrypt event has only a single element in its *out* set, it must be the case that $rt_0 = \{\!|\, m \,|\!\}_k$ for some terms m, k. Thus, we add an encrypt event $encr(m,k,\{\!|\, m \,|\!\}_k)$ to TPE, and add an edge $encr(m,k,\{\!|\, m \,|\!\}_k) \xrightarrow{rt_0} e_0$, and iterate the algorithm. As a side result, two open goals are added: the components m and k from which rt_0 is constructed.

Thus, if there exist m, k such that $\{\!|\, m \,|\!\}_k = rt_0$, we have that

$$\Psi_{(2.2)}(P,(TPE,\rightarrow)) = \Psi(P,(TPE\cup\{e'\},\rightarrow\cup(e' \xrightarrow{rt_0} e_0)))$$

where $e' = encr(m,k,\{\!|\, m \,|\!\}_k)$.

If rt_0 is not an encrypted term, it cannot be the result of an encrypt event, and we have that

$$\Psi_{(2.2)}(P,(TPE,\rightarrow)) = \emptyset$$

Case 2.3: e_{N+1} is an instantiated role event.

We apply case distinction on the type of role event to which e_{N+1} corresponds, and whether the run identifier of e_{N+1} does not occur in *TPE* (case 2.3.1) or whether it already occurs (case 2.3.2). Thus, we have that

$$\Psi_{(2.3)}(P, TP) = \bigcup_{\substack{r \in Role, \\ \varepsilon \in P(r)}} \Psi_{(2.3.1,r,\varepsilon)}(P, TP) \cup \Psi_{(2.3.2,r,\varepsilon)}(P, TP)$$

In all cases, we refine the trace pattern by adding an instantiation of the role event.

Case 2.3.1: The run identifier of e_{N+1} does not occur in *TPE*.

Because trace pattern interpretation (4.6) abstracts away from the actual choice of run identifiers, the particular choice of the run identifier for e_{N+1} is irrelevant, as long as it does not occur in *TPE* yet, as the set of traces of each choice are equivalent. Let θ be a run identifier that does not occur in *TPE*, and let $inst = (\theta, \emptyset, \emptyset)$.

For all explicit trace patterns, we require that *RoleConsistent* holds. Thus, if some event occurs in a trace pattern, it must be the case that all events that precede it in the role description, are also part of *TPE*. Let *pev* be the set of events that includes the instantiation of ε and the events that precede ε according to the role definitions:

$$pev = \{\langle inst \rangle(\varepsilon') \mid \varepsilon' \prec_r \varepsilon\}$$

Let *por* be the set of edges that defines the ordering on these events (as required by the role definitions):

$$por = \bigcup_{\substack{e1, e2 \in pev \\ e1 \prec_r e2}} \{e1 \rightarrow e2\}$$

Using these abbreviations, we can define the details of the algorithm for this case:

$$\Psi_{(2.3.1,r,\varepsilon)}(P, (TPE, \rightarrow)) = \bigcup_{\substack{t \in out(\langle inst \rangle(\varepsilon)), \\ (\phi, L) \in MGDU(rt_0, t)}} \Psi\Big(P, \phi\big(chain((TPE \cup pev, \rightarrow \cup por), \langle inst \rangle(\varepsilon), L, e_0, rt_0)\big)\Big)$$

Case 2.3.2: The run identifier of e_{N+1} occurs in *TPE*.

Let *RID* be the set of run identifiers occurring in *TPE*, such that each run θ is executing role r, but no instantiation of the role event ε occurs yet in *TPE* for this run. As events of each of these run identifiers already occur in *TPE*, some variables might already be instantiated, which would put constraints on the instantiation of the event ε. For each $\theta \in RID$, we define $inst_\theta = (\theta, \rho, \sigma)$, where ρ and σ are the smallest substitutions (on role names and variable names, respectively) such that $\forall e \in TPE : runidof(e) = \theta \Rightarrow e = \langle inst_\theta \rangle(roleevent(e))$. For each θ and

corresponding $inst_\theta$, we define pev_θ and por_θ as in 2.3.1.

$$\Psi_{(2.3.1,r,\varepsilon)}(P,(TPE,\rightarrow)) = \bigcup_{\substack{\theta\in RID,\\ t\in out(\langle inst_\theta\rangle(\varepsilon)),\\ (\phi,L)\in MGDU(rt_0,t)}} \Psi\Big(P,\phi\big(chain((TPE\cup pev_\theta,\rightarrow\cup por_\theta),\langle inst_\theta\rangle(\varepsilon),L,e_0,rt_0)\big)\Big)$$

Contradictory trace patterns In the process of refining trace patterns, we may refine them in such a way that we can immediately conclude that the trace pattern can never contain any traces that are traces of the protocol.

In particular, if the relation $\rightarrow$ contains a cycle, there can be no linear trace that satisfies the order. As a second criterion, we observe that if constants of a run θ occur in events that precede the first send event of the run θ, there can be no trace in the trace pattern that is also trace of the protocol.

Results If the algorithm terminates, the result is a set of refined patterns. The results are sound, in the sense that the traces in each resulting trace pattern are both traces of the original trace pattern (as only refinement was applied), and also traces of the protocol (as the trace patterns are explicit). With respect to completeness, the method is complete if there are no variables that can contain tuples or encryptions.

4.2.5 Termination

In general, the algorithm is not guaranteed to terminate. This is in line with the undecidability results for security protocol analysis, as e. g. in [81]. In order to guarantee termination, we introduce a bound *maxruns* on the number of runs represented by the events in *TPE*. All states where the set of trace events represent more than *maxruns* runs, are pruned from the search space.

Definition 4.31 (Bounded characterization algorithm). *We improve the algorithm Ψ by introducing a parameter maxruns, and by changing the algorithm in two ways. First, we consider all trace patterns that contain more than maxruns run identifiers to be contradictory. Second, we add an extra output boolean that is true if and only if a trace pattern is encountered (and deemed contradictory) where the number of runs exceeds maxruns. The resulting bounded algorithm $\Psi_{maxruns}$ has signature*

$$\Psi_{maxruns} : Protocol \times Pattern \rightarrow \mathcal{P}(Pattern) \times Bool$$

Lemma 4.32 (Bound on the number of runs and termination). *The bounded version of the algorithm terminates.*

Proof. We associate each trace pattern TP with a tuple rnk, which is defined as

$$rnk(TP) = (maxruns - runs(TP), enc(TP), og(TP))$$

where $runs(TP)$ is the number of runs represented by the events in TP, $enc(TP)$ is the number of encryptions occurring in open goals, and $og(TP)$ is the number of open goals. On these tuples, we assume a lexicographic ordering. In each iteration, there are a finite number of branches, and for each branch rnk decreases. Furthermore we have that $rnk \geq (0, 0, 0)$, and thus termination is guaranteed. □

Of course, introducing a bound implies that some states might be pruned, and thus that the characterization is no longer complete: some behaviours of the trace pattern might be missed. However, the introduction of the *incomplete* boolean will signal this case.

Theorem 4.33 (Bound on the number of runs and completeness). *For all protocols P, trace patterns $(TPE, \rightarrow)$, bound on the number of runs $maxruns$, and sets of explicit trace patterns ES, we have that*

$$\big(\Psi_{maxruns}(P, (TPE, \rightarrow)) = (ES, \mathit{False})\big) \;\Rightarrow\; \Psi(P, (TPE, \rightarrow)) = ES$$

The theorem states that if after termination of the bounded version of the algorithm, no patterns have been deemed contradictory on the basis of having surpassed the maximum number of runs, the resulting realizable trace patterns represent a complete characterization of the trace pattern. They capture all possible behaviours of the trace pattern.

Proof. If no trace patterns are pruned, the bounded algorithm behaves identical to the unbounded algorithm. □

This result allows the tool to verify protocols for an unbounded number of runs in many cases, even though it is guaranteed to terminate. In practice, unbounded verification results occur for about 90 percent of the protocols, as we will see in Section 4.4.5.

Note that we can have additional rules for pruning trace patterns, as long as they are monotonous for each refinement step. For example, our prototype tool includes rules that limit e. g. trace length, or whether agents are allowed to initiate sessions with themselves. Again, if patterns are pruned because of such rules, this is signalled through the boolean output.

4.3 Verification of security properties by means of characterization

Given a protocol and trace pattern, the algorithm sketched above yields a complete characterization of a protocol: a set of explicit trace pattern representations, which together capture all possible ways in which the trace pattern can occur in a trace. This does not immediately imply we can verify security properties using this method.

4.3.1 Verifying secrecy properties

Secrecy properties are defined such that for each claim that occurs in a trace, such that all its communication partners are trusted, some term must never be in the intruder knowledge. We aim to construct a trace pattern that captures all possible attacks on such a claim. This requires that the trace pattern is defined in such a way that (1) it includes an instance of the secrecy claim, which claims that some term t is secret, (2) all agents the run communicates with are trusted, and (3) the intruder learns the term t.

Regarding (1), we define a trace pattern that includes all events of a run θ of the role, up to and including the claim.

For (2), enforcing that the agents of the run are trusted, we exploit the mechanism used for type-checking variables, defined in the $type(V)$ function, to define similar constraints for the role names of the run with identifier θ, by defining e. g. $type(i\sharp\theta) = type(r\sharp\theta) = Agent_T$.

With respect to (3), we observe that is not immediately possible to express statements about the intruder knowledge in terms of trace patterns. Rather, the intruder knowledge is encoded implicitly in the events. We extend the set of events with events of the form $e = intruderKnows(rt)$, with $in(e) = parts(rt)$ and $out(e) = \emptyset$. Thus, the set of traces violating the secrecy claim is captured exactly to the trace pattern that contains two events: a symbolic instance of the claim with secret term rt, and an event $intruderKnows(rt)$. If we apply the algorithm to this trace set, and it turns out the complete characterization has no elements, we have that the claim holds. If the complete characterization is non-empty, each trace pattern it contains defines traces that violate the secrecy property.[2]

We apply the algorithm Ψ to the constructed trace pattern. If the set of explicit trace patterns is non-empty, each trace of these patterns constitutes an attack on the secrecy claim, and we can use the procedure used in the proof of Theorem 4.17 to establish an attack trace. If it is empty, no attack exists.

4.3.2 Verifying authentication properties

As we have seen in the previous chapter, authentication properties are formalized as the fact that for each instance of a claim in a trace, there must exist certain other events in the trace. The requirements on these other events vary according to the specific authentication property.

Given the complete characterization of a run of this role (up to the claim event, including trustedness constraints on the type of the role substitutions as we did with secrecy claims), we can verify whether each explicit trace pattern satisfies the required property. If it does not, we can use the procedure used in the proof

[2]Observe that such an "*intruderKnows*" event would behave exactly as a read event. We therefore refrain from introducing a new event, and encode such intruder knowledge constraints as read events.

of Theorem 4.17 to establish an attack trace. If the property holds for the trace pattern, it holds for all traces defined for the pattern.

4.3.3 Correctness and complete characterizations

Intuitively one might expect that a correct protocol allows for its expected behaviour and nothing else. This thought is also captured by the definition of synchronisation. Thus, one would expect that a correct protocol has few different behaviours (and thus only a few explicit trace patterns that include a claim), whereas an incorrect protocol has more behaviours, some of which were not foreseen by the protocol designers. Thus, we would expect that for a correct protocol, the complete characterization set of a claim contains a single element.

For our test set of 128 protocols, which includes the vast majority of the protocols in the SPORE library [174], there are some striking correlations between the correctness of a protocol, and the number of explicit trace patterns that are output by the algorithm.

All protocols in the SPORE library, with the exception of the KSL protocol that have exactly one realizable trace pattern for each role, are correct with respect to synchronisation.

The converse also holds for the test set. In fact, all correct protocols in the set that satisfy agreement or synchronisation, have exactly one explicit trace pattern for each role. We conjecture that for a role that satisfies agreement as defined in the previous chapter, there exists only one realizable trace pattern. Proving the conjecture is left as future work.

4.4 Prototype implementation: Scyther

We have implemented the algorithm sketched above in a prototype called *Scyther*. In this section we briefly discuss some aspects of the prototype.

4.4.1 Requirements and design

The main requirement of the prototype tool is to enable verification of security properties within the operational semantics as developed in the previous chapter. Consequently, we have that

- The input language closely follows the syntax used for describing protocols. In Figure 4.4 on the following page we show the input for the tool used to describe the initiator role of the Needham-Schroeder protocol.
- The semantics of the tool correspond to the model developed in the previous two chapters.

- Security properties are expressed as claims in the protocol descriptions, and translated by the tool into appropriate trace patterns and related tests.

In addition, the tool can provide complete characterizations of claim events.

```
role I
{
    const ni: Nonce;
    var nr: Nonce;

    send_1(I,R, { I,ni }pk(R) );
    read_2(R,I, { ni,nr }pk(I) );
    send_3(I,R, { nr }pk(R) );

    claim_i1(I, Secret, ni);
    claim_i2(I, Secret, nr);
    claim_i3(I, Nisynch);
}
```

Figure 4.4: Syntax for describing the initiator role of the Needham-Schroeder protocol

Regarding the design, we have opted to split the tool into two main components. The core of the Scyther toolset is the Scyther command-line tool, which incorporates the characterization and verification algorithms. This stand-alone tool is sufficient to perform protocol analysis.

A second component has been developed for user convenience, and comprises an optional graphical user interface for the Scyther command-line tool. This component acts as a wrapper for the command-line tool, and provides a protocol editor, and graphical output of the attacks or trace patterns.

4.4.2 Implementation

The current implementation is available for both Linux and Windows platforms, and can be ported with little effort to other platforms.

Command-line tool

The command-line tool is written in the C language, and is optimized for verification speed.

The tool takes as input a protocol description, and optional parameters (such as bounds, or matching functions), and outputs a summary report, and optionally, representations of trace patterns in XML or visual graph descriptions in the dot language.[3] In Figure 4.5 on the next page we show the input and output formats of the tool, where a dotted line denotes an optional feature.

[3]Graph descriptions in the dot language can be used to generate graph pictures using the

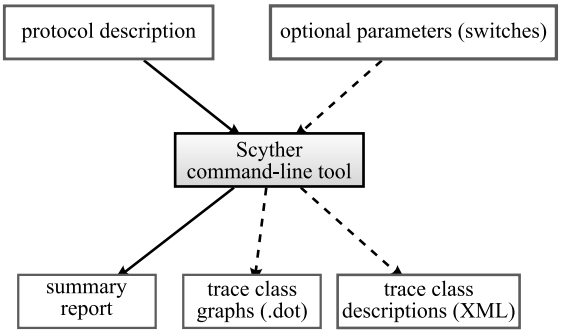

Figure 4.5: Scyther : Command-line tool

We list some features of the tool:

- By default the tool returns a summary of the claim's status; whether a claim is true, false, or whether there is no attack within the bounds. Optionally, it can output explicit trace patterns (which represent attacks or characterizations). For the output there are two formats available:
  - Graph output (in the dot language from the Graphviz project)
  - XML output (describing the trace pattern)
- By default, the tool yields at most one attack on each claim. If multiple explicit trace patterns are found in which a claim is violated, a heuristic is applied to select the *most feasible* attack (related to the experimental results from [102]). To give a simple example: If there are two patterns, but in one pattern an agent A starts a run in which she tries to communicate with herself, the other pattern is used to generate an attack. Optionally, the tool can generate all possible attacks.

Furthermore, a set of Python bindings exist for the Scyther command-line tool, which provide a convenient means to script Scyther experiments. As an example, an example Python program is provided for large scale multi-protocol analysis experiments, as described in the next chapter.

Graphical user interface

For the graphical user interface, speed is not the biggest concern. The interface is written in the Python language, combined with the wxPython interface library. The user interface adds no essential features to the command-line tool other than improved usability.

In Figure 4.6 on the following page we show the input and output formats of the graphical user interface, and the interaction with the command-line tool.

GraphViz package.

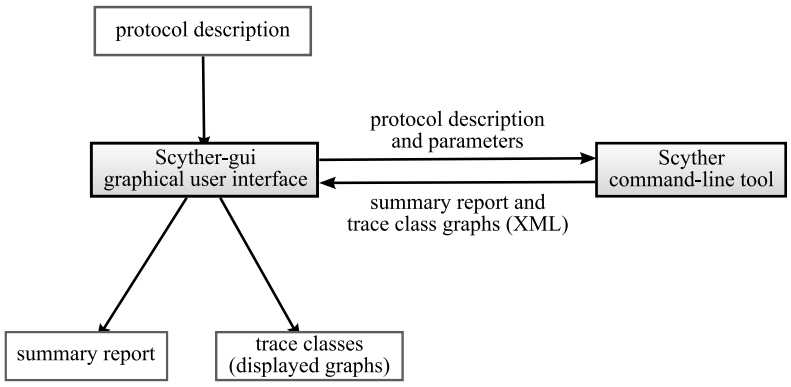

Figure 4.6: Scyther-gui : Graphical interface for Scyther

4.4.3 Validation

Introducing a new security protocol semantics and corresponding verification procedure requires some form of validation. For this, it does not suffice to show that the tool produces the expected result for a single protocol. In this section we report some of the cases to which the implementation has been successfully applied. In all cases, the tool produced the expected result for protocols known to be correct. In the cases where the tool found attacks, inspection of the output revealed that the explicit trace pattern indeed violated the property under investigation.

Over the last three years, we have modeled over a hundred protocols in the Scyther input language. This set includes the majority of the protocols in the SPORE library from [174] (except those that use Diffie-Hellman exponentiation or the exclusive-or operator). The prototype is able to find all known attacks on the properties from Chapter 3 in less than a second.

During the last two years, Scyther has been used for teaching the course "Provable correctness of security protocols" at the Eindhoven University of Technology, which is part of the curriculum for MSC students partaking in the Information Security programme. Furthermore, it has been used for the course "Formal Methods in the Software Life Cycle" taught at the Radboud University Nijmegen and the Eindhoven University of Technology.

The XML output of the Scyther tool has been used for experiments regarding the systematic analysis of attacks. A program was developed for the automated classification of attacks. A partial report can be found in [102].

Using Scyther, we have discovered previously unknown multi-protocol attacks, which we describe in the next chapter. We have discovered previously unknown attacks on the synchronisation of two multi-party authentication protocols, which we describe in Chapter 6, and the tool has been used to discard candidates for the multi-party authentication protocols developed in the same chapter. Scyther has also been used to verify theoretical results regarding protocol composition in cooperation with the

Norwegian University of Science and Technology (NTNU) in Trondheim [6].

As the above examples indicate, Scyther has been successfully used for the verification and design of protocols, as well as for supporting theoretical research.

We proceed by fine-tuning two parameters of the algorithm. First, we address the choice of the goal selection heuristic. Next, we discuss the choice for the bound on the number of runs.

4.4.4 Choosing a heuristic

The algorithm sketched above includes an important heuristic. Given that all read events must be enabled, we must ensure that there are no open goals. An open goal is a tuple (e, rt), consisting of a read event e and a non-tuple term rt, from which one is selected for case distinction and pattern refinement. Although the algorithm will try to bind any other open goals in further iterations, any substitutions made by the case distinctions and refinement steps influence the branching factors further on. Furthermore, contradictory states may occur earlier depending on the choices made by the heuristic.

Thus, the heuristic can influence the number of states traversed, but also the maximum size of the set TPE at which contradictions are found. This means the heuristic is important not only for the speed of the verification, but also for improving the number of cases in which verification is complete when the algorithm is invoked with a bound. A similar heuristic must exist in [173], however it is not explained in detail by the authors of the Athena method.

We have devised over 20 candidate heuristics and investigated their effectiveness. Here we report our main findings and illustrate them by means of a few selected heuristics, ordered according to their effectiveness.

- *Heuristic 1: Random.* An open goal is selected randomly for case splitting.
- *Heuristic 2: Constants.* For each open goal term rt, the number of local constants that are a subterm of rt, is divided by the number of basic terms that are a subterm of rt. The goal with the highest ratio is selected.
- *Heuristic 3:* Open goals that correspond to the keys needed for decrypt events are given higher priority, unless these keys are in the initial intruder knowledge.
- *Heuristic 4:* Give priority to goals with that contain a private key as a subterm; next, give priority to goals that contain a public key ; all other terms have lower priority.
- *Heuristic 5:* A combination of heuristics 2, 3 and 4, where first heuristic 4 is applied. If this yields equal priorities for a goal, heuristics 2 and 3 are applied.

Regarding heuristic 4, we observe that the semantics do not explicitly mention such concepts as 'private' or 'public' key (there might be no such terms, or multiple

key infrastructures). We derive these from the initial intruder knowledge and role descriptions by identifying function names which are never sent as a subterm, but only as keys. This will typically include sk and pk. Second, we observe that for some functions, all applications are in the initial intruder knowledge, usually the public keys such as pk, whereas for others only a strict subset of the domain is part of M_0, usually the private keys sk.

For all heuristics, we have that if two open goals are assigned the same priority value, the open goal that was added first is selected. Tests have shown this to be slightly more effective for all heuristics involved.

The first heuristic acts as a reference point for establishing relative effectiveness of each heuristic. The second heuristic corresponds to the intuition that terms which contain more local constants of particular runs, can only be bound to very particular send events (as opposed to terms with many globals or variables), resulting in less case distinctions. We believe a similar heuristic was used in a version of the Athena tool. The third heuristic captures the intuition that there should be few ways in which the intruder can gain access to a decryption key, as in general keys should not be known to the intruder. (Unless it concerns signatures, in which case the decryption key is the public key, which is part of the initial intruder knowledge.) For the fourth heuristic, a strict priority is given to cases where e. g. the intruder decrypts something with a key that is never sent by the regular agents, usually corresponding to long-term keys, as these branches often lead to contradictory states. Finally, the fifth heuristic is a combination of the previous three heuristics, using a lexicographic order. For the fifth heuristic various weighting functions were also considered, of which the lexicographical order performed best in general.

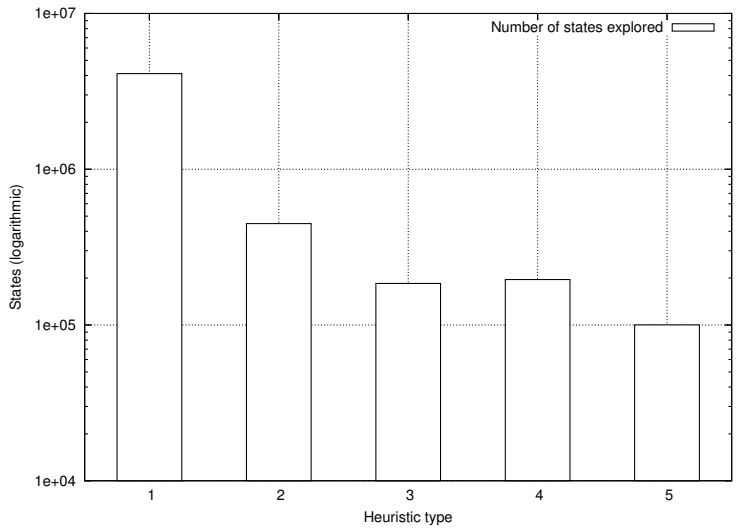

Figure 4.7: The impact of the heuristics on the number of states traversed (for 518 claims, strict bound)

Given a fairly strict bound of four runs, we investigated how each heuristic performed, when applied to a test set of 128 protocol descriptions, with 518 claims. Our test set includes the vast majority of the protocols in the SPORE library [174], various protocols from (un)published papers, variations on existing protocols, and new protocols, modeled by users of the Scyther tool throughout the last two years. A time limit was set for the iteration procedure, which was only used to abort tests for the first two heuristics. In Figure 4.7 on the preceding page we show the impact of the heuristics on the number of states explored. From the graph it is clear that heuristic 5 explores almost 40 times less states than the random heuristic 1. Intuitively, this corresponds to avoiding unnecessary branching, and a tendency to arrive at contradictory trace patterns in less iterations.

Because the effectiveness of the heuristics depends to a large degree on the particular protocol under investigation, it is difficult to give an analytical explanation of the results for the complete test set. However, it seems that the heuristics 2, 3 and 4 can be used to support each other, as is shown by the performance of heuristic 5.

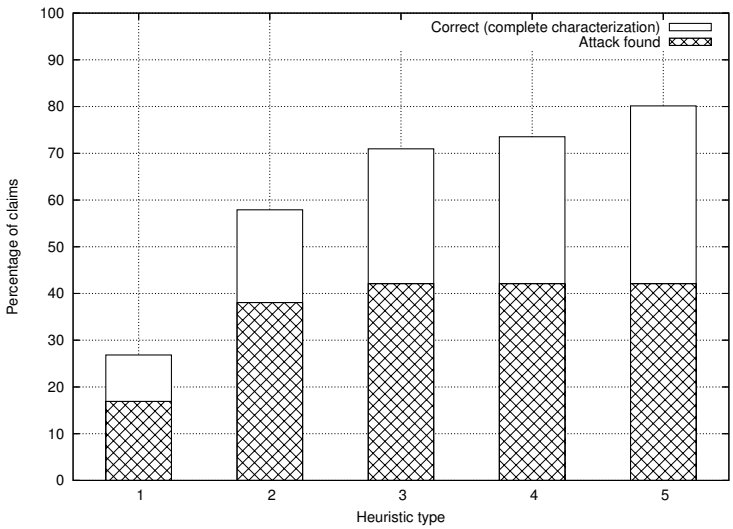

Figure 4.8: The impact of the heuristics on the decidability (for 518 claims, strict bound)

This also has a direct result on the completeness of the results, which is depicted in Figure 4.8. For heuristic 1, we get a complete result (based on complete characterization) for less than 30 percent of the claims. This improves for each heuristic, leading to an 82 percent rating for heuristic 5. In other words, if we use heuristic 5, we have that for 82 percent of the claims, the algorithm is able to either find an attack, or verify correctness for an unbounded number of runs. In the remaining 18 percent of the cases the algorithm determines that there are no attacks involving four runs or less, but it might be possible that there are attacks involving five or more runs.

Clearly, heuristic 5 is to be preferred from the set of investigated heuristics. This leaves the question of choosing an appropriate bound on the number of runs.

4.4.5 Choosing a bound on the number of runs

For the protocols analyzed for this thesis, we did not find any attacks that involved more than $x+1$ runs, where x is the number of roles of the protocol, except for the $f^N g^N$ family of protocols from [139]. The exceptions involve a family of protocols that was specifically tailored to be correct for N runs, but incorrect for $N+1$ runs.[4] This indicates that for practical purposes, initial verification with three or four runs would be sufficient. If verification with a low bound yields no attacks, but neither a complete characterization, the bound can be increased.

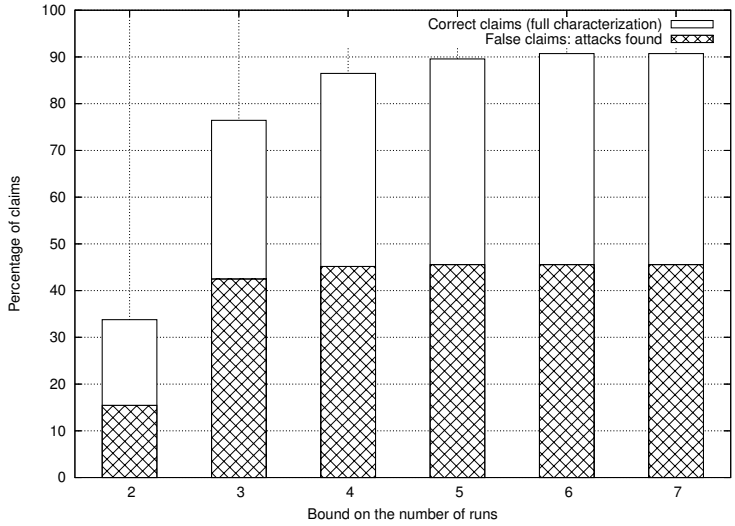

Figure 4.9: The impact of the bound on runs on decidability for the protocols in the test set.

Because a higher bound on the number of runs can improve the rate of complete characterization, but also increases verification time, there is an inherent trade-off between completeness and verification time. We have investigated the impact of the bound on the number of complete characterizations, within the set of protocols we used for the previous graphs. In Figure 4.9 we show the decidability results on the test set as a function of the bound on the runs, using heuristic 5, and using no time limit for the tests. There is no difference between the decidability results of 6 and 7 run bounds, but in general, the higher the bound, the more claims are decided.

[4] This seems to suggest a correlation between the number of roles in the protocol and the runs involved in the attacks. In general, the undecidability of the problem [81] implies that there is no such bound for all protocols, but maybe it is possible to establish a tight lower bound for decidable subclasses [183].

In the test case, no further attacks are found for bounds of three runs or more, but some additional claims can be proven to be correct for an unbounded number of results.

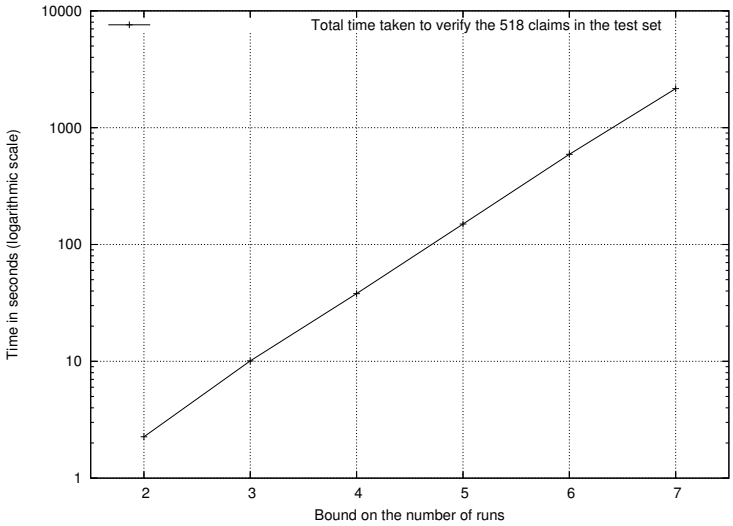

Figure 4.10: The impact of the bound on runs on the total verification time for 518 claims

The drawback of a high bound on the number of runs is time. As the algorithm employs a depth-first search, memory usage is linear in the number of runs, but verification time is exponential. For the test set, we show verification times in Figure 4.10 for some specific bounds on the number of runs. The figure thus corresponds to the time it took to generate the results in Figure 4.9, on a desktop computer with an AMD 3000+ Sempron CPU running at 1.6 GHz, with 1GB of RAM.[5] It is clear that the total verification time of a large set of protocols is exponential with respect to the maximum number of runs, even though verification time is constant for the vast majority of the protocols. This is in line with results such as those in [162].

The protocols in our test set did not include attacks involving more than a handful of runs. To evaluate the effectiveness of Scyther in finding large attacks, we analysed instances of the $f^N g^N$ family of protocols from [139]. These protocols were specifically designed to show theoretical possibilities of the size of attacks. For each $N > 2$, the protocol $f^N g^N$ has no attacks with N runs or less, but there exists an attack involving $N+1$ runs. The protocol contains only two roles and four messages, and the parameter N mainly influences the message size (by increasing the number of nonces and variables). For this protocol, Scyther yields the expected results: for protocols $f^N g^N$ with a bound $maxruns \leq N$, bounded verification is performed. With a bound $maxruns > N$, an attack is found. As an extreme example, we find

[5]Note that because the algorithm uses an iterative depth-first search, it uses a negligible amount of RAM.

the attack that involves 51 runs (on the $f^{50}g^{50}$ protocol) in 137 seconds.

4.5 Conclusions

The Scyther tool has proven to be an effective tool for the verification of security properties. It combines the possibility of verification (proving the protocol correct within the model) and falsification (finding a concrete attack) whilst still being guaranteed to terminate. It is the only currently existing tool capable of verifying synchronisation. It has been extensively used by students and researchers, resulting in the discovery of many previously unknown attacks.

We investigated several options for fine-tuning the algorithm, by e. g. evaluation of different heuristics, and by analyzing the effect of the choice of the bound on the verification result (percentage of protocols verified or falsified) and verification time.

Differences between Athena and Scyther The ideas behind the verification method stem from the Athena tool as described in [171, 173]. Unfortunately, the Athena tool is not publicly available, making it impossible to compare the tools in any meaningful way. Due to changes in the theoretical foundations, we improve on the basic Athena algorithm described in a number of ways:

- Our algorithm is guaranteed to terminate whilst still giving complete characterizations (and thus allowing for deciding security properties for an unbounded number of sessions) for the vast majority of protocols.
- We extend the method to our operational semantics, allowing for e.g. multiple key structures. As a side result, security properties are defined as claim events, and there is no need to set up complex (and thus error-prone) scenarios for the verification of properties. We return to this in Chapter 7.
- We revise the algorithm to yield complete characterizations. This allows us to verify a larger class of security properties, including ordering-based security properties such as synchronisation.
- We make the heuristic in the algorithm explicit and analyse its impact on the verification process.

In fact, our version of the algorithm differs significantly from the algorithm sketched in [173]. In particular, there seems to be an important problem which is most obvious in Definition 4.6 and 4.7 of the journal publication [173] (note that the same problem exists in [171], and implicitly in [172]). According to Definition 4.6 of that paper, unification is defined as *interm unification*, roughly corresponding to the notion that it is possible to interm-unify a term $t1$ not only with $t1$ but also with a tuple $(t1, t2)$. However, for this relation there exists no most general interm unifier if variables are allowed to be instantiated with tuples. In fact, there exists no most general interm unifier, but rather an infinite set of incomparable

unifiers for this relation. As an example, consider the interm unification of a term A with a variable V. We have that A interm-unifies with A, as well as with (V', A), but also with $((V', A), V'')$, etc.. As the Athena algorithm is based on the Strand Spaces approach, which models the intruder actions as protocol events, it is strictly required that variables can be instantiated with tuples, in order to correctly model encryption and decryption as protocol events. Consequently, the set $U_P(t)$ defined below Definition 4.7 in the same paper cannot be guaranteed to be finite. As a result, the next state function $\mathcal{F}$ is not complete-inclusive, which is required for the completeness of the method. The upshot of this is that attacks might be missed by the described algorithm. One of the authors of the Athena paper seems to be aware of this problem, as described in a technical report [26], where a possible fix is suggested by using so-called *interm constraints*. However, the suggested solution is described as being "possibly undecidable".

The problem of instantiating variables with tuples can be avoided in our version of the algorithm, as we do not model intruder events as protocol actions. The additional events *decr* and *encr* are only constructed on-demand by the refinement process, and thus we can restrict ourselves to variables that are not instantiated as tuples.

5

Multi-Protocol Attacks

In this chapter we will apply some of the methodology and tools developed in the previous chapters. We turn our attention to one of the assumptions underlying modern security protocol analysis, namely that the protocol under scrutiny is the only protocol that is executed in the system.

As indicated in the previous chapters, a number of successful formal methods and corresponding tools have been developed to analyse security protocols in recent years. These methods are generally limited to verification of protocols that run in isolation: for a protocol that is used over an untrusted network, the formal models generally assume that there is only one protocol being executed over the network.

However, the assumption that a protocol is the only protocol that is executed over the untrusted network is not realistic. Unfortunately, when multiple protocols are used over a shared untrusted network, the problem of verifying security properties becomes significantly harder. The cause of this is the fact that security properties are not compositional. It may happen that two protocols that are correct when run in isolation, are vulnerable to new attacks when used over the same network.

An attack that necessarily involves more than one protocol, is called a *multi-protocol attack*. The existence of such attacks was established first by Kelsey, Schneier and Wagner (see [111]). They devise a procedure that starts from any correct protocol. Then they show that it is possible to construct a specially tailored protocol such that (1) the second protocol is correct, and (2) when these two protocols are executed over the same network, the intruder can use messages from the second protocol to mount an attack against the first protocol. Some specific examples can be found in literature, e. g. [5, 184]. Because the protocols in these papers are constructed especially for the purpose of breaking some particular protocol, they seem contrived.

At the other end of the spectrum, sufficient conditions for compositionality have been established by e.g. Guttman and Thayer in [95]. Alternative requirements can be found in [93, 48, 47]. If *all* protocols that use the same network and key infrastructure satisfy certain requirements, e.g. messages of one protocol can never be mistaken for messages from the other protocols, compositionality of the individual security properties is guaranteed. In that case, in order to prove correctness of the system it suffices to prove correctness of the protocols in isolation. However, the standard protocols found in literature do not meet these requirements, as many of these protocols have very similar messages, often containing permutations of a list of

nonces and a list of agent names. Thus the theoretical possibility of multi-protocol attacks remains.

A third approach to this compositionality problem are methods that aim to establish that a given set of protocols does not interfere. A recent example of such an approach is e.g. the invariant based approach in [71].

The question that is answered in this chapter is: Are multi-protocol attacks a realistic threat which we should consider when using protocols from the literature? This question has not been addressed before, because verification methods for security protocols traditionally do not scale well. This holds for manual methods (e.g. proofs by hand) as well as (semi-)automatic methods. It is feasible to verify small to medium-sized protocols in isolation, but large and/or composed protocols have been outside the reach of most formal methods. This lack of scalability has also induced a limited expressiveness of the semantics of security protocol models: most models only allow for modeling a single protocol and its requirements.

Modifying semantics and tools for multi-protocol analysis was already pointed out as an open research question in [61], and has been addressed partially in e.g. [181, 125]. The semantics from Chapter 2 can handle multi-protocol analysis in an intuitive way. The semantics are role-based and the security properties are modeled as local claim events. Therefore, modeling multiple protocols (and their security properties) concurrently amounts to the protocol described by the union of the role descriptions of each protocol.

In the previous chapter a verification tool was developed on the basis of the semantics. In line with the semantics, the tool can handle multi-protocol analysis in a straightforward manner, by concatenation of multiple protocol descriptions. Providing the tool with the concatenation of two protocol descriptions, amounts to verification of their security properties when both protocols are executed over the same network. Using this tool, we have been able to verify two- and three-protocol composition of 30 protocols from the literature. The tests have revealed a significant number of multi-protocol attacks. Because this particular type of analysis has not been performed before, all attacks we find are previously unreported.

We proceed by defining in Section 5.1 the notion of multi-protocol attacks, and we describe the conducted experiments in Section 5.2. The results of the experiments are analysed in Section 5.3, and two practical attack scenarios are treated in Section 5.4. We briefly discuss some preventive measures in Section 5.5 and draw conclusions in Section 5.6.

5.1 Multi-protocol attacks

In the security protocol semantics presented in Chapter 2, there is no explicit notion of "a single protocol". A protocol is simply a collection of roles. If we take the union of two disjoint protocols, the result is still a protocol. Intuitively, we take "a single protocol" to mean a set of roles that are connected by means of the communication relation $\rightsquigarrow$ from Definition 2.10 on page 17. If we join two such protocols, not all

roles will be connected. We say that such a set of role descriptions contains multiple protocols.

Definition 5.1 (Connected roles). *We call two roles connected if they are equivalent under the equivalence relation generated by the communication relation $\rightsquigarrow$.*

Definition 5.2 (Single/multiple protocols). *Given a protocol specification P, the multiplicity of P is defined as the number of equivalence classes as defined by the connected roles relation. If the multiplicity is equal to one we say that P contains a single protocol. If it is more than one, we say it contains multiple protocols.*

All protocols shown in this thesis up to this point are single protocols. There is only one equivalence class of connected roles. If we join the role specifications of two or more protocols with disjoint role sets, the result contains multiple protocols.

Now the definition of a multi-protocol attack is fairly straightforward.

Definition 5.3 (Multi-protocol attack). *Let P be a protocol description which contains multiple protocols, and P' a single protocol with $P' \subseteq P$, i. e. their role descriptions are equal for all roles in the domain of P'. Let γ be a security claim event occurring in P'. We say that there exists a (multi-protocol attack) on γ if there exists an attack on γ within the protocol P, but there is no attack on γ within the protocol P'.*

Note that this definition does not put any constraints on the intruder behaviour. Therefore, this definition is more general than the definition given in [75], where only replaying a message from one protocol into another is allowed.

Corollary 5.4 (No multi-protocol attacks on false claims). *Let P be a single protocol, in which a claim γ occurs. If we have that γ does not hold for P, then there exists no multi-protocol attack on γ.*

5.2 Experiments

In this section we describe the experiments we conducted. The experiments involved using the Scyther tool for the analysis of multi-protocol attacks for a set of protocols, in order to determine whether multi-protocol attacks can occur on existing protocols from the literature.

The set of protocols included in our test is shown in Table 5.1 on page 111. The protocols were selected from the literature: the Clark and Jacob library in [54], the related SPORE library at [174], and the work on protocols for authentication and key establishment in [39] by Boyd and Mathuria. This resulted in a set of 30 protocols. For these experiments, we have considered three security properties: secrecy and two forms of authentication: non-injective agreement and non-injective synchronisation, as defined in Chapter 3. Because the *LOOP* property holds for all protocols in our test, there is no need to separately investigate the injective variants.

The computational costs of verifying properties in multi-protocol environments are exponential with respect to the number of protocols, conform e. g. [81, 183]. Currently it is therefore infeasible to verify an environment with all these protocols in parallel. Instead, we have chosen to test all possible combinations of two or three protocols from this set. Using this method, we managed to find multi-protocol attacks that involve two or three protocols. When such a test yielded an attack, it was verified automatically whether the attack actually required multiple protocols, or could be mounted against a single protocol, in which case the attack is not a multi-protocol attack and is discarded.

The verification results also depend on the type of *matching* used: in particular, whether or not so-called *type-flaw* attacks are possible. We explain this in more detail in the next section. All tests were conducted three times, for a fully typed definition of match, one that allows for basic type flaws, and the untyped version, by varying the definition of the *Match* predicate as described on page 23. In total, over 14000 tests were performed to obtain these results. The experiments have been conducted using the Scyther tool described in the previous chapter.

5.3 Results

The tests reveal that there is a large number of multi-protocol attacks possible on the selected set of protocols. Out of the 30 protocols from the literature, 23 had security claims that are correct in isolation, but had attacks when put in parallel with one or two other protocols from the set.

The three types of matching that were used in the test have a clear hierarchy. Any attack that occurs in the strict type model will also occur in the other two models. Similarly, any attack occurring in the model that allows for basic type flaws, will also be an attack in the typeless model.

For multi-protocol attacks however, there is no such inherent monotonicity. This is caused by the fact that the multi-protocol definition states that the property should be correct in the context of the single protocol. Consider for example the Otway-Rees protocol. For this protocol, the secrecy claims are true in the strict type model, so it may be the case that there exist multi-protocol attacks on these claims in the strict type model. For the typeless model however, there exists type-flaw attacks on both these claims. Consequently we have from Corollary 5.4 that there can be no multi-protocol attacks in the typeless model.

We discuss each of these three typing categories separately.

Strict type matching: no type-flaws

We start off with the most restricted model, in which it is assumed that the agents can somehow check the type of the data they receive, and thus only accept terms of the correct type. For the protocols from literature we found 17 two-protocol attacks, and no three-protocol attacks. We found attacks violating authentication as well as

Table 5.1: Protocol list

Protocols with Multi-Protocol Attacks	
Bilateral Key Exchange	NS symm.
Boyd key agreement	NS symm. amended
Denning-Sacco shared key	SOPH
Gong (nonce)	Splice-AS Hwang and Chen[a]
Gong (nonce) v2	Wide Mouthed Frog (Brutus)
ISO ccitt 509 (BAN)	Woo and Lam pi f
Kao-Chow	Woo-Lam mutual auth.
Kao-Chow v2	Yahalom
Kao-Chow v3	Yahalom (BAN)
KSL	Yahalom-Lowe
Needham-Schroeder	Yahalom-Paulson
Needham-Schroeder-Lowe	

Protocols for which we found no Multi-Protocol Attacks	
Andrew Secure RPC (BAN)	Otway-Rees
Andrew Secure RPC (Lowe)	Splice-AS
ISO IEC 11770 2-13	Splice-AS Hwang and Chen[b]
TMN	

[a] Modified version 1
[b] Modified version 2 (Clark and Jacob)

secrecy requirements.

The vast majority of these attacks involve variants of Woo-Lam Pi protocol as described in [186]. These protocols contain a read/send pattern which can be used as a so-called encryption oracle, which is a protocol mechanism that allows an intruder to encrypt arbitrary values with some key. In this case arbitrary nonces can be encrypted with the symmetric key shared by an agent and the server. This enables many attacks on other protocols that involve this shared key.

The remainder of the attacks share a common pattern that we call *ambiguous authentication*, and will be explained in detail in Section 5.4.

Simple type matching: basic type-flaws only

If we relax the typing constraints on messages, such that variables can contain any term that is not a tuple or an encryption, the number of attack possibilities increases dramatically. Attacks in this category are called basic type-flaw attacks. Specifically, many attacks become possible because (session) keys can be mistaken for Nonces, leading to the revealing of the session keys. This can also cause new authentication attacks. In our tests, we found 40 attacks using basic type-flaw mistakes.

Typeless matching: all type-flaws

As expected, the situation gets worse when any kind of type-flaw attack is possible. Random values can now be mistaken for any tuple term or encrypted term. We found 106 multi-protocol attacks based on all type-flaws.

We also found examples of three-protocol attacks. For example, the Yahalom-Lowe claim of the secrecy of the received session key, is correct in isolation, and is also correct in combination with any other protocol from the test, but can be broken in the presence of the Denning-Sacco shared key and Yahalom-BAN protocols if all type-flaws are possible.

To conclude, we have summarized the influence of the strictness of the matching function (and thus the susceptibility to type-flaw attacks) in Table 5.2.

Table 5.2: Influence of matching function

No type-flaws possible	17 attacks
Basic type-flaws possible	40 attacks
All type-flaws possible	106 attacks

Attack example

As an example of a basic type-flaw multi-protocol attack, we show an attack on the Woo-Lam mutual authentication protocol from [186], together with the Yahalom-Lowe protocol from [123]. The Woo-Lam mutual authentication protocol is shown in Figure 5.2 on page 114, and the Yahalom-Lowe protocol is shown in Figure 5.1 on the next page.

Both protocols have similar goals and preconditions. They use symmetric encryption and a trusted server, in order to generate a fresh session key for authentication communication between two agents. They even operate in a similar way: the initiator i and responder r both create a nonce, which they send to the server. The server creates a new session key ks, and distributes the key, combined with the nonces, back to i and r. They check the nonces, to confirm that the key is indeed fresh.

In Figure 5.3 on page 115 we show a multi-protocol attack on these protocols, exploiting a basic type flaw. This attack is possible if the agent cannot distinguish between a session key and a nonce, assuming that he has not encountered either before.

The attack proceeds as follows. An agent A starts the Woo-Lam protocol in the i role, wants to communicate with another instance of A, and sends a fresh nonce $ni\sharp 1$. The intruder intercepts the nonce. The agent A starts a Yahalom-Lowe session in parallel, in the i role. A creates and sends a second nonce $ni\sharp 2$. This is also intercepted by the intruder.

The intruder now sends the nonce $ni\sharp 2$ to A in the Woo-Lam protocol, as if it was sent by a Woo-Lam responder role. The agent responds with a server request

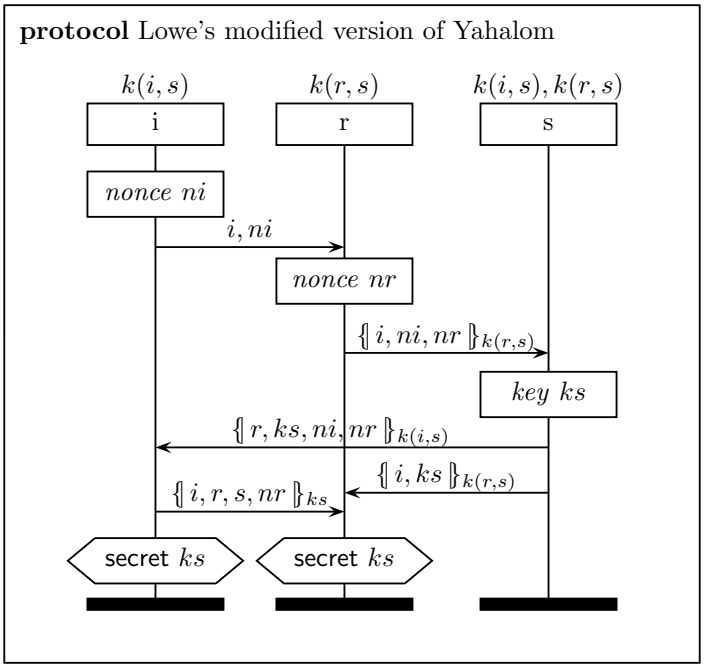

Figure 5.1: Yahalom-Lowe

with the names of both agents and the nonces $\{\!|\, A, A, ni\sharp 1, ni\sharp 2 \,|\!\}_{k(A,S)}$. This message is intercepted, concatenated with itself, and sent to the Woo-Lam server S. The server generates a fresh session key and sends back two (identical) messages $\{\!|\, A, ni\sharp 1, ni\sharp 2, ks\sharp 3 \,|\!\}_{k(A,S)}$. One of these is redirected to the Yahalom-Lowe i role. This role is expecting a message of the form $\{\!|\, A, \mathrm{Key}, ni\sharp 2, \mathrm{Nonce} \,|\!\}_{k(A,S)}$, where Key is a new key and Nonce is a nonce, which he has not encountered before. Thus, he cannot tell the difference between these terms. Because of type confusion, he accepts the message, under the assumption that $ni\sharp 1$ is the fresh session key, and that $ks\sharp 3$ is the responder nonce. Thus, he encrypts the key using the nonce, sends $\{\!|\, A, A, ni\sharp 2, ks\sharp 3 \,|\!\}_{ni\sharp 1}$ and claims that $ni\sharp 1$ is secret. Because the intruder knows $ni\sharp 1$, this is clearly not the case. This is an attack on the Yahalom-Lowe i role. However, we can continue the attack. The intruder intercepts this last message. Because he knows $ni\sharp 1$, he can decrypt the message, and learns the key $ks\sharp 3$. This enables him to create the last message that is expected by the Woo-Lam i role. This role then claims secrecy of $ks\sharp 3$, which is also known to the intruder.

This basic type-flaw attack enables an intruder to break two protocols at the same time.

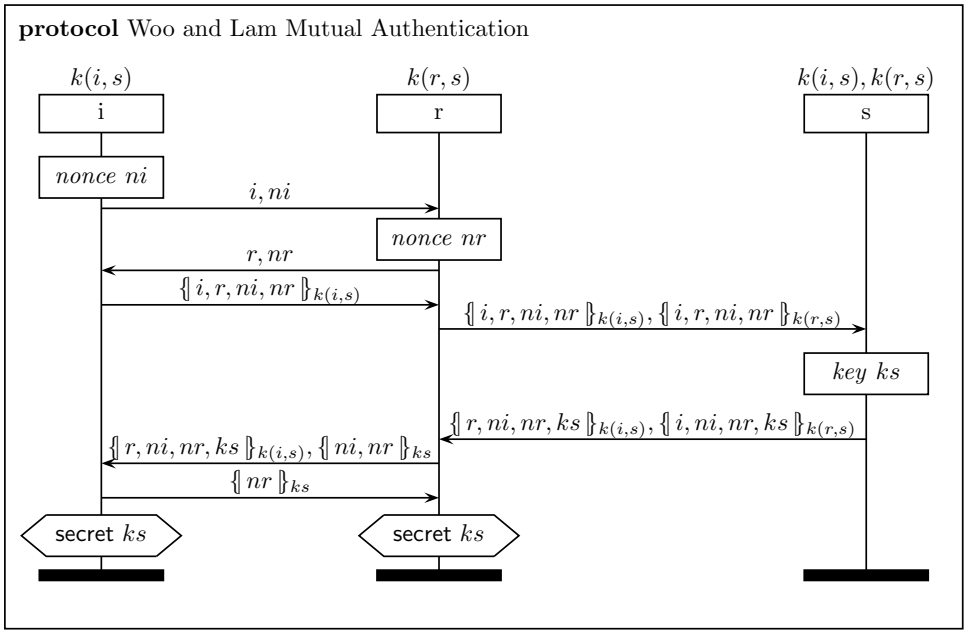

Figure 5.2: Woo-Lam mutual authentication

5.4 Attack scenarios

The experiments have revealed that although many multi-protocol attacks can occur, their scope is limited if type-flaw attacks are prevented. But even if such attacks are excluded, two main scenarios remain in which multi-protocol attacks are likely to occur. We discuss the scenarios in this section, and we discuss some preventive measures in the next section.

Protocol updates

The experiments have shown that multi-protocol attacks are likely for protocols that use similar messages. We describe here a practical scenario where such similarities arise in practice.

It often occurs that security protocols are broken in some way, and that this is discovered after the protocol is deployed. The problem can be fixed by issuing a security update. This is effectively a second protocol, that shares the same key structure, which is very similar to the first one. Such a situation makes multi-protocol attacks very likely.

As an example, we show in Figure 5.4 on page 116 a broken authentication protocol. It is susceptible to a man-in-the-middle attack, similar to the one described in [119]. For our purposes, we only assume this protocol has been distributed to and is being

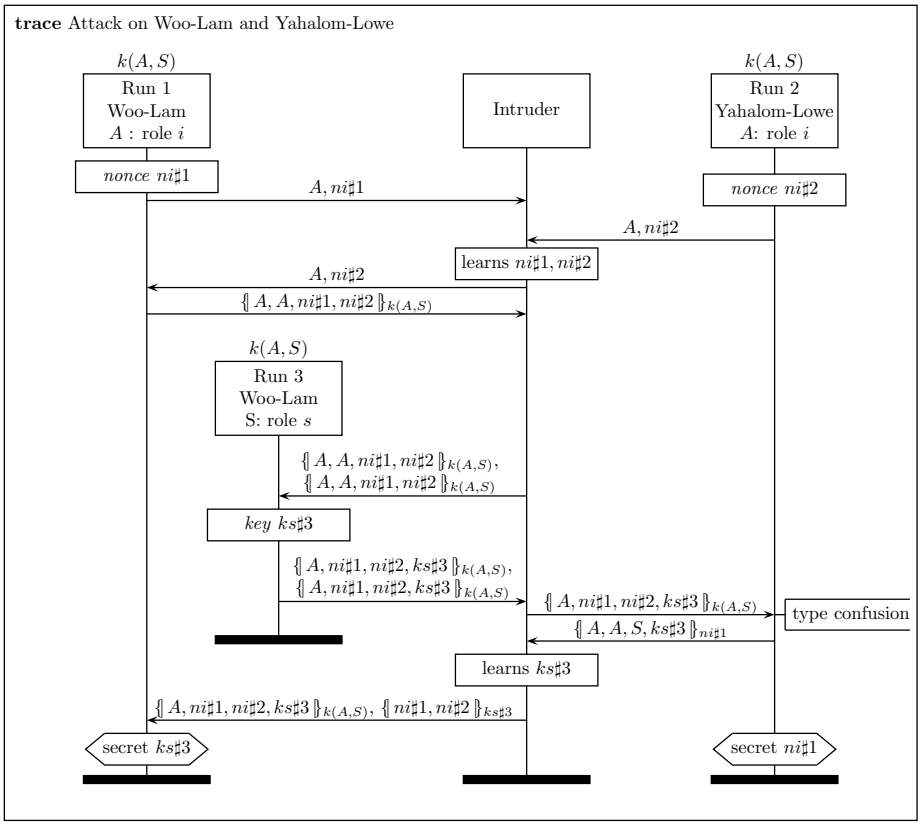

Figure 5.3: Attack on two protocols

used by clients, and that we need to update it with a security fix. The easiest way to fix the protocol is to replace the name in the first message, resulting in the protocol in Figure 5.5 on page 117. This protocol is also known as the Needham-Schroeder-Lowe protocol as we have seen before in this thesis, which can be proven to be correct when run in isolation.

If the broken protocol is updated in such a way that the old version of the protocol can still be running as well by some clients, then there is a multi-protocol attack possible on the new protocol, as shown in Figure 5.6 on page 118. In this attack, the intruder uses two instances of the old protocol (denoted by "Broken i" and "Broken r") to learn the value of a nonce $ni\sharp 1$. Then, an instance of the responder role of the new protocol ("NSL r") is completed using the initiator role of the old protocol. Thus, an agent completing the responder role of the new protocol claims that the nonces $ni\sharp 1$ and $nr\sharp 3$ are secret. Because the nonce $ni\sharp 1$ was already leaked by the old protocol version, this claim is false.

The cause of the problems is that the messages of the updated protocol often closely resemble the messages of the original protocol. Because of this, many possibilities

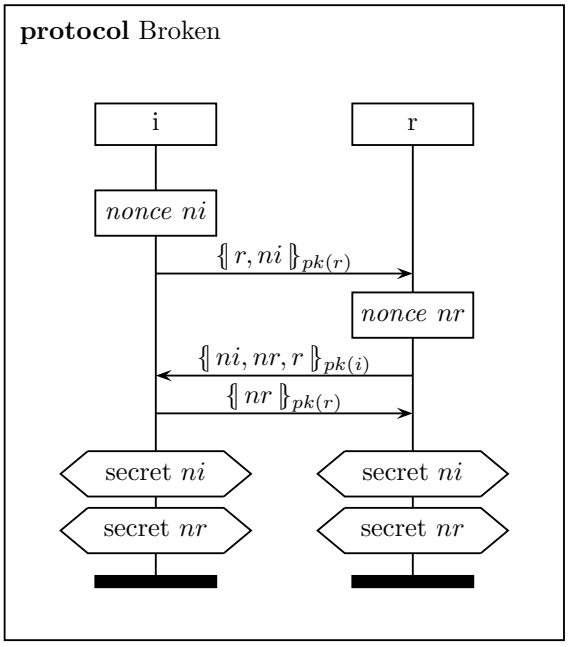

Figure 5.4: Needham-Schroeder: broken

are available for an intruder to insert messages from one protocol at unforeseen places in the other protocol, which opens the door for multi-protocol attacks.

Ambiguous authentication

We use the term "ambiguous authentication" to refer to two or more protocols that share a similar initial authentication phase. This can lead to ambiguity, where protocol mismatches occur between communicating partners.

In particular, authentication protocols are often used to set up session keys for other protocols. The resulting protocol then consists of the sequential execution of the authentication protocol and the protocol that uses the session key. Often the same protocols are used for authentication, which are then composed with different follow-up protocols. In such cases ambiguous authentication can occur: although the authentication protocol is correct in isolation, there can be a multi-protocol attack involving different follow-up protocols.

In the experiments, ambiguous authentication occurred frequently among similar protocols, such as in protocol families, and among broken protocols and their fixed variants.

We give an example of this phenomenon. Consider the protocol pattern "Service 1", as shown in Figure 5.7 on page 119. In this figure, there is a large rectangle denoted as protocol P. For this rectangle, we can substitute any protocol that authenticates

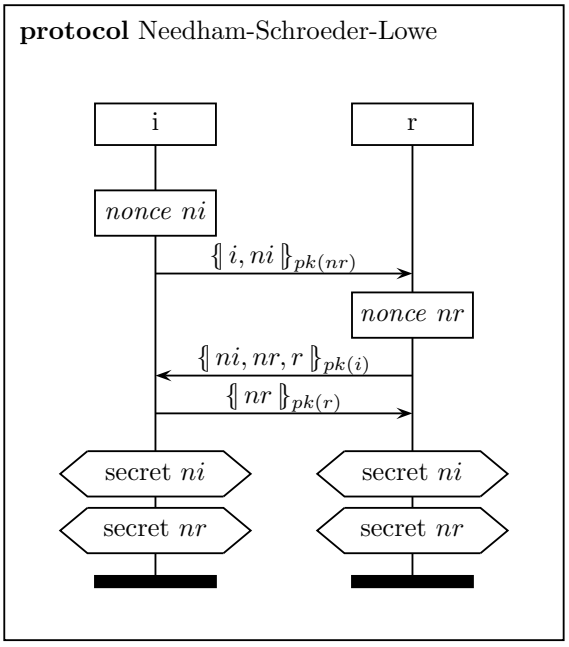

Figure 5.5: Needham-Schroeder-Lowe: fixed

the partners and generates a fresh shared secret. (For example we could insert here the Needham-Schroeder-Lowe protocol from Figure 5.5, and take either of the nonces as the fresh secret ta.) This protocol P is then extended by a single message that sends the secret tb, encrypted with the fresh secret value from the protocol P, from the initiator i to the responder r. Given that the protocol P is correct, we can prove that the complete protocol for Service 1 as a single protocol is correct.

Now we re-use the protocol P to implement another protocol referred to as the Service 2 protocol. See Figure 5.8 on page 119. Here we again use the same base protocol, but we extend it by sending a session identifier and some message m. For the session identifier, we use the fresh random value ta from the base protocol. (If we substitute Needham-Schroeder-Lowe for P, the protocol for Service 2 is correct in isolation.)

If we run Service 1 in parallel with Service 2, the combined protocols are broken. The attack is shown in Figure 5.9 on page 120. In this attack, the intruder simply re-routes the initial messages from Service 1 to Service 2. A executes her side of protocol steps of Service 1 as in the normal execution of the protocol. B on the other hand also executes his side correctly, but using the steps from Service 2. Because both services use the same initial sequence of messages, they cannot detect that the other agent is performing steps of the wrong protocol. After this initial phase A is halfway into Service 1, and B is halfway into Service 2. Therefore B will now use the random value ta as a session identifier, effectively revealing it to the intruder.

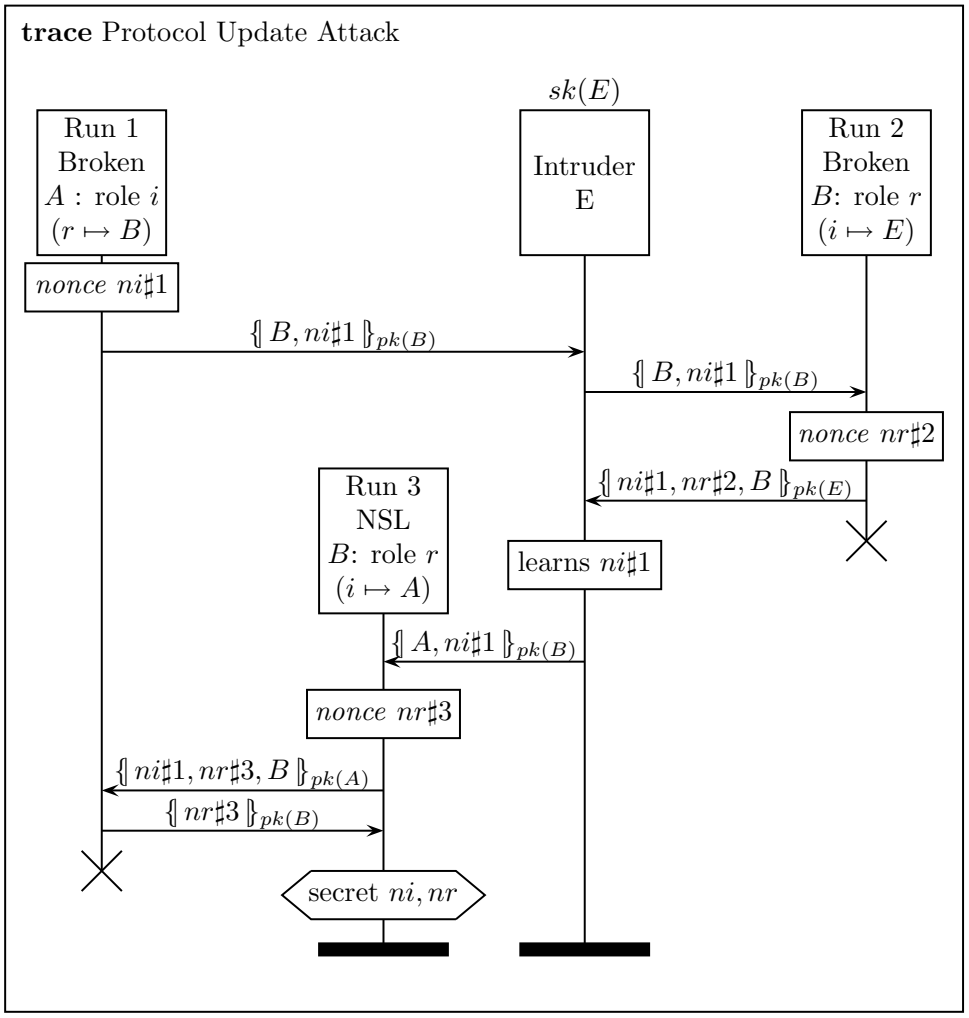

Figure 5.6: NSL attack using the broken variant

Then, when A uses this value ta as a session key for the secret tb, the intruder can decrypt it. Thus the security claim of Service 1 is violated.

5.5 Preventing multi-protocol attacks

Analysis of the experiments has also indicated what is required to effectively prevent multi-protocol attacks. We discuss methods used for type-flaw attack prevention, which can also be used to prevent multi-protocol attacks. We furthermore discuss verification of multi-protocol attacks.

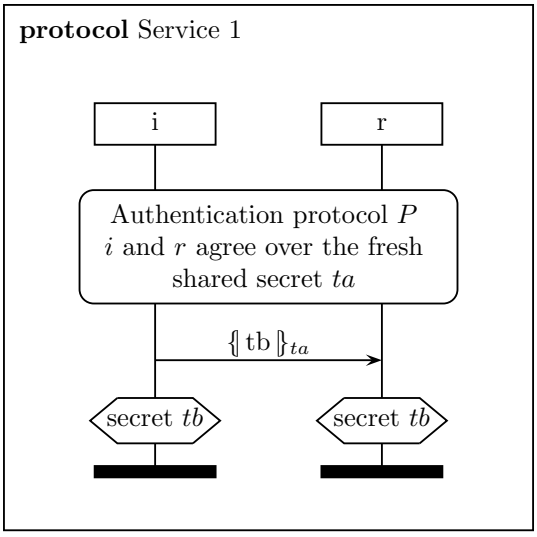

Figure 5.7: Service 1

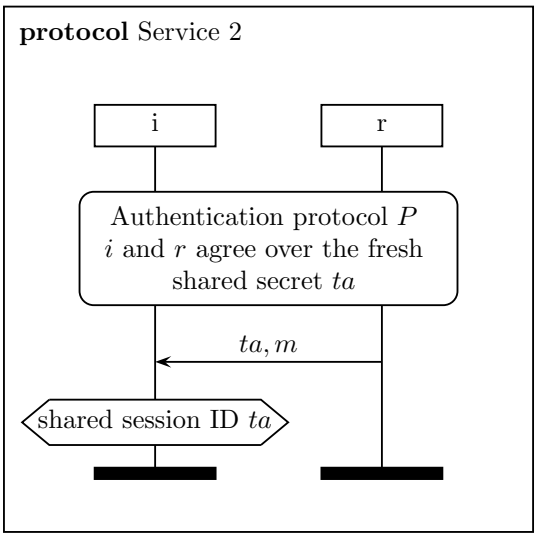

Figure 5.8: Service 2

Strict type detection

As noted in [99], it is possible to prevent type-flaw attacks by adding type information to the messages occurring in a protocol. This significantly reduces the number of possible attacks on a single security protocol, and is therefore advisable even when not considering a multi-protocol environment.

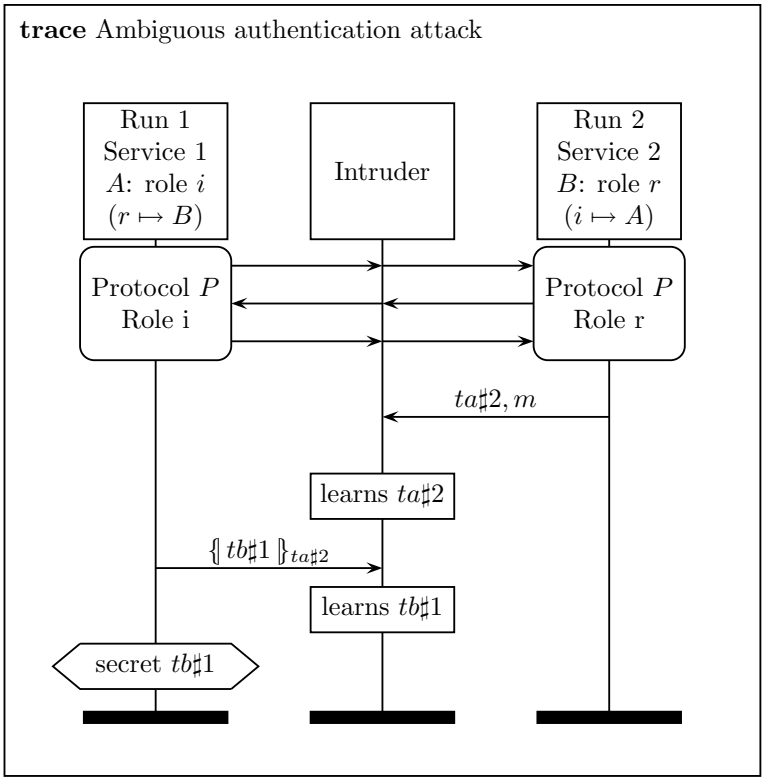

Figure 5.9: Attack on combined services 1 and 2

The experiments detailed here have shown that making sure no type-flaw attacks can occur prevents many multi-protocol attacks. In fact, 84 percent of the attacks in the test set can not occur in a setting without type-flaw errors. Ensuring all messages are typed is therefore also a preventive measure for a large class of multi-protocol attacks. For details of preventing type-flaw attacks we refer the reader to [99, 118].

Verification

In some cases it is undesirable to modify a set of protocols, unless it can be proven that a vulnerability exists. In such cases verification of (multi-protocol) attacks is a realistic option. We have shown here that using suitable tools such as developed in this thesis it is possible to perform automated analysis of concurrent protocols.

5.6 Conclusions

By conducting these multi-protocol verification experiments, we have found 163 multi-protocol attacks. This shows that multi-protocol attacks on protocols from the literature exist in large numbers, and are realistic. All attacks found here are previously, as far as we are aware, unreported. The number of multi-protocol attacks may surprise the reader. The experiments indicate that multi-protocol attacks are a major threat to security protocols in general. The problem of multi-protocol attacks is not limited to a small subset of the protocols. Out of the 30 protocols, we found that 23 of them had security claims that are correct in isolation but for which multi-protocol attacks existed. We identified two common and realistic attack scenarios: protocol updates and ambiguous authentication.

Some of the security claims of the protocols are correct in isolation, and are even correct when put in parallel with any other protocol from the set, but are broken by a 3-protocol attack. This proves that it is not sufficient to check for 2-protocol attacks only. Here we have not investigated the existence of attacks involving four or more protocols, and thus there might even be multi-protocol attacks on these protocols that we have not yet detected. Using formal models and tools has proven invaluable to assess the feasibility of these attacks, and has allowed us to conduct such large scale tests. In fact, many of the attacks are intricate and we would not have been able to find them without tool support.

For multi-protocol environments, it is absolutely necessary to address the interaction between the protocols. This can only be done by looking at all the protocols in the environment: a single protocol can cause all others to break. Taking protocols from the literature that have been proven to be correct in isolation, gives no guarantees at all for multi-protocol environments.

6

Generalizing NSL for Multi-Party Authentication

As a second application of the methods and tools developed in this thesis, we propose a protocol for multi-party authentication for any number of parties, which generalizes the well-known Needham-Schroeder-Lowe protocol. We show that the protocol satisfies injective synchronisation of the communicating parties and secrecy of the generated nonces. For p parties, the protocol consists of $2p-1$ messages, which we show to be the minimal number of messages required to achieve the desired security properties in the presence of a Dolev-Yao style intruder with compromised agents. The underlying communication structure of the generalized protocol can serve as the core of a range of authentication protocols.

In the context of formal black-box analysis of security protocols, several protocols have been proposed in order to satisfy forms of *mutual authentication*. For an overview of authentication protocols see [54, 174]. The Needham-Schroeder-Lowe protocol (NSL) is such a protocol, which satisfies even the strongest forms of authentication such as injective synchronisation, and has been studied extensively.

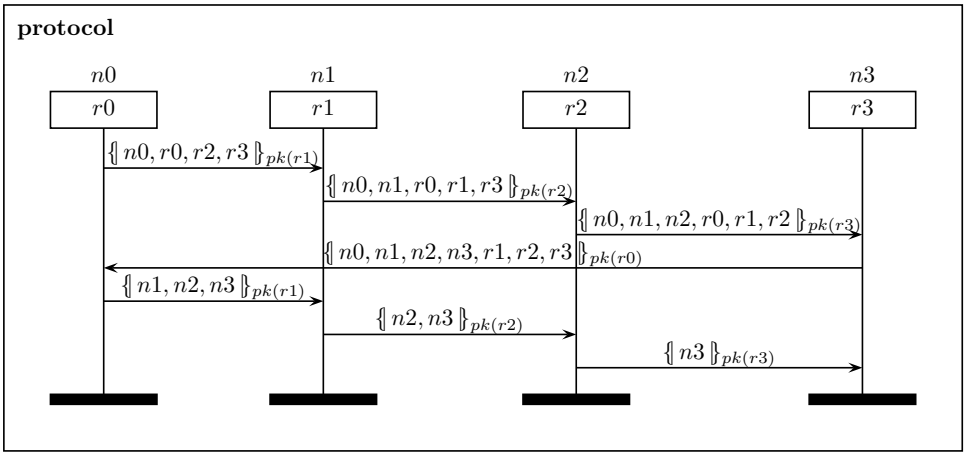

Figure 6.1: Four-party generalized NSL.

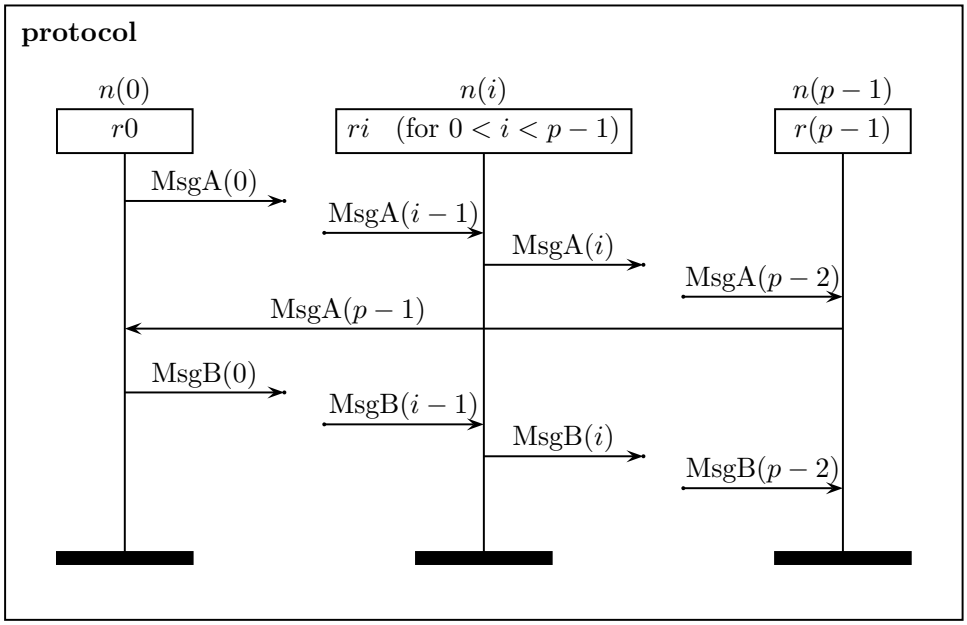

Figure 6.2: Generalized NSL pattern.

The NSL protocol was designed for two parties who want to authenticate each other, and is often referred to as bilateral authentication. In many settings such as modern e-commerce protocols there are three or more parties that need to authenticate each other. In such a setting we could naively instantiate multiple NSL protocols to mutually authenticate all partners. For p parties, such mutual authentication would require $\binom{p}{2} = (p \times (p-1))/2$ instantiations of the protocol, and three times as many messages. In practice, when multi-party authentication protocols are needed, protocol designers instead opt to design new protocols (often 3 or 4-way handshakes, see e.g. [97]). Unfortunately, each new protocol runs the risk of containing flaws, which could have been avoided by re-using known correct protocols.

The goal of this chapter is to improve upon the approach that uses multiple instances of bilateral authentication, and to generalize the NSL protocol so as to obtain a multi-party authentication protocol with optimal message complexity (which turns out to be $2p - 1$ for p parties). We require that, in the setting of a Dolev-Yao intruder model with compromised agents, the generalized protocol satisfies at least the same security requirements as NSL. The situation where we have a family of protocols (one protocol for each p) is a bit different from that of a single protocol. Normally, a single protocol is proven correct in a context where we assume parallel sessions of that same protocol. For a protocol family, we assume that an agent may run sessions of different protocols of this family in parallel. This may increase the possibilities of the adversary to attack the protocol. Therefore, we require for each p that the protocol is correct, even if the agents run instances of the protocol for different values of p in parallel.

The existing methodologies and tools for verifying security protocols in a Dolev-Yao setting have been focussed on protocols with a fixed number of parties. The challenge is in proving the proposed parameterized protocol correct, using the formal model developed here.

We proceed as follows. In Section 6.1 we generalize the Needham-Schroeder-Lowe protocol to any number of parties. In Section 6.2 we show the security properties that the protocol satisfies and sketch proofs of correctness and preconditions. We discuss some variations of the pattern of the generalized protocol in Section 6.3. In Section 6.4 we show that a previously generalized protocol for symmetrical keys does not satisfy our authentication requirements. We draw conclusions and discuss further work in Section 6.5.

6.1 A multi-party authentication protocol

The basic idea behind the NSL protocol is that each agent has a challenge-response cycle to validate the other agent's identity. These two challenge-response cycles are linked together by identifying the response of the second agent with its challenge.

Its generalization follows the same line of thinking. Every agent conducts a challenge-response cycle with its neighbouring agent, while combining its own challenge with a response to another agent's challenge whenever possible. We will first explain the four-party version of the protocol in some detail. Afterwards we give a generalized specification for p parties.

The four-party protocol goes as follows (see Figure 6.1). First, the initiating agent chooses which parties he wants to communicate with. He creates a new random value, $n0$, and combines this with his name and the names of agents $r2$ and $r3$. The resulting message is encrypted with the public key of $r1$, and sent to $r1$. Upon receipt and decryption of this message, the second agent adds his own name and a fresh nonce, and removes the name of the next agent in the line from the message. This modified message is then encrypted with the public key of the next agent and sent along. This continues until each agent has added his nonce, upon which the message is sent back to the initiating agent. This agent checks whether the message contains the nonce he created earlier, and whether all agent names match. Then he can conclude that the other agents are authenticated. Next, in order to prove his own identity, he sends a message containing the other agents' nonces to $r1$. The subsequent agents again check whether their own nonces are in the message, remove this nonce, and pass the resulting message on.

This four-party protocol can be generalized to any number of agents p. In Figure 6.2 we schematically describe the communication structure of the protocol. The abstract messages are defined below. The function *next* determines the next role in the list of participants in a cyclic way. The ordered list $\mathrm{AL}(x)$ contains all roles, except for role x. The protocol makes use of two types of messages. The first p messages are of type MsgA and the final $p-1$ messages are of type MsgB.

$$
\begin{aligned}
next(i) &= r((i+1) \bmod p) \\
\mathrm{AL}(x) &= [r0, r1, \ldots, r(p-1)] \setminus \{x\} \\
\mathrm{MsgA}(i) &= \{\!|\, [n(0) \ldots n(i)], \mathrm{AL}(next(i)) \,|\!\}_{pk(next(i))} && 0 \le i < p \\
\mathrm{MsgB}(i) &= \{\!|\, [n(i+1) \ldots n(p-1)] \,|\!\}_{pk(next(i))} && 0 \le i < p-1
\end{aligned}
$$

For i such that $0 \le i < 2p-1$, protocol message labeled with (p, i) is now defined by

$$
\mathrm{Msg}(i) = \begin{cases} \mathrm{MsgA}(i) & \text{if } 0 \le i < p \\ \mathrm{MsgB}(i-p) & \text{if } p \le i < 2p-1 \end{cases}
$$

The purpose of the protocol is to achieve authentication of all parties and secrecy of all nonces, in the presence of a Dolev-Yao intruder that has full control over the network. We will make this precise in the next section, but first we make two observations. First, a run of role rx reads messages with labels $x-1$ and $x+p-1$, and sends out messages with labels x and $x+p$. Second, a nonce created by a run of role rx occurs in p messages, in the messages labeled with i, where $x \le i < x+p$.

The protocol can be deployed in two main ways. First, it can be instantiated for a specific number of parties, to yield e.g. a four-party authentication protocol. In this way it can be used instead of custom n-way handshake protocols. Second, it can be used in its most generic form, and have the initiating role $r0$ choose the number of participants p. Agents receiving messages can deduce the chosen p at runtime from the number of agents in the first message, and their supposed role from the number of nonces in the message. For the analysis in the next section we use the generic form, where the number of parties p is not fixed. The properties that we prove will then automatically hold for specific instantiations as well, where only a certain number of parties is allowed.

6.2 Analysis

The multi-party authentication protocol described above has a number of interesting properties. It satisfies secrecy of each of the nonces, and authentication for all parties. Furthermore, it uses a minimal number of messages to achieve these properties.

6.2.1 Properties of generalized NSL

We want the generalized version of NSL to satisfy injective synchronisation as well as secrecy of all the nonces, for each role.

We notice that the proposed multi-party authentication protocol performs an *all or none* authentication. This means that whenever an agent finishes his part of the protocol successfully, he will be sure that all other parties are authenticated to him.

On the other hand, if any of the communication partners is not able to authenticate himself, the protocol does not terminate successfully. So, this protocol does not establish authentication of a subset of the selected agents.

A second observation concerning this protocol is the fact that authentication is only guaranteed if all agents indicated in the first message of the initiator are *trusted*. This means that if the decryption key of any of the agents is compromised, the other agents in the list can be falsely authenticated. The reason is that e.g. an agent performing role $r0$ only verifies the authenticity of agent $r1$. Verification of the identity of an agent performing role $r2$ is delegated to an agent performing role $r1$, and so on. This chain of trust is essential to the design of an efficient multi-party authentication protocol. This does not mean that we restrict the Dolev-Yao intruder model, since sessions with untrusted agents are still possible. As is the case for the standard NSL protocol and most other authentication protocols, a session with a compromised partner does not have to satisfy authentication.

Finally we want to mention that the proof does not only imply correctness for any specific choice of p. Rather, we prove that the protocol is correct when p is chosen at run-time by the initiator. Put differently, we prove correctness of the protocol in an environment with all p-party variants running in parallel.

6.2.2 Proof of correctness

We will show the proofs of the security claims of the generalized version of Needham-Schroeder-Lowe in some detail.

Proof outline. The proof exploits the fact that nonces generated by agents are initially secret, even if the intruder learns them later. Using the secrecy of the nonce up to the point where it is leaked, we establish a sequence of messages. If the nonce is leaked at the end of the sequence, we establish a contradiction, which is the basis of the secrecy proof. Once we know the nonce of a role remains secret, we can use the same message sequence to establish synchronisation.

We assume a matching function that checks the type of the incoming messages. In other words, we assume the messages are constructed in such a way that the recipient can distinguish e.g. a nonce from an agent name. This assumption is addressed in more detail in Section 6.2.3

In order to prove our results, we use the notation $M(\alpha, i)$ to denote the knowledge set of the intruder after executing the events in a trace α, up to but not including α_i. If $i \geq |\alpha|$ we have that $M(\alpha, i)$ is equal to the knowledge of the intruder after the execution of all events in the trace.

Recall that local constants (e.g. nonces) created in runs are not known to the intruder initially. Even if the intruder learns them at some point, there is a point before which the constant was not known.

Lemma 6.1. *Given a trace α, and a run term of the form $n\sharp\theta$, we have that:*

$$\exists j : M(\alpha, j) \vdash n\sharp\theta \;\Rightarrow\; \exists i : M(\alpha, i) \nvdash n\sharp\theta \;\wedge\; M(\alpha, i+1) \vdash n\sharp\theta$$

Proof. We have that $M_0 \nvdash n\sharp\theta$ by the definition of initial intruder knowledge. Furthermore, the intruder knowledge is non-decreasing according to the operational semantics rules for the Dolev-Yao intruder (i. e. according to the take and fake rules). □

If a local constant of some run is not known to the intruder, he cannot construct terms that have this constant value as a direct subterm himself. Thus, these messages must have been learned somewhere before.

The previous result holds for all protocols in our model. In contrast, the following lemmas only hold for the specific protocol under investigation.

Lemma 6.2. *We assume a matching function that does not allow nonce variables to be instantiated with encrypted terms. Given a trace α, run terms $n\sharp\theta, m, k$ and an index i, and let α_i be a protocol event of the generalized NSL protocol. Given that $n\sharp\theta \in parts(m)$, we have that*

$$\big(M(\alpha,i) \nvdash n\sharp\theta \wedge \alpha_i \in \mathit{ReadRunEv} \wedge \{\!|\, m \,|\!\}_k = \mathit{cont}(\alpha_i)\big) \Rightarrow \\ \big(\exists j : j < i \wedge \alpha_j \in \mathit{SendRunEv} \wedge \{\!|\, m \,|\!\}_k = \mathit{cont}(\alpha_j)\big)$$

Proof. Observe that $\{\!|\, m \,|\!\}_k$ is the generic form of all the read messages $\mathit{cont}(\alpha_i)$ of the generalized NSL protocol. From the read semantics we conclude that $M(\alpha,i) \vdash \{\!|\, m \,|\!\}_k$. Looking at the components of this term, we find that because $M(\alpha,i) \nvdash n\sharp\theta$ and $n\sharp\theta \in \mathit{parts}(m)$, it must be the case that $M(\alpha,i) \nvdash m$. In general, for all knowledge sets M we have that

$$M \vdash \{\!|\, m \,|\!\}_k \Rightarrow (M \vdash m \wedge M \vdash k) \vee (\exists t \in M : \{\!|\, m \,|\!\}_k \sqsubseteq t)$$

as a direct result of the definitions of $\vdash$ and $\sqsubseteq$. The elements of $M(\alpha,i)$ are either elements of M_0, or they were used as part of a term for take transitions, which were sent before. As m cannot be part of M_0 (as it contains the nonce), in order for $\{\!|\, m \,|\!\}_k$ to be in the role knowledge, it must be a subterm of a term that was previously sent. We identify this send event with α_j. We investigate the protocol send events, and observe that variables may not be instantiated with encryptions. This implies that $\{\!|\, m \,|\!\}_k = \mathit{cont}(\alpha_j)$. □

Based on Lemma 6.2 we can establish the following property:

Lemma 6.3. *Consider the generalized version of NSL. Let α be a trace, and $n\sharp\theta$ an instantiated constant. If we have*

$$M(\alpha,i) \nvdash n\sharp\theta \wedge \alpha_i \in \mathit{SendRunEv} \wedge n\sharp\theta \sqsubseteq \mathit{cont}(\alpha_i) \wedge \mathit{runidof}(\alpha_i) \neq \theta$$

then we have

$$\exists i'', i' : i'' < i' < i \ \wedge \ n\sharp\theta \sqsubseteq \mathit{cont}(\alpha_{i''}) = \mathit{cont}(\alpha_{i'}) \wedge \\ \alpha_{i''} \in \mathit{SendRunEv} \ \wedge \ \alpha_{i'} \in \mathit{ReadRunEv} \ \wedge \ \mathit{runidof}(\alpha_{i'}) = \mathit{runidof}(\alpha_i)$$

Proof. Given that $runidof(\alpha_i) \neq \theta$, the run to which α_i belongs is not the run that created the nonce. Thus, there must be a variable V of this run that is instantiated with the nonce. Variables are assigned only at read events, and thus the run $runidof(\alpha_i)$ must have a preceding read event $\alpha_{i'}$ of which the nonce is a subterm. Based on this event and the fact that the nonce is secret, we find from Lemma 6.2 that there must be a previous send event with identical contents. □

Intuitively, the lemma states that if an agent sends out a nonce generated in a run (which can therefore not be part of M_0), then it is either its own nonce or it is one that it learned before from the send of some other agent.

The previous lemma gives us a preceding run for a message that is sent. If we repeatedly apply this lemma to a similar situation where a nonce is received, we can establish a sequence of events that must have occurred before a nonce is received. This is expressed by the next lemma.

Lemma 6.4. *Consider the generalized version of NSL. Let α be a trace, and rx be the identifier of a run executing role x in which a nonce $n\sharp rx$ was created. If we have*

$$M(\alpha, i) \nvdash n\sharp rx \ \wedge \ \alpha_i \in ReadRunEv \ \wedge \ n\sharp rx \sqsubseteq cont(\alpha_i) \ \wedge \ runidof(\alpha_i) \neq rx$$

then there exists a non-empty finite sequence of events β such that the events in β are a subset of the events in α, and

$$\begin{aligned} &runidof(\beta_0) = rx \wedge cont(\beta_{|\beta|-1}) = cont(\alpha_i) \wedge \\ &\quad \Big(\forall k : 0 \leq k < |\beta| : \exists j, j' : j < j' < i \ \wedge \ \beta_k = \alpha_j \in SendRunEv \wedge \\ &\qquad \alpha_{j'} \in ReadRunEv \ \wedge \ n\sharp rx \sqsubseteq cont(\alpha_j) = cont(\alpha_{j'}) \wedge \\ &\qquad\quad \big(k < |\beta| - 1 \Rightarrow runidof(\alpha_{j'}) = runidof(\beta_{k+1})\big)\Big)\Big) \end{aligned}$$

Proof. This lemma is the result of repeated application of the previous lemma. Because the events in the trace that precede α_i are finite, β must also be finite. Note that the last conjunct expresses that the nonce is read by a certain run, it is sent out later by that same run, unless it is the final read event (expressed by $k = |\beta| - 1$). □

The resulting sequence of events is a chain of send events that include the nonce, where each sent message is read by the run of the next send. This lemma is used to trace the path of the nonce from the creator of a nonce to a recipient of a message with the nonce, as long as the nonce is not known to the intruder. In other words, β represents a subset of send events of α through which the nonce $n\sharp rx$ has passed before α_i.

We now derive some additional information from the sequence of events β as established by Lemma 6.4. Given a send event e, the type of message is either A or B, and is denoted as mtype(e).

Lemma 6.5. *Given a sequence of run events β established by application of Lemma 6.4, we have that there exists an index k, where $1 \leq k \leq |\beta|$ such that*

$$(k < |\beta| \Rightarrow role(\beta_k) = r0) \ \wedge \ \forall n : 0 \leq n < |\beta| : mtype(\beta_n) = \begin{cases} A & \text{if } n < k \\ B & \text{if } n \geq k \end{cases}$$

where $mtype(e)$ yields either A or B for an event e, corresponding to the two message types of the generalized NSL protocol. Type A contains agent names and nonces, whereas type B contains only nonces.

Proof. This lemma follows from the protocol rules. When a run creates a nonce, it is sent out first as part of a message of type A. Messages of type A contain agent names, whereas messages of type B do not. The run that receives such a message, sends it out again within a type A message (containing agent names), unless it is executing role $r0$, in which case it sends out the nonce within a type B message, without agent names. After receiving a message without agent names (type B), runs only send out type B messages. □

In the lemma, the function of k is to indicate the point where the sequence changes from type A to type B, which occurs when an agent in run $r0$ sends out the type B message after reading the type A message. It can be the case that no such change point exists. This is covered by $k = |\beta|$: all messages are of type A, and thus we cannot immediately derive that there is an event of role $r0$ occurring in β.

Given a run identifier r, we denote the local mapping of roles to agents (which is used to denote the intended communication partners of a run) with $\rho(r)$. This allows us to draw some further conclusions. Because the messages in the sequence β are received as they were sent, and because the messages before k include a list of agents, we deduce:

- The runs executing the events $\beta_0 \ldots \beta_k$ have the same parameter p (the number of agents in the messages plus one) and each run has the same agent mapping ρ.
- Given the parameter p and the number of nonces in a message, we can uniquely determine the role of a message.

This leads to the following result:

Lemma 6.6. *Given a sequence β resulting from applying Lemma 6.4 for a nonce created in a run rx executing role x, and an index k resulting from Lemma 6.5 we have*

$$(k < |\beta| \Rightarrow role(\beta_k) = r0) \wedge \\ \forall n : 0 \leq n < k : \rho(runidof(\beta_n)) = \rho(rx) \ \wedge \ role(\beta_n) = r(n + x)$$

Proof. All messages of type A explicitly contain the agent names, except for the name of the agent whom the message is for, which is encoded in the public key that is used. The number of agents defines the parameter p for both the send and read event, and combined with the number of nonces, uniquely defines the role of which the send and read events must be a part. □

For all previous lemmas we required as a precondition that some nonce was secret. In order to identify the point up to which a given nonce is secret, we have the following lemma:

Lemma 6.7. *For the generalized version of NSL, given a trace α, and a nonce $n\sharp rx$ that was created by a run rx, and a trace index k:*

$$\begin{aligned} &rng(\rho(rx)) \subseteq Agent_T \ \wedge\ M(\alpha,k) \vdash n\sharp rx \Rightarrow \\ &\big(\exists ry, j : rng(\rho(ry)) \not\subseteq Agent_T \ \wedge\ j < k \ \wedge\ M(\alpha,j) \nvdash n\sharp rx \ \wedge\ M(\alpha,j+1) \vdash n\sharp rx \wedge \\ &\qquad \alpha_j \in SendRunEv \ \wedge\ runidof(\alpha_j) = ry \ \wedge\ n\sharp rx \sqsubseteq cont(\alpha_j)\big) \end{aligned}$$

Proof. Observe that for the generalized version of the NSL protocol, the initial intruder knowledge only contains the secret keys of the untrusted agents. All messages that are sent by a run ry are encrypted with public keys of agents from the set $rng(\rho(ry))$. Thus, from the inference rules we find that the intruder knowledge can only derive the contents of such a message if a run communicates with untrusted agents. □

Secrecy of nonces created in role $r0$

Based on the previous lemmas we can prove that nonces generated in a run that performs role $r0$, and tries to communicate with trusted agents only, are kept secret.

Lemma 6.8. *Given a trace α, a nonce $n\sharp rx$ created by a run rx in role $r0$, we have that*

$$rng(\rho(rx)) \subseteq Agent_T \Rightarrow \forall i : M(\alpha,i) \nvdash n\sharp rx$$

Proof. We prove this by contradiction. We assume the generated nonce is leaked to the intruder, and establish a contradiction, from which we conclude the nonce cannot be leaked.

Let α be a trace in which a nonce $n\sharp rx$ was generated in a run rx executing role $r0$, and that this run tries to communicate with trusted agents only. In other words, $rng(\rho(rx)) \subseteq Agent_T$. Assume the nonce is learned by the intruder at some point. We apply Lemma 6.7 to find an event α_j of a run ry where the nonce is first leaked. Thus we have $M(\alpha,j) \nvdash n\sharp rx$ and $M(\alpha,j+1) \vdash n\sharp rx$. Note that $rx \neq ry$: The nonce could not have been created in run ry because that would imply ry only communicates with trusted agents, contradicting the lemma. In fact, in all sub cases we will arrive at a contradiction of the type $\rho(rx) = \rho(ry)$ combined with the

communication with (un)trusted agents in the runs. We apply Lemmas 6.4 and 6.5 to yield a sequence of events β and an index k.

We split cases based on the type of message of the send event at α_j. We distinguish two cases, and show that both lead to a contradiction.

- *The message sent at α_j is of type A.* Thus we conclude $k = |\beta|$, and from Lemma 6.6 we have that $\rho(ry) = \rho(rx)$. Because rx communicates with trusted agents only, and ry does not, we arrive at a contradiction.

- *The message sent of α_j is of type B*, so it does not contain agent names. Because the nonce $n\sharp rx$ was not created by the run ry, it must have been received before. If we look at the protocol definitions, we see that each send of message B is preceded by a read of a message containing one extra nonce, the one created by the run. Thus, there must be a read event α_{j2}, $j2 < j$, with message $\{\!|\, [n\sharp ry, \ldots, n\sharp rx, \ldots] \,|\!\}_{pk(\ldots)}$, where $n\sharp ry$ is the nonce created by run ry. Because this message again contains $n\sharp rx$, the intruder could not have created this message himself, and an agent must have sent it. If we look at the messages in the sequence β and the protocol rules, we find that there must exist an index $k2$, such that $runidof(\beta_{k2}) = ry$ (where $n\sharp ry$ was sent out first), and that $k2 < k$ on the basis of Lemma 6.5. If we combine this with Lemma 6.6, we arrive at $\rho(ry) = \rho(rx)$, which yields a contradiction.

□

Non-injective synchronisation of role $r0$

Given that the secrecy of nonces generated by role $r0$ holds, the following is straightforward:

Lemma 6.9. *Non-injective synchronisation holds for role $r0$.*

Proof. We sketch the proof, which is not difficult given the secrecy of the nonce. Given a trace α with a run rx executing role $r0$, we have that the nonce n generated by this role is secret on the basis of Lemma 6.8. Thus, if the agent completes his run, there must have been two indices j and i, $j < i$ such that $\alpha_j = send_{(p,0)}(m)$ and $\alpha_i = read_{(p,p-1)}(m')$. If we use Lemma 6.4 and Lemma 6.5 we find that the events in the sequence β are exactly the events that are required to exist for the synchronisation of the role $r0$. The messages of these events are received exactly as they were sent, which is required for synchronisation. This leaves us with only one proof obligation. We have to show that the sequence β contains all the messages of type A, in the right order, after the start of run rx, and before the end of run rx. This follows directly from the role assignment in Lemma 6.6. □

Secrecy of nonces created in role rx for $x > 0$

Lemma 6.10. *Given a trace α, a nonce $n\sharp rx$ created by a run rx executing role rx with $x > 0$, we have that*

$$rng(\rho(rx)) \subseteq Agent_T \Rightarrow \forall i : M(\alpha, i) \nvdash n\sharp rx$$

Proof. Proof by contradiction, similar to that of Lemma 6.8. Assume the nonce $n\sharp rx$ was created by a run rx executing role x, and is leaked by a run ry to the intruder in some trace α. We have $rng(\rho(rx)) \subseteq Agent_T$ and $rng(\rho(ry)) \not\subseteq Agent_T$. We use Lemma 6.4 and Lemma 6.5 to yield a sequence β and index k. We distinguish two cases:

- $k = |\beta|$: We derive $\rho(rx) = \rho(ry)$, leading to a contradiction.
- $k < |\beta|$: Because $role(\beta_k) = r0$, we use Lemma 6.9 to extend the sequence β backwards, by merging the sequence from the leaking of the nonce with the sequence from the synchronisation. From the messages in the initial segment, which now includes all roles, we find that $\rho(rx) = \rho(ry)$, leading to a contradiction.

□

Theorem 6.11 (Secrecy of all nonces). *For the generalized version of the NSL protocol, we have that all nonces created in runs rx for which we have $rng(\rho(rx)) \subseteq Agent_T$, are secret.*

Proof. A direct result of Lemmas 6.8 and 6.10. □

Non-injective synchronisation of role rx for $x > 0$

For non-injective synchronisation of role rx for $x > 0$, we not only have to prove that all messages of type A have occurred as expected, but also all messages MsgB(x), and that there is so-called run-consistency: for each role ry we want there to be a single run that sends and receives the actual messages.

Lemma 6.12. *Non-injective synchronisation holds for role rx, where $x > 0$.*

Proof. Proof sketch: based on the secrecy of the nonce generated in such a role, we determine an index k, and sequence β that precedes the last read event of the role, with $role(\beta_0) = rx$. Because the sequence must include an event of role $r0$, for which non-injective synchronisation holds, we merge the sequences for both (as in the previous lemma). This gives us a complete sequence of send and read events which exactly meet the requirements for non-injective synchronisation. The messages of these events are received exactly as they were sent. The requirement of the existence of all messages of type A follows from Lemma 6.6, which allows us to conclude that there is a run for each role in the protocol. Furthermore, the nonce of each these runs is present in the message sent at α_k. If we examine the protocol rules, we see

that the message of type B is only accepted by runs whose own nonce is contained in the message. Therefore we have that the run executing role $r1$ must be equal to $runidof(\alpha_{k+1})$, and that the read event must be labeled as the MsgB(0). Similarly, we establish that the correct messages have been consistently read and sent by the runs that created the nonces. Thus all conditions for non-injective synchronisation are met. □

Theorem 6.13 (Non-injective synchronisation). *For the generalized version of the NSL protocol, we have that for all runs rx with $rng(\rho(rx)) \subseteq Agent_T$, non-injective synchronisation holds.*

Proof. A direct result of Lemmas 6.9 and 6.12. □

Injective synchronisation of all roles

The additional requirement of *injectivity* ensures that a security protocol is not vulnerable to a certain class of replay attacks. In 3 we formalised the notion of injectivity and proved that for synchronising protocols inspection of the protocol at a syntactic level suffices to conclude injectivity. This syntactic criterion, the *loop-property*, clearly holds for all roles of the generalized protocol. Therefore, the synchronisation proof presented above implies injective synchronisation as well.

Theorem 6.14 (Injective synchronisation). *For the generalized version of the NSL protocol, we have that for all runs rx with $rng(\rho(rx)) \subseteq Agent_T$, injective synchronisation holds.*

Proof. Follows from Theorem 3.24 and Theorem 6.13. □

6.2.3 Observations

The Needham-Schroeder protocol. If we instantiate the generalized NSL protocol for two parties ($p = 2$), we get exactly the three message version of the NSL protocol. The NSL protocol was designed to fix a flaw in the Needham-Schroeder protocol, shown in Figure 6.3. If we compare our generalized version with the original Needham-Schroeder protocol, we see that the second message of the Needham-Schroeder protocol does not contain the agent list ($\mathrm{AL}(a)$). Therefore we cannot conclude all the contradictions based on the matching agent lists, as we did in the proof of our protocol.

Type-flaw attacks. We have assumed that type-flaw attacks are not possible, i.e. agents can verify whether an incoming message is correctly typed. There are several reasons for doing this.

Without this assumption, there are type-flaw attacks on the generalized version of the protocol. This is not restricted to simple ones for specific instances of p, but also multi-protocol type-flaw attacks involving instances for several choices of p in one

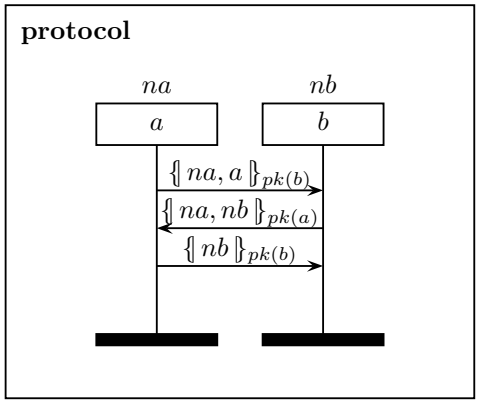

Figure 6.3: The Needham-Schroeder protocol with public keys.

attack, as in the previous chapter. Thus, we find that typing is crucial. Solutions for preventing type flaw attacks using type information is examined in detail in e.g. [99]. Such type information can be easily added to each message, but a simple labeling will also suffice. If we add a tuple (p, l) before each message inside the encryption, where p is the number of participants for this instance, and l is the label of the message, the protocol becomes robust against type-flaw attacks and multi-protocol attacks with other instances of itself. In the proof, we used the fact that the label could be derived from the type. If we explicitly add such a label, the proof works in the same way for untyped models, except that the label is now explicit instead of being derived from the type.

Using the tools described in Chapter 4, we have established that the type-flaw attacks are not due to the specific ordering of the nonces and agent names within the messages. In particular, we examined different options for the message contents (without adding labels), such as reversing the order of either the agent or the nonce list, interleaving the lists, etc. We established the existence of type-flaw attacks for some choices of p for all variants we constructed.

6.2.4 Message minimality

As discussed in Section 3.5.1, the loop-property is instrumental to achieve injectivity. Moreover, in the context of a unicast communication model with a Dolev-Yao intruder, this loop-property turns out to be a necessity. Phrased in terms of challenge-response behaviour, we can say that in order to achieve injective synchronisation, each role must send a challenge that is replied to by all other roles.

From this requirement we can easily derive the minimal number of messages to achieve injective synchronisation. Consider the first message sent by some role rx, and call this message m. In order to achieve a loop to all other roles after this first message, every role will have to send at least one message after m. Including message m this will yield at least p messages. Next we observe that every role must

take part in the protocol and we consider the first message sent by each of the roles. If we take rx to be the last of the p roles that becomes active in the protocol, it must be the case that before rx sends his first message, at least $p-1$ messages have been sent. Adding this to the p messages that must have been sent after that message, yields a lower bound of $2p-1$ messages.

6.3 Variations on the pattern

The communication structure from Figure 6.2 can be instantiated in several different ways as to obtain authentication protocols satisfying different requirements. In this section we list some of the more interesting possibilities.

Generalized Bilateral Key Exchange. First, we observe that the nonces generated in the protocol are random and unknown to the adversary, which makes them suitable keys for symmetric encryption. Furthermore, if we examine the proofs, the authentication of the messages is only derived from the encryption of the messages of type A, not of type B. Similar to the Bilateral Key Exchange protocol (BKE) as described in [54] we can opt to replace the asymmetric encryption for the messages of type B by symmetric encryption with the nonce of the recipient. We can then omit this nonce from the list. We use ϵ to denote a constant representing the empty list. This yields the following message definitions. Figure 6.4 illustrates the four-party BKE protocol.

$$\begin{aligned}
\mathrm{nlist}(i) &= \begin{cases} [n(i+2)\ldots n(p-1)] & \text{if } i < p-1 \\ \epsilon & \text{if } i = p-1 \end{cases} \\
\mathrm{MsgA}(i) &= \{\!|\, [n0\ldots ni], \mathrm{AL}(next(i)) \,|\!\}_{pk(next(i))} \\
\mathrm{MsgB}(i) &= \{\!|\, \mathrm{nlist}(i) \,|\!\}_{n(i+1)}
\end{aligned}$$

Using private keys. If secrecy of the nonces is not required, we can use the private key of the sender of a message for encryption, instead of the public key of the receiver. This gives the following protocol.

$$\begin{aligned}
\mathrm{MsgA}(i) &= \{\!|\, [n0,\ldots,ni], \mathrm{AL}(ri) \,|\!\}_{sk(r(i))} \\
\mathrm{MsgB}(i) &= \{\!|\, [n(i+1),\ldots,n(p-1)], \mathrm{AL}(ri) \,|\!\}_{sk(r(i))}
\end{aligned}$$

Figure 6.5 illustrates the four-party version of this protocol. Although this protocol is minimal in the number of messages, it is not minimal in the complexity of the messages. In the first message of role $r0$, e.g., we can take the role names outside the encryption operator. Although there are some other local optimizations, we prefer to present this more regular protocol. Finding such a protocol with minimal complexity of the messages is still an open question.

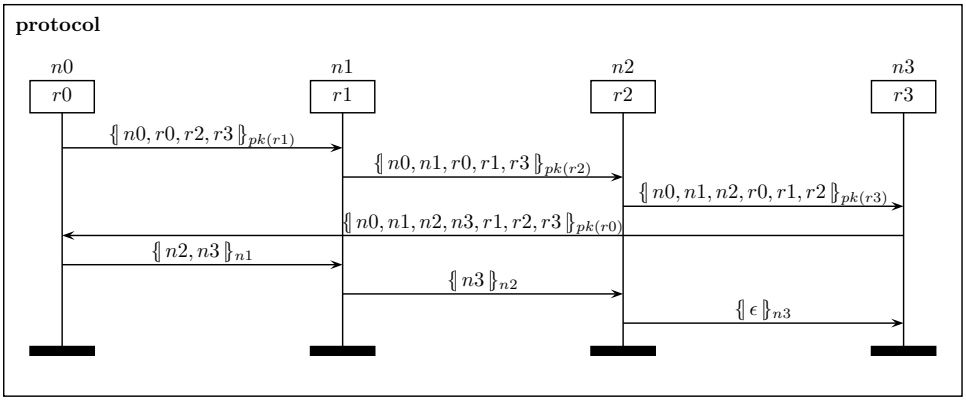

Figure 6.4: Four-party BKE protocol.

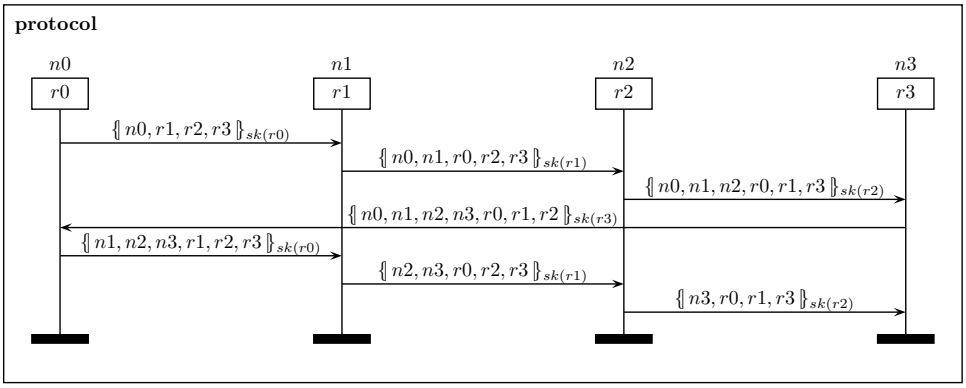

Figure 6.5: Four-party NSL private key protocol.

Rearranging message contents. In the proofs of correctness, we have used some (but not all) information that distinguishes the messages in the protocol. In particular, we used:

- *The ordered agent list AL().* We used this to derive the parameter p from an incoming message, and the order in the list is required to be able to derive that the agent list of the sender is identical to the agent list of the recipient.
- *The list of nonces.* We used the number of nonces to derive the role an agent is supposed to assume (given p).

A direct consequence of this is that the exact order of the agent list and nonce list is not relevant. We could e.g. redefine messages of type A as to start with a reversed list of roles, followed by the list of nonces.

As a second observation, we note that besides the distinct type of each message, the proof did not depend on the fact that there is no other content inside the

encryption besides nonces and agent names. Thus, we can add any payload inside the encryption, as long as we ensure that it cannot be confused with an agent term or a nonce.

This opens up several possibilities for establishing e.g. keys between pairs of agents inside of the generalized NSL protocol. Next, we discuss one such option.

Key agreement protocols. In the field of cryptographic protocols many, so-called, group key agreement protocols have been developed. Although these have different goals from the protocol mentioned here, we see some possibilities to use the underlying structure of the protocol for these purposes.

The generalized NSL presented here can be turned into a naive group key agreement protocol by deriving a session key using a hash function over all the nonces, e.g. $h(n(0)\dots n(p-1))$. This would constitute a fresh authenticated session key, which is shared by all the participants. However, the resulting protocol would not satisfy e.g. *forward secrecy* of the session key. If one of the private keys of one of the participants is leaked after a session, the nonces that were used can be determined. From this the original session key can be retrieved, which allows an intruder to decrypt a session afterwards.

To establish forward secrecy of a session key, derivatives of the Diffie-Hellman key agreement protocol are used, as e.g. in [176]. We envisage that such an approach would be possible here as well, for example either by adding Diffie-Hellman derivatives as payload, or more efficiently, by replacing the nonces that are sent by the public halves of the Diffie-Hellman constructs.

6.4 Attacks on two previous multi-party authentication protocols

In this section we show why the two multi-party authentication protocols in [45] achieve their intended goals, but do not achieve agreement, and therefore not synchronisation. We want to mention explicitly that this was not a goal of the protocol designers. Their notion of authentication is what is also known as *recent aliveness*. From Definition 1 in the paper [45]:

> *We say that A authenticated B if there exists a bounded time interval I in the local time of A such that A is convinced that B was alive (i.e., sent some messages) in I.*

Recent aliveness is weaker than agreement in the sense that there are no requirements of the role that B is playing, or on any shared data, or that messages are sent and received correctly.

The protocols in [45] also differ in setup when compared to ours. Whilst we assume agents have private and public keys, they assume all agents share symmetric keys

before starting the protocol. Here we will write $k(r0, r1)$ to denote the key shared by the agent in role $r0$ and the agent in role $r1$.

In the paper we find two protocols. They both satisfy recent aliveness, which is established by a challenge-response loop, and the authors make sure no reflection attack (which can arise from using symmetric keys) is possible. However, neither of them satisfies agreement or synchronisation. In particular, there can be much confusion about who is playing which role. Here we show that for both protocols, the instances for three parties already do not satisfy stronger notions of authentication such as agreement and synchronisation.

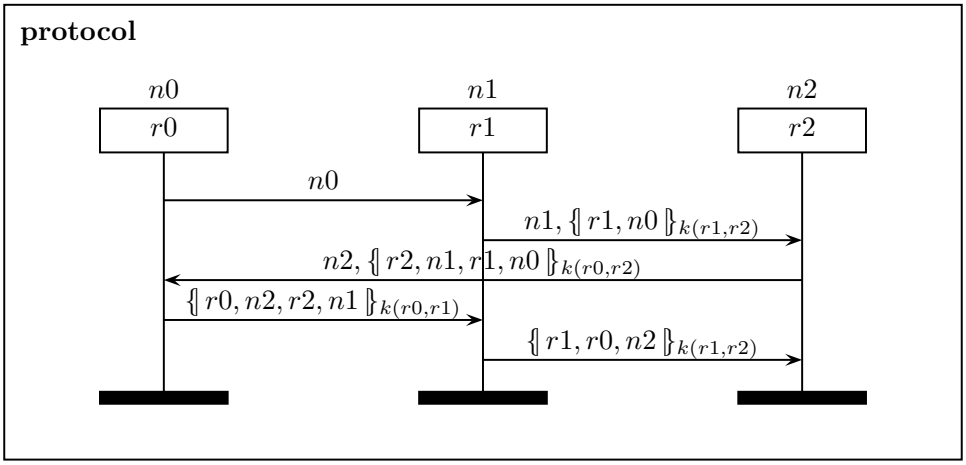

Figure 6.6: BNV Protocol 1, 3-party version.

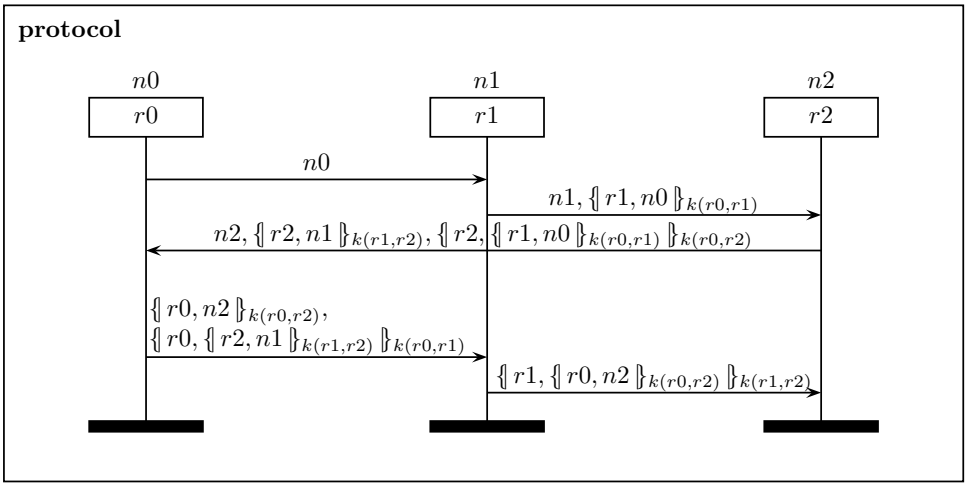

Figure 6.7: BNV Protocol 2, 3-party version.

The first protocol is shown in Figure 6.6, and we show an attack in Figure 6.8. In the attack, A completes role $r0$ up to the claim, after starting a session in which he wants to talk to B in role $r1$ and C in role $r2$. However, although B and C are indeed performing actions (guaranteed by the recent aliveness), it is clear that B performs role $r1$ thinking he is talking to D, and not to A. Furthermore, A ends with two nonces, Nonce1 and Nonce2 which can both be determined by the intruder. Clearly, there is no strong authentication property that is satisfied here.

The second protocol is shown in Figure 6.7, and we show an attack in Figure 6.9. In this particular attack, A again starts a session with B and C. At the end of the attack, there is no one performing role $r1$. B thinks A is executing role $r1$, and C thinks F is executing role $r1$, neither of which is true. Furthermore, A ends with a nonce Nonce2 which can be determined by the intruder.

We therefore conclude that neither one of the two protocols from [45] satisfies agreement or synchronisation.

6.5 Conclusions

We proposed a security protocol for multi-party authentication and proved it correct, i.e. the protocol satisfies injective synchronisation and all nonces are secret. The proof does not require a fixed number of parties p. This is in line with more recent attempts (e.g. [175]) to develop methodologies for such (parameterised) multi-party protocols, for which this protocol could be used as a case study.

In particular, the proof establishes that the generalized protocol satisfies the same type of security requirements as the original Needham-Schroeder-Lowe protocol, but now for any number of parties.

Correctness of the generalized protocol is subject to the assumption that the messages include enough information as to allow a receiving agent to check if a message is correctly typed, and correctly split into subterms.

As has been shown by history, constructing correct security protocols is not trivial. Even knowing this, we were surprised to find that all variants of the proposed protocol (irrespective of the ordering of nonces and role names in the messages) suffer from type-flaw attacks. We found this out by using the Scyther tool developed in this thesis. In fact, we extensively used this tool to investigate instances of the protocol for a specific number of participants to guide us in our research and to study the variations presented in Section 6.3. A simple (and standard) extension of the messages will make the protocol resilient against such type-flaw attacks.

The communication structure underlying the protocol can serve as a generic pattern for multi-party challenge-response mechanisms, in which we can capture generalized NSL, BKE, and several other variants. These generalized protocols can serve as efficient communication structures underlying multi-party authentication schemes as used in electronic commerce protocols.

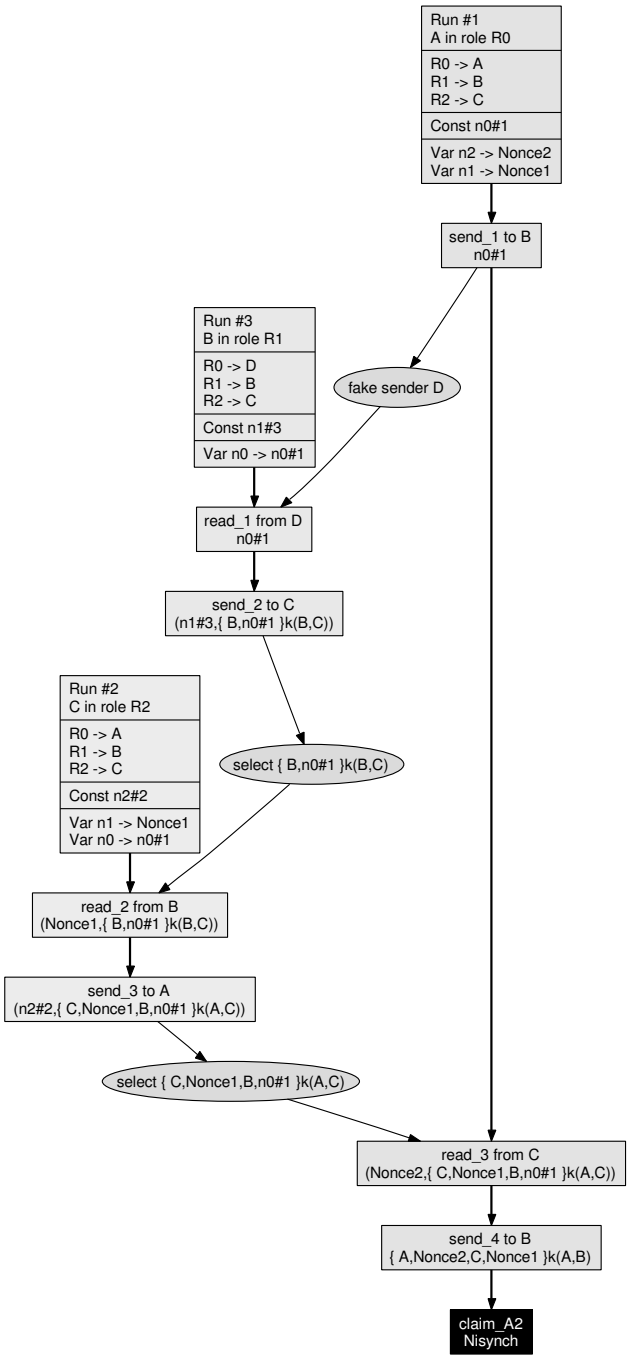

Figure 6.8: BNV Protocol 1, Attack on 3-party version.

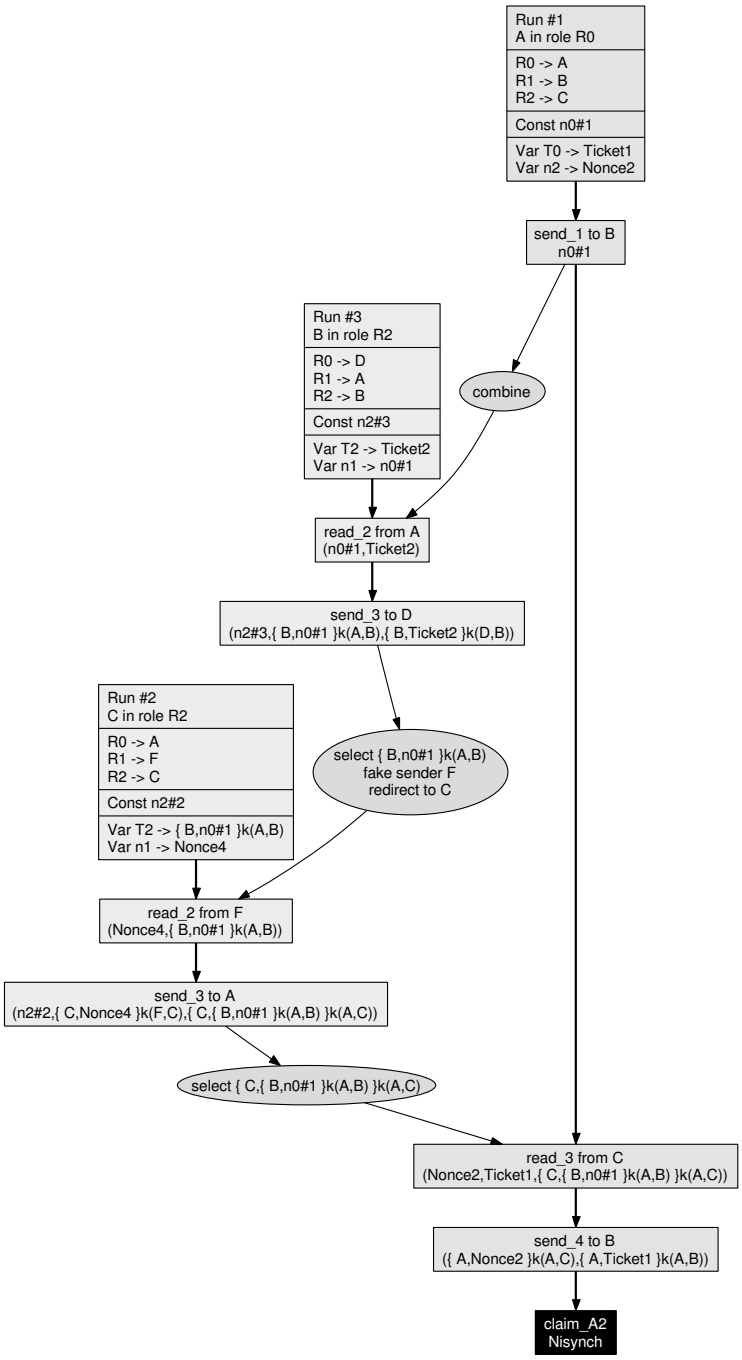

Figure 6.9: BNV Protocol 2, attack on 3-party version.

7

Related Work

In the previous chapters we have developed theoretical models and tools, which have been applied in various settings. In this chapter we discuss related work. During the last twenty years, much research has been performed in the area of black-box analysis of security protocols. We discuss security protocol models and security properties in Section 7.1. We proceed by addressing related work on some recent verification methods in Section 7.2. We close off by discussing other related work, including multi-protocol analysis and multi-party protocols in Section 7.3.

7.1 Security protocol models

There is a wealth of different approaches for the black-box modeling of security protocols. Very often the focus is on verification tools, where the underlying model is only informally or implicitly defined.

7.1.1 Current models

We will briefly compare our approach to some prominent approaches: BAN logic (because of its historic interest), Casper/FDR (because it has powerful tool support), Strand Spaces (because this approach has much in common with ours), the Protocol Composition Logic (as a recent promising development), and the Spi calculus (as a process calculus based approach). This selection is by no means exhaustive, but is meant to be representative for the approaches that are currently used.

BAN logic

In 1989 Burrows, Abadi and Needham published their ground-breaking work on a logic for the verification of authentication properties [44]. In this so-called BAN-logic, predicates have the form "P believes X". Such predicates are derived from a set of assumptions, using derivation rules like "If P believes that P and Q share key K, and if P sees message $\{\!|\, X \,|\!\}_K$ then P believes that Q once said X". Note that this rule implies a peculiarity of the agent model, which is not required in most other approaches, viz. an agent can detect (and ignore) his own messages. The BAN-logic has a fixed intruder model, which does not consider conspiring agents. The

Needham-Schroeder protocol was proven correct in BAN-logic because the man-in-the-middle attack could not be modeled. Another major difference with our approach is that the BAN-logic uses a rather weak notion of authentication. The authentication properties verified for most protocols have the form "A believes that A and B share key K" (or "... share secret X"), and "A believes that B believes that A and B share key K". This weak form of agreement is sometimes even further reduced to *recent aliveness*. An interesting feature is that BAN logic treats time stamps at an appropriate abstract level, while an extension of our semantics with time stamps is not obvious. Due to the above mentioned restrictions interest in BAN logic has decreased. Recent research concerns its extension and the development of models for the logic, e. g. [2, 146, 178, 177].

Casper/FDR

Developed originally by Gavin Lowe, the Casper/FDR tool set as described in [121] is not a formal security protocol semantics, but a model checking tool. However, as the input is translated into a CSP process algebraic model, there is an implicit semantics. The reason we mention it here, is that Casper/FDR is a mature tool set, and none of the other semantics we mention has such a tool set available. For Casper/FDR many interesting security properties have been formulated in terms of CSP models (see e.g. [122]) and some of these have been consequently adapted in other models. However, for Casper/FDR there is no explicit formal semantics of the protocol language and properties except in terms of CSP. Casper/FDR has been used for case studies on many protocols as reported e. g. in [80].

Strand Spaces

The Strand Spaces approach [182] is closely related to the use of Message Sequence Charts which we advocate for the description of protocols and protocol runs. The approach provides an elegant way to reason about protocol executions in the presence of a Dolev-Yao intruder, and has been used to prove many theoretical results [96, 181, 95, 78, 180] and is the basis of several tools [140, 57, 173]. Roughly, the main difference is that we provide a totally ordered formal semantics that makes the relation between a protocol and its behaviour explicit, whereas Strand Spaces describe protocol execution using a partial order on the events. The notion of a strand is similar to our notion of run, and a strand space is the set of all possible combinations of strands, reflecting our semantical model of interleaved runs. The notion of a bundle closely corresponds to our notion of realizable trace pattern, and allows one to effectively reason about classes of traces. In Strand Spaces, a protocol is modeled as a collection of (parameterized) strands, with additional constraints that imply which parameters denote locally generated values (nonces), and which denote variables. Strand Spaces seem to be very tightly linked to the Dolev-Yao intruder model and although the intruder is modeled as a collection of strands, just like normal agents, it is not easy to vary over the intruder network model. With respect to the security properties, we mention that both secrecy and agreement

are expressible in the Strand Spaces model, but no immediate mechanism exists to formalize injectivity or individual order on events, as required for injective synchronisation. Another difference with the operational semantics presented here is the way in which some subtleties are handled, e. g. nonce generation, the handling of initial intruder knowledge and trusted/untrusted agents. In our model, such concepts are explicit and solved at a syntactical level where possible, whereas in the Strand Spaces model these are often formalized as additional requirements on e. g. nonce occurrence (that somehow exist outside the base model), or encoded as the parameters of a strand. In the same vein, there is no strict distinction between local constants and variables within a strand. Thus, it seems that Strand Spaces is mainly focused on describing the protocol execution, and only implicitly concerned with protocol description. As a result, the exact relation between the protocol and its execution is only implicitly defined.

Protocol Composition Logic

A more recent example of a security protocol methodology is the Protocol Composition Logic described in [73]. In this model, protocols are described using a formalism called the cord calculus [83]. It is accompanied by a BAN-like logic which can be used to establish correctness of protocol properties, but which can also be used to reason about the composition or refinement of protocols. Currently, there is no automated tool support for this logic, but many case studies have been performed manually, in particular ones concerning protocol composition in [71, 97, 98], which are related to the large-scale studies performed here in Chapter 5. An interesting feature of the logic is the way in which it can be used to derive classes of protocols, as in [72].

Spi calculus

As an example of a process calculus approach, we have the Spi calculus developed by Abadi and Gordon in [1]. It is an extension of the pi calculus in [141], and has been succesfully applied for the analysis of security protocols, e. g. as in [88]. The Spi calculus shares many advantages with the pi calculus, such as e. g. having a well-defined semantics. It also inherits properties of the pi calculus that do not immediately seem useful for security protocol analysis. As an example, expressing that a run is synchronising with another run over multiple messages is non-trivial, because it can be hard to tell two runs of the same role (with identical parameters) apart. To always be able to distinguish two runs, additional constructs are needed as in [36]. Having an explicit run identifier in the semantics makes it easier to express such properties. Nevertheless, the Spi calculus has been successfully used in the area of web services [27] in combination with the ProVerif tool [29], which has led to the development of the TulaFale security tool for Web Services [28].

In the methods mentioned here, except for the Protocol Composition Logic, the relation between the concept of (un)trusted agents and the intruder knowledge is left implicit.

7.1.2 Modeling security properties

Each security protocol model has its own distinct means for defining security properties. Furthermore, specifying security properties is a research area in itself, and models and logic have been developed with the specific purpose of modeling security properties, such as e. g. [59, 86, 165].

Looking into more detail at specific security properties, we find that secrecy is handled in a similar way by the majority of models. For authentication properties, the situation is more complicated. Historically, many different interpretations of authentication exist for communication protocols. Early authentication concepts simply mention that one "end" of a communication channel can assure itself regarding the identity of the other end, as in e.g. [145, 159]. An identity is considered to be an "end". These concepts seem very similar to Lowe's later definition of *aliveness* in [122]. For this form of authentication, it is only required that the party to be authenticated performs some action to prove his identity (i.e. applying his secret key) regardless of the context, or whom he is proving his identity to. This is a rather weak form of authentication.

Some more advanced notions of authentication can be found in [113], for example, where mechanisms are sketched to prevent *message stream modification*. This can be prevented by achieving three subgoals: determine message authenticity, integrity, and ordering. Although no formalisation is provided, this concept is akin to our notion of non-injective synchronisation.

The international standard ISO/IEC 9798-1 [87] states that authentication requires verification of an entity's claimed identity. In response, Gollmann points out in [90] that the concept of a sender of a message should be treated with caution, and that replays should be prevented. Gollmann argues that authentication of an entire communication session is done by first setting up a session key, and that further messages are authenticated on the basis of this key. Based on this assumption, Gollmann identifies four authentication goals. These goals explicitly assume that the protocols are implemented using private keys and session keys. Our definition of injective synchronisation is independent of protocol details, though. Therefore, it can be applied to a wider range of protocols.

An alternative formulation of authentication is given by Diffie, Oorschot, and Wiener in [76]. Here, participants are required to have matching message histories. The message history is allowed to be partial, if the last message sent is never received, which corresponds to our notion of messages that *causally precede* a claim. This notion corresponds closely to non-injective agreement.

Intensional specifications

Roscoe introduces in [161] intensional specifications. These can be viewed as authentication of communications. The notion of injectivity does not seem to play a role in Roscoe's definition. Besides further research by Roscoe et al, there have been few attempts at formalising intensional forms of authentication. A notable

exception is the definition of authentication by Adi, Debbabi and Mejri in [3]. Their authentication property requires a strict order on the messages. Furthermore, injectivity of the runs is required.
The authentication definition of [3] differs from injective synchronisation on two main points. First, it has a parameter, consisting of the set of communications to be authenticated. In our work, this parameter is fixed: it is defined as the set of communications that causally precede the claim event. We argue that this choice results in the strongest possible form of authentication. If the parameter is chosen to be a proper subset of the causally preceding communications set, it can be shown that the authentication is strictly weaker than injective synchronisation for the normal intruder model.
A second difference is that the authentication definition is strictly tailored for protocols involving two parties that communicate directly with each other. Thus, it cannot straightforwardly be used to express that two parties authenticate each other when they only communicate via e.g. a server. Also, it is not clear how it generalises to multi-party settings.

Extensional specifications

In [122], Lowe introduces an entire hierarchy of extensional specifications, corresponding to authentication of data. This builds on earlier work of [76] and [90], resulting in four different forms of authentication, viz. aliveness, weak agreement, non-injective agreement and injective agreement. On top of this, agreement on subsets of data items and recentness are considered (two topics we do not address here). In the course of time many subtly different extensional authentication properties have been proposed. Most of these derive directly from the work by Lowe.

In [38], Boyd proposes an alternative hierarchy of extensional goals for authentication protocols, which are oriented towards goals regarding established keys as well as the end results for the user. Similar to Gollmann, Boyd assumes that authentication is comprised of session key establishment and further communications being authenticated through the use of this key.

Authentication and agreement are also studied in [86] by Focardi and Martinelli in the context of the so-called GNDC scheme. In a process algebra extended with inference systems reflecting cryptographic actions, one can reduce reasoning about security properties with respect to any arbitrary environment to the analysis of the behavior of the security protocol in the most general environment. It is argued that the GNDC scheme is valid for establishing various security properties, in particular agreement (as well as its weaker variants). In [85], Focardi and Martinelli recast Lowe's notion of agreement in the GNDC scheme, and show that it is strictly stronger than two other notions of authentication: GNDC-authentication and spi-authentication from [1]. This implies that the latter two notions are also strictly weaker than injective synchronisation.

Injectivity

With respect to the analysis and verification of injectivity, we note that most approaches implement Lowe's definition of injectivity by way of a *counting* strategy: in any possible execution of the protocol the number of initiator runs may not exceed the number of corresponding responder runs. This counting argument can easily be used in a model-checking approach. Indeed, this is how injectivity is verified in the Casper/FDR tool chain [121, 164]. Since it is only possible to model a finite and fixed number of scenarios, this approach will only provide an approximation of injectivity. Other approaches to the verification of security protocols, e.g. those based on logics (such as [44]) or on term rewriting (such as [89]) do not consider injectivity. The Strand Spaces [182] approach does not provide formal means to deal with injectivity. Instead, it is proposed to check authentication based on nonces, for example by using so-called *solicited authentication tests* as defined in [96]. These tests guarantee injectivity, based on nonce freshness. Authentication and injectivity are strongly connected in this view. In the HLPSL framework used by the Avispa tool set [104], the definition of agreement includes a parameter, typically a nonce, over which injectivity is established. For this approach, the injectivity is strongly connected to the notion of freshness of the parameter. If the correct parameter is chosen, this can be used to verify that injective agreement holds. A drawback of this method can be that it is focused on the Dolev-Yao intruder model: for weaker intruder models, injective agreement might hold even when there is no fresh value to agree upon, which cannot be captured by this notion of agreement.
We mention two examples of security protocol formalisms that deal explicitly with injectivity. Gordon and Jeffrey have devised a method [106] to verify injective correspondence relations for the π-calculus. This method allows for verification by type-checking of (injective) correspondences. A second example can be found in the protocol logic by Adi, Debbabi and Mejri [3], where pattern matching is used to express injectivity for two-party protocols. However, it is not clear how verification can be done efficiently.

7.1.3 Complete characterization

As stated in Section 4.1.4, we owe the term *complete characterization* to independently developed work of Dogmi, Guttman and Thayer [78]. They develop an algebraic theory of skeletons and shapes: skeletons are comparable to trace patterns without intruder events, and shapes are similar to to explicit trace patterns without intruder events. A characterization is defined as the set of shapes of a skeleton. Given a skeleton, they sketch a procedure by which the set of shapes may be determined. The sketched procedure exploits the idea of authentication tests [96] to determine the set of shapes. Interestingly, for characterization of the Needham-Schroeder responder role, they determine exactly one shape, which captures the correct execution and the attack at the same time, where in our approach we find two explicit trace patterns. This is a result of their choice to exclude intruder events: in terms of shapes, the normal behaviour is a special case of the attack. In terms of explicit trace patterns, the attack and the normal behaviour are not instances of

each other, as one strictly requires decrypt and encrypt events, and the other does not.

7.2 Protocol analysis tools

The current generation of security protocol analysis methods works with abstract black-box security protocol models that are based on abstractions (of computer networks and cryptographic operations). Even with these simplifying abstractions, the problem of whether a protocol actually provides the security properties it has been designed for is undecidable [81]. Despite this fact, over the last two decades a wide variety of security protocol analysis tools have been developed that are able to detect attacks on protocols or, in some cases, establish their correctness.

Theorem proving versus model checking

Current protocol verification tools combat undecidability in different ways. Automatic tools either sacrifice soundness (providing false positives) or completeness (missing some attacks), or simply may not always terminate. Alternatively, tools may be interactive and require user guidance, for example, to construct interactively a proof that the protocol is correct. We address these two main types of tool here. In the next section, we survey some recent tools, briefly highlighting the principles on which they are based.

Theorem Proving One type of mechanized verification process is theorem proving using a higher-order logic theorem prover such as Isabelle/HOL [149] or PVS [147]. Using a theorem prover, one formalizes the system (the agents running the protocol along with the attacker) as a set of possible communication traces. Afterwards, one states and proves theorems expressing that the system in question has certain desirable properties, i. e. each system trace satisfies properties such as authentication or secrecy. The proofs are usually carried out under strong restrictions, e. g. that all variables are strictly typed and that all keys are atomic.

The approach has been applied now to over a dozen protocols. These include simple authentication protocols like the Needham-Schroeder-Lowe public-key protocol and the Otway-Rees protocol [152], key distribution protocols like Yahalom [154], multi-party protocols like recursive authentication protocols [152], and non-repudiation protocols like Zhou-Gollmann [22]. Moreover, the method has been applied to a number of industrial-strength protocols. These include Kerberos Version 4 [20, 21], a version of SSL/TLS [153], and parts of SET [19, 155, 16, 17, 18].

The main drawback of this approach is that verification is quite time consuming and requires considerable expertise. Moreover, theorem provers provide poor support for error detection when the protocols are flawed.

Model Checking The second kind of verification centers around the use of model checkers [121, 57, 52, 143, 133, 173, 30], which are fully automatic. We distinguish three classes: tools that attempt verification (proving a protocol correct), those that attempt falsification (finding attacks), and hybrids that attempt to provide both proofs and counterexamples.

The first class of tools, which focus on verification, typically rely on encoding protocols as Horn clauses and applying resolution-based theorem proving to them (without termination guarantee). Analysis tools of this kind include NRL [133] and ProVerif [30].

In contrast to verification, the second class of tools detects protocol errors (i. e. attacks) using model checking [121, 143] or constraint solving [52, 57]. Model checking attempts to find a reachable state where some supposedly secret term is learnt by the intruder, or in which an authentication property fails. Constraint solving uses symbolic representations of classes of such states, using variables that have to satisfy certain constraints (e. g. the term $\{\!|\, m \,|\!\}_k$ must be inferable from the intruder knowledge). To ensure termination, these tools usually bound the maximum number of runs of the protocol that can be involved in an attack. Therefore, they can only detect attacks that involve no more runs of the protocol than the stated maximum.

In the third class, attempts to combine model checking with elements from theorem proving have resulted in backward-search-based model checkers. These use pruning theorems, resulting in hybrid tools that in some cases can establish correctness of a protocol (for an unbounded number of sessions) or yield a counterexample, but for which termination cannot be guaranteed [173]. Scyther falls into this last category, but is guaranteed to terminate.

Tools

Because the model checking approach has proven very successful in the last years, the vast majority of existing tools is based on model checking rather than on theorem proving. We briefly describe some of the basic ideas behind the most recent tools. The selection presented here is by no means exhaustive; rather, we have tried to include representatives of each type of verification method.

Some earlier approaches rely on general purpose verification tools:

Isabelle. Paulson [152] has proposed to state security properties such as secrecy as predicates (formalized in higher-order logic) over execution traces of the protocol, without limitations on the number of agents. These predicates can be verified by induction with automated support provided by the Isabelle proof assistant [149].

Casper/FDR. FDR [124] is a model checker for the CSP process algebra. Roughly speaking, FDR checks whether the behaviors of a CSP process associated with a protocol implementation are included in the behaviors allowed by its specification. FDR is provided with a user-friendly interface for security protocol

analysis, Casper [121] that automatically translates protocols in an "Alice & Bob-notation" (with possible annotations) to CSP code. Gavin Lowe has discovered the now well-known attack on the Needham-Schroeder Public-Key Protocol using FDR [119].

Similarly, many protocol-specific case studies have been performed in various general-purpose model checkers. We mention μCRL [94] as used in [33], UPPAAL [25] as used in [58], and SPIN [103] as used in [126].

Other tools are completely dedicated to security protocols:

NRL. In the NRL Protocol Analyzer [133], the protocol steps are represented as conditional rewriting rules. NRL invokes a backward search strategy from some specified insecure state to see if it can be reached from an initial state. It has been used for verification of e. g. the Internet Key Exchange protocol [134]. Unfortunately, NRL is not publicly available.

Athena. The Athena [173] tool is an automatic checking algorithm for security protocols. The algorithm described in [172] served as a starting point for the development of Scyther. It is based on the Strand Spaces model [182] and, when terminating, provides either a counterexample if the formula under examination is false, or establishes a proof that the formula is true. Alternatively, Athena can be used with a bound (e. g. on the number of runs), in which case termination is guaranteed, but it can guarantee at best that there exist no attacks within the bound. Unfortunately, Athena is not publicly available.

ProVerif. In ProVerif [30], protocol steps are represented by Horn clauses. The system can handle an unbounded number of sessions of the protocol but performs some approximations (on random numbers). As a consequence, when the system claims that the protocol preserves the secrecy of some value, this is correct; however it can generates false attacks too. Recently an algorithm was developed [4] that attempts to reconstruct attacks, in case the verification procedure fails, adding the possibility of falsification to ProVerif.

LySatool. The LySatool [34] implements security protocol analysis based on a process algebra enhanced with cryptographic constructs. The approach is based on over-approximation techniques and can verify confidentiality and authentication properties.

Constraint solver. Based on [140], in which verification in the Strand Spaces model is translated into a constraint solving problem, an efficient constraint solving method was developed in [57]. The method uses constraint solving, optimized for protocol analysis, and a minimal form of partial order reduction, similar to the one used in [56]. A second version of this tool does not use partial order reduction, enabling it to verify properties of the logic PS-LTL [59].

OFMC. The On-the-Fly Model Checker (OFMC, [14]) is part of the AVISPA tool set [7], and is a model checker for security protocols. It combines infinite

Tool name	publicly available	falsification	verification		termination
			bounded	unbounded	
Casper/FDR	yes	yes	yes	no	yes
ProVerif	yes	yes[1]	no	yes	yes
Constraint solver	yes	yes	yes	no	yes
LySatool	yes	no	no	yes	yes
OFMC	yes	yes	yes	no	yes
Scyther	yes	yes	yes	yes	yes
NRL	no	no	no	yes	yes
Athena (bounded)	no	yes	yes	no	yes
Athena (unbounded)	no	yes	no	yes	no

Table 7.1: A comparison of automatic protocol verification tools in relation to falsification, verification and termination

state forward model-checking with the concept of a lazy intruder [11], where terms are generated on-demand during the forward model-checking process. A technique called constraint differentiation [12] is employed to avoid exploring similar states in different branches of the search, which is similar to the ideas in [64]. It furthermore supports user-defined algebraic theories [13], allowing for correct modeling of e. g. Diffie-Hellman exponentiation.

In Table 7.1 we give a brief overview of the automatic tools mentioned here. Of the tools mentioned here, only NRL and Athena are not publicly available, shown in the second column of the table. The third column shows whether a tool can perform falsification, i. e. whether or not it can find attacks. Finding attacks can be useful for understanding or repairing a protocol, as an attack often suggests what the nature of the problem with the protocol is. The next column, verification, reports which tools can ensure that a protocol is (correct) within the model. We distinguish two cases. Some tools can perform unbounded verification: for some protocols, they can establish that the property holds within the security protocol model of the tool. Other tools (typically model-checking based approaches) can only perform bounded verification: when such a tool finds no attack, this means that there is no attack *within a certain bound*, e. g. there is no attack which involves less than three runs. The final column indicates whether the verification procedure of the tool is guaranteed to terminate.

A more subtle advantage of Scyther over the majority of existing tools is that there is no need to specify so-called *scenarios*. A scenario is commonly used for model-checking methods, and is a way to limit the state space, and is thus comparable to our concept of maximum number of runs.[2] In most cases (as e. g. in Casper/FDR), one tailors the scenario in such a way that it closely resembles the attack, to reduce

[1] Recently a attack reconstruction tool was introduced for ProVerif that in many cases can reconstruct an attack after a failed verification attempt, in [4].

[2] In OFMC it is possible to define symbolic session instances, which are solved during search. These can be used in a similar way as the maximum number of runs, thus avoiding the specification of scenarios.

verification time (or even to make verification feasible). For some tools, the scenarios even partially encode the security properties that one wants to verify (e. g. the constraint solver, Athena). The creation of such scenarios is subtle and error-prone. If set up incorrectly, the result can be that attacks are missed.

Although Scyther has many advantages over existing methods, it currently has a smaller scope than some of the other tools in the table. In particular, Scyther currently cannot handle Diffie-Hellman exponentiation or the use of the exclusive-or operator. Thus, for protocols that include such operations, which are not yet supported by Scyther, other state-of-the-art tools such as the AVISPA tool set would be an appropriate choice.

7.3 Other related work

With regard to multi-protocol attacks, there is only very limited related work. The concept of multi-protocol attacks originates in [111]. Given a correct protocol, a procedure is sketched to construct a second protocol. Both protocols can be shown to be correct in isolation, but when these two protocols are executed over the same network, the intruder can use messages from one protocol to attack the other protocol. An analysis of the interaction between sub-protocols of a single protocol was performed by Meadows in [134]. Some detailed examples of multi-protocol attacks for public-key infrastructures have been given in [5]. For a particular class of attacks (so-called guessing attacks) the multi-protocol attack concept was further refined in [128], where preventive measures are also discussed.

With respect to preventing multi-protocol attacks, we observe that most countermeasures that prevent type-flaw attacks as in [99], introduce a mechanism to ensure that messages are sufficiently distinct. These mechanisms effectively add distinct tags to each message, in order to ensure that a message sent from a send event with label ℓ can only match with the pattern of the read event with the same label ℓ. If such tags are unique throughout a collection of protocols as suggested in [62], the "disjoint encryption" conditions suggested by Guttman and Thayer in [95] are met. Consequently, protocols cannot interfere with each other, preventing multi-protocol attacks.

Alternative requirements addressing protocol interference in various settings are found in [93, 48, 47].

Regarding multi-party authentication, we find that much research has been performed in the cryptographic setting, e. g. [9, 8, 41]. As a result, these types of analysis usually only address the problem of a static adversary, and do not consider the dynamic behaviour of the protocol and the adversary. These protocols are typically assumed to employ a multicast primitive, and based on this primitive their complexity can be analyzed, as in e.g. [109, 137]. Unfortunately the protocols in this category are designed to meet different goals than the protocols presented here, and are therefore difficult to compare to our approach.

In black-box analysis of security protocols, protocols usually consist of two or three

roles only. All the tools mentioned previously assume the protocols have a fixed number of participants, and are not suited for analysis of parameterized protocol specifications. Therefore, they cannot be used to analyze multi-party protocols in general, but they can be used to analyze specific instances of such protocols. For example, Proverif [29] has been used to analyze instances of the GDH protocols from [9]. In spite of the success of the black-box analysis methods, few multi-party protocols have been constructed in this setting. As a notable exception we would like to mention [45], where the authors construct two challenge-response protocols for any number of parties. However, the protocols described there do not satisfy synchronisation or agreement, as was shown in Section 6.4.

On the borderline between the cryptographic approach and the formal Dolev-Yao approach, the Bull protocol from [43] provides an interesting example. This protocol is flawed, as shown in [163], although a more abstract version was shown to be correct in [150].[3] Unlike the generalized NSL protocol, this protocol is based on agents sharing a symmetric key with a server, and furthermore the server is involved in each session.

Recently, a corpus of multi-party protocols have been established as part of the Coral project [175], aiming to establish a reference set for multi-party protocol analysis.

Regarding proving authentication protocols correct, there have been some recent attempts to simplify such proofs. For example, one successful approach is to use static analysis of the protocol to prove authentication properties, as described in e.g. [42]. However, the notions of authentication used there are weaker than synchronisation or agreement, and the methods used there do not seem suitable for proving synchronisation.

Another approach to proving authenticity is taken in [50], where a simple logic is developed to prove authentication, assuming that some secrecy properties hold. The idea is then to delegate the proof of the secrecy properties to a dedicated secrecy logic. While a promising idea, in this case we have seen that the lemmas used to prove secrecy (i.e. yielding a sequence of send events) are also used to prove authenticity: thus, in this particular case, the proof structure seems to suggest that splitting the proof strictly into two (one for secrecy, one for authentication) leads to duplication of a large part of the proof.

Finally, we like to mention the preliminary work related to the development of the models and tools in this thesis. In particular, the development of the semantics has been guided by conclusions drawn from the work in [148]. An early version of the Scyther tool was based on partial order reduction as developed in [64], using an algorithm similar to the one developed in [169]. Development of the Scyther tool has benefited from the work on attack analysis in [102], of which some heuristics have found their way into the tool.

[3] In this particular case, the cryptographic implementation differs in a significant way from the abstract version, which allows for new attacks: the exclusive or operator does not satisfy the properties required for black-box encryption, because under certain conditions it is possible to retrieve the key encrypted term.

8

Conclusions and Future Work

In this chapter we draw conclusions, summarize the contributions made in this thesis, and discuss some future work.

8.1 Conclusions

Detailed conclusions were drawn at the end of each chapter. Here, we return to the research question posed in the introduction:

Research question: *How to integrate developments in state-of-the-art black-box security protocol analysis into a formal semantics with tool support?*

As shown by this thesis, we can conclude the model presented here meets our requirements. Although some issues remain for future work (detailed in the next section), we have successfully integrated a novel analysis model with corresponding tool support. Both the model and the tool have contributed to new results.

The semantics give rise to authentication properties with a number of theoretical results, and have been used for the design and verification of multi-party authentication protocols. The tool has been used for research as well as education. It has been used for the verification of existing and new protocols, and has been used to obtain new results on multi-protocol attacks. In teaching, the semantics have been used as a model to explain security protocol analysis. The Scyther tool has been used to give students hands-on experience with protocol verification, and subsequently make them aware of some of the pitfalls of protocol design, by verifying existing protocols, or protocols they designed themselves.

In a practical sense, the work here (in the line of other black-box analysis results) shows that the design of correct security protocols is a difficult task. The current increase in electronic communications has resulted in the widespread application of security protocols. Our results with regard to multi-protocol analysis have a direct impact on current practices of internet protocols (internet protocols for payments or transmission of private data) and embedded systems design (protocols for smartcards, or cellular phones). For the protocols used in these areas, the problem of multi-protocol attacks has barely been addressed. The semantics and tools developed here can help to verify these protocols. The methodology can also be used to educate developers about the subtle problems of protocol design. As such, the work

here is at least useful in three phases: (1) during the education of developers, (2) during the protocol development process itself, and also (3) by enabling the process of assessment and certification of protocols.

The subjects in this thesis have been presented in the same order in which they have been researched. Work has started from a formal model, with intuitive basic concepts, and an operational semantics. This lead to the development of new security properties. Afterwards, tool support was developed which helped in validating the methodology. In a later stage, the case studies have produced new results, and serve as a further confirmation of the validity the model.

8.2 Summary of contributions

Some of the contributions made in this thesis are collected in Table 8.1 on the facing page, and discussed in more detail below.

In Chapter 2 a new security protocol model for defining security protocols and their behaviour was presented. The model makes protocol execution explicit by means of an operational semantics, defining the set of traces (possible behaviours) of a protocol. The result is a role-based security protocol model, with a parameterized intruder model. The initial intruder knowledge is derived from the protocol description, as opposed to being defined separately. Within the protocol model, security properties are modeled as local claim events.

The model was extended with the definitions of several trace-based security properties in Chapter 3. This included reformulation of existing notions of secrecy, as well as definitions of existing properties such as agreement and liveness. Furthermore, we introduced two new strong authentication properties: injective and non-injective synchronisation. The security properties have been related to each other by means of a hierarchy. A specific syntactic property, the LOOP property, was defined and a theorem was proven which states that non-injectively synchronising protocol roles that also satisfy LOOP, must also satisfy injective synchronisation.

Based on the security protocol model, we introduce the concept of trace classes in Chapter 4, which can be used to reason about sets of traces. We combine reasoning about traces with the basic ideas behind the existing Athena algorithm developed by Song. We develop an improved algorithm, which can be used to verify security properties or find attacks. The algorithm is semantically sound with respect to the security protocol model. We establish a link between the output of the algorithm and the notion of characterization as put forward by Dogmi, Guttman and Thayer. The new algorithm improves on the Athena algorithm in a number of ways. As a result of the new underlying semantics, security properties are defined as single events, as opposed to being encoded in error-prone scenarios, and the algorithm can cope with e. g. composed keys and multiple key structures. Contrary to the Athena algorithm, the new algorithm is guaranteed to terminate, whilst retaining completeness of the results for the vast majority of protocols.

We have implemented the algorithm in a prototype called *Scyther*. The performance

	Theoretical Contributions	Practical Contributions
Ch. 2	Novel security protocol model · formal trace semantics · parameterized over intruder model · initial intruder knowledge derived from role knowledge	(n.a.)
Ch. 3	Local security properties as claim events · novel definitions of existing properties · novel property: synchronisation Syntactic criterion for injectivity Hierarchy of authentication properties	(n.a.)
Ch. 4	Improved version of Athena algorithm · within operational semantics · termination is guaranteed, even when yielding complete results for an unbounded number of sessions · in terms of trace patterns · linked algorithm to characterizations · security properties as simple claim events (as opposed to error-prone verification scenarios) · verification of order-based properties Introduced a framework for reasoning about trace patterns	Developed tool support · semantically correct · state-of-the-art efficiency Heuristics choice analysis Bound choice analysis
Ch. 5	Novel definition of multi-protocol attacks	Found new attacks Identification of attack scenarios
Ch. 6	Linked minimal number of messages and injectivity Parameterized protocol proof	Development of a family of authentication protocols

Table 8.1: Contributions

of the tool is strongly connected to two parameters. The first is the heuristic (which was undocumented for the Athena algorithm). We investigated several options and have shown their influence on the performance. The second parameter is the bound on the number of runs, which is used to guarantee termination. We have investigated the consequences of particular choices for this parameter. The prototype has been in active use for over two years and has been successfully applied to both existing and new verification problems, and used for education.

In Chapter 5 we gave a definition of multi-protocol attacks. Subsequently, the Scyther tool was used for the automated analysis of multi-protocol attacks on a

set of protocols from literature. This has resulted in the discovery of several new attacks. From the tests, we have determined likely multi-protocol attack scenarios.

A further application of the model and tool were examined in Chapter 6, where a link was established between the minimal number of messages in a multi-party protocol, and injective synchronisation. The Needham-Schroeder-Lowe protocol was generalized to a family of multi-party authentication protocols, which are minimal in the number of messages, and which satisfy synchronisation. A parametrized proof was given for the correctness of the secrecy and synchronisation claims of the generalized version of Needham-Schroeder-Lowe. The developed protocols can serve as an efficient basis for multi-party synchronising protocols.

8.3 Future work

Although significant progress has been made, there is still a lot of work to be done. We discuss some future work related to semantics, verification and applications.

Semantics

The semantics presented here serve as a basis for further study. For the core of the model we have chosen the essential ingredients needed for security protocol analysis. Some interesting future work remains.

Strict formalisation in a theorem proving environment: The model presented in Chapter 2 was developed to serve as a core model for security protocol analysis, and make the relations between important concepts precise. However, given the complexity of the resulting model, a strict formalisation (in theorem proving environments such as PVS [147] or Isabelle [149]) is of interest, e. g. to prove that the model is consistent. In a second step, this would also facilitate interactive proofs of properties of the model.

Additional algebraic operators: There are mechanisms used by protocols that are not part of the core model yet, such as Diffie-Hellman exponentiation, and the exclusive-or operator.

Additional security properties: Here we have only presented some forms of secrecy and authentication. There are other security properties which we expect to be easily integrated into the model, such as e. g. non-repudiation.

Flow-control: For some protocols, it is convenient to express flow control operations such as branching and looping.

Time: Adding a notion of time to the model is an interesting future option. We conjecture that adding a useful notion of time to the model adds a significant amount of complexity to the semantics.

Extending the semantics should be a straightforward process for additional operators and security properties, and flow control.

Verification

Note that any changes in the semantics also have consequences for the verification process. We conjecture that it is possible to adapt the current algorithm for efficient verification of protocols that use Diffie-Hellman exponentiation and exclusive-or operations.

Verification of arbitrary trace-based security properties: The characterization algorithm is not specifically tailored to any security property, and can be used as a back-end by other tools for analysis of a large class of security properties. Scyther can be used to generate XML descriptions of the complete characterizations of trace patterns, which can be used as input for other verification tools. We expect that it is possible to efficiently verify formulas of certain security protocol logics using complete characterizations. A likely candidate would be e. g. PS-LTL as presented in [59].

Decidability and the unbounded algorithm: The characterization algorithm decides the security (i. e. establishes correctness for any number of runs, or finds an attack) for more than 90 percent of the tested protocols. For the remaining ten percent, the algorithm finds that there exists no attack within the bounds. This is in line with the undecidability of the problem in general. However, the algorithm structure can serve as a basis to prove decidability of certain properties for a subclass of protocols. If one can prove that the unbounded algorithm terminates for a certain set of trace patterns of protocols, this would imply that authentication properties are decidable for these protocols (and for which the algorithm is a decision procedure).

Applications

There are more possible applications of the methods developed here. We suggest some possible future applications.

Minimal protocols: In Chapter 6 we presented some protocols that used a minimal number of messages required for establishing multi-party injective synchronisation. Similarly, we conjecture that there are lower bounds on the number of messages required to satisfy any security goal for a given number of parties. Furthermore, minimality in messages is not the only possible notion of minimality. Alternatives include e. g. computational cost, or required storage. Although some of these concerns have been addressed for simple broadcast protocols in a cryptographic setting, we conjecture that it is more efficient to address these at the black-box abstraction level. Ultimately this can lead to establishment of a suite of protocols, all of which are provably correct. From this set of protocols, and given a minimality criterion, a developer can simply select the best protocol, instead of having to design a new (and possibly flawed) protocol.

Attack taxonomy: If we compare two different attack traces (of a particular claim in a protocol), they are usually similar. In fact, for the responder role of the Needham-Schroeder protocol, there were only two explicit trace patterns: the trace pattern capturing the normal execution of the protocol, and a trace pattern representing all

attacks (which are all instances of the man-in-the-middle attack). We conjecture that it is possible to define an equivalence relation on attacks, such that a taxonomy of all known attacks can be constructed, which would provide new insights in the relation between a protocol and its flaws.

To conclude: Security is a very difficult thing to get right, but also critical for many applications. The final conjecture is: Security cannot be solved by trial-and-error: each supposed improvement can result in a new fault. Security can only be solved by being very strict, something for which formal methods are very well suited.

Bibliography

[1] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The Spi calculus. *Information and Computation*, 148:1–70, 1999.

[2] M. Abadi and M. Tuttle. A semantics for a logic of authentication. In *Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 201–216, Montreal, 1991. ACM Press.

[3] K. Adi, M. Debbabi, and M. Mejri. A new logic for electronic commerce protocols. *Theoretical Computer Science*, 291:223–283, 2003.

[4] X. Allamigeon and B. Blanchet. Reconstruction of attacks against cryptographic protocols. In *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 140–154, Aix-en-Provence, France, June 2005. IEEE Computer Society.

[5] J. Alves-Foss. Multiprotocol attacks and the public key infrastructure. In *Proc. 21st National Information Systems Security Conference*, pages 566–576, Arlington, 1998.

[6] S. Andova, C.J.F. Cremers, K. Gjøsteen, S. Mauw, S.F. Mjølsnes, and S. Radomirović. Sufficient conditions for composing security protocols, 2006. In preparation.

[7] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, L. Cuellar, P.H. Drielsma, P. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proc. Computer Aided Verification'05 (CAV)*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.

[8] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 17–26. ACM Press, 1998.

[9] G. Ateniese, M. Steiner, and G. Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–639, 2000.

[10] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230. ACM Press, 2003.

[11] D. Basin. Lazy infinite-state analysis of security protocols. In R. Baumgart, editor, *Secure Networking: CQRE'99*, volume 1740 of *Lecture Notes in Computer Science*, pages 30–42. Springer, Berlin, 1999.

[12] D. Basin, S. Mödersheim, and L. Viganò. Constraint differentiation: A new reduction technique for constraint-based analysis of security protocols. In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 335–344. ACM Press, 2003.

[13] D. Basin, S. Mödersheim, and L. Viganò. Algebraic intruder deductions. In *Proc. Logic for Programming, Artificial Intelligence and Reasoning'05 (LPAR)*, volume 3835 of *Lecture Notes in Artificial Intelligence*, pages 549–564. Springer, 2005.

[14] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[15] K. Becker and U. Wille. Communication complexity of group key distribution. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 1–6. ACM Press, 1998.

[16] G. Bella, F. Massacci, and L.C. Paulson. The verification of an industrial payment protocol: The SET purchase phase. In V. Atluri, editor, *Proc. 9th ACM Conference on Computer and Communications Security*, pages 12–20. ACM Press, 2002.

[17] G. Bella, F. Massacci, and L.C. Paulson. Verifying the SET registration protocols. *IEEE Journal on Selected Areas in Communications*, 21(1):77–87, 2003.

[18] G. Bella, F. Massacci, and L.C. Paulson. An overview of the verification of SET. *International Journal of Information Security*, 4(1-2):17–28, 2005.

[19] G. Bella, F. Massacci, L.C. Paulson, and P. Tramontano. Formal verification of cardholder registration in SET. In F. Cuppens, Y. Deswarte, D. Gollman, and M. Waidner, editors, *Proc. 6th European Symposium on Research in Computer Security (ESORICS)*, volume 1895 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2000.

[20] G. Bella and L.C. Paulson. Using Isabelle to prove properties of the Kerberos authentication system. In H. Orman and C. Meadows, editors, *Workshop on Design and Formal Verification of Security Protocols*. DIMACS, 1997.

[21] G. Bella and L.C. Paulson. Kerberos version IV: Inductive analysis of the secrecy goals. In J.-J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann,

editors, *Proc. 5th European Symposium on Research in Computer Security (ESORICS)*, volume 1485 of *Lecture Notes in Computer Science*, pages 361–375. Springer, 1998.

[22] G. Bella and L.C. Paulson. Mechanical proofs about a non-repudiation protocol. In R.J. Boulton and P.B. Jackson, editors, *Proc. International Conference on Theorem Proving in Higher Order Logics (TPHOL)*, volume 2152 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 2001.

[23] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *Proc. 27th Annual ACM Symposium on Theory of Computing (STOC)*, pages 57–66. ACM Press, 1995.

[24] J. Benaloh, B. Lampson, D. Simon, T. Spies, and B. Yee. Private communication technology protocol. Internet draft, Microsoft Corporation, Redmond, WA, 10 1995.

[25] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — a tool suite for automatic verification of real–time systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer, 1995.

[26] S. Berezin. Extensions to Athena: Constraint satisfiability problem and new pruning theorems based on type system extensions for messages. `http://www.sergeyberezin.com/papers/athena-extensions.ps`, 2001.

[27] K. Bhargavan, C. Fournet, and A.D. Gordon. A semantics for web services authentication. *Theoretical Computer Science*, 340(1):102–153, 2005.

[28] K. Bhargavan, C. Fournet, A.D. Gordon, and R. Pucella. Tulafale: A security tool for web services. In *Proc. of the 2nd International Symposium on Formal Methods for Components and Objects (FMCS'03)*, volume 3188 of *Lecture Notes in Computer Science*, pages 197–222, 2004.

[29] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE Computer Society, 2001.

[30] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96, Cape Breton, June 2001. IEEE Computer Society.

[31] B. Blanchet. From secrecy to authenticity in security protocols. In *Proc. 9th International Static Analysis Symposium (SAS)*, pages 342–359. Springer, 2002.

[32] A. Bleeker and L. Meertens. A semantics for BAN logic. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[33] S.C.C. Blom, J.F. Groote, S. Mauw, and A. Serebrenik. Analysing the BKE-security protocol with μCRL. In *Proc. of the 6th AMAST Workshop on Real-Time Systems (ARTS 2004)*, volume 139 of *Electronic Notes in Theoretical Computer Science*, pages 49–90. Elsevier ScienceDirect, 2005.

[34] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Nielson. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390, 2005.

[35] C. Bodei, P. Degano, R. Focardi, and C. Priami. Authentication via localized names. In *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW)*, pages 98–110. IEEE Computer Society, 1999.

[36] C. Bodei, P. Degano, R. Focardi, and C. Priami. Primitives for authentication in process algebras. *Theoretical Computer Science*, 283(2):271–304, 2002.

[37] D. Bolignano. Towards a mechanization of cryptographic protocol verification. In *Proc. Computer Aided Verification'97 (CAV)*, volume 1254 of *Lecture Notes in Computer Science*, pages 131–142. Springer, 1997.

[38] C. Boyd. Towards extensional goals in authentication protocols. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[39] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer, 2003. ISBN: 3-540-43107-1.

[40] C. Boyd and J. Nieto. Round-optimal contributory conference key agreement. In *Proc. of the 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 161–174. Springer, 2003.

[41] E. Bresson, O. Chevassut, D. Pointcheval, and J.J. Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 255–264. ACM Press, 2001.

[42] M. Bugliesi, R. Focardi, and M. Maffei. Authenticity by tagging and typing. In *Proc. 2nd ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 1–12. ACM Press, 2004.

[43] J. Bull and D. Otway. A nested mutual authentication protocol. *Operating Systems Review*, 33(4):42–47, 1999.

[44] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[45] L. Buttyán, A. Nagy, and I. Vajda. Efficient multi-party challenge-response protocols for entity authentication. *Periodica Polytechnica*, 45(1):43–64, April 2001.

[46] C. Caleiro, L. Viganò, and D. Basin. Deconstructing Alice and Bob. *Proc. of the Workshop on Automated Reasoning for Security Protocol Analysis, ARSPA 2005*, 135(1):3–22, 2005.

[47] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `eprint.iacr.org/`.

[48] R. Canetti, C. Meadows, and P. Syverson. Environmental requirements for authentication protocols. In M. Okada, B.C. Pierce, A. Scedrov, H. Tokuda, and A. Yonezawa, editors, *Software Security – Theories and Systems, Mext-NSF-JSPS International Symposium, ISSS 2002, Tokyo, Japan, Revised Papers*, volume 2609 of *Lecture Notes in Computer Science*, pages 339–355. Springer, 2002.

[49] I. Cervesato, N.A. Durgin, P. Lincoln, J.C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW)*, pages 55–69. IEEE Computer Society, 1999.

[50] I. Cervesato, C. Meadows, and D. Pavlovic. An encapsulated authentication logic for reasoning about key distribution protocols. In *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 48–61. IEEE Computer Society, 2005.

[51] P.C. Cheng and V.D. Gligor. On the formal specification and verification of a multiparty session protocol. In *1990 IEEE Symposium on Security and Privacy*, pages 216–233. IEEE Computer Society, 1990.

[52] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In E. Brinksma and K.G. Larsen, editors, *Proc. Computer Aided Verification'02 (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 324–337. Springer, 2002.

[53] K.K.R. Choo, C. Boyd, and Y. Hitchcock. Errors in computational complexity proofs for protocols. Cryptology ePrint Archive, Report 2005/351, 2005. `http://eprint.iacr.org/`.

[54] J.A. Clark and J.L. Jacob. A survey of authentication protocol literature. `http://citeseer.ist.psu.edu/clark97survey.html`, 1997.

[55] E. Clarke, S. Jha, and W. Marrero. Partial order reductions for security protocol verification. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, pages 503–518. Springer, 2000.

[56] E.M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with Brutus. *ACMTSEM: ACM Transactions on Software Engineering and Methodology*, 9(4):443–487, 2000.

[57] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In M.V. Hermenegildo and G. Puebla, editors, *Proc. 9th International Static Analysis Symposium (SAS)*, volume 2477 of *Lecture Notes in Computer Science*, pages 326–341, Spain, Sep 2002. Springer.

[58] R. Corin, S. Etalle, P.H. Hartel, and A. Mader. Timed model checking of security protocols. In *Proc. 2nd ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 23–32. ACM Press, 2004.

[59] R.J. Corin, A. Saptawijaya, and S. Etalle. A logic for constraint-based security protocol analysis. In *2006 IEEE Symposium on Security and Privacy*, Los Alamitos, California, 2006. IEEE Computer Society.

[60] C.J.F. Cremers. The Scyther tool: Automatic verification of security protocols. `http://www.win.tue.nl/~ccremers/scyther/`.

[61] C.J.F. Cremers. Compositionality of security protocols: A research agenda. In F. Gadducci and M. ter Beek, editors, *Proc. of the 1st VODCA Workshop*, volume 142 of *Electronic Notes in Theoretical Computer Science*, pages 99–110. Elsevier ScienceDirect, 2004.

[62] C.J.F. Cremers. Feasibility of multi-protocol attacks. In *Proc. of The First International Conference on Availability, Reliability and Security (ARES)*, pages 287–294, Vienna, Austria, April 2006. IEEE Computer Society.

[63] C.J.F. Cremers, V. Issarny, and S. Mauw, editors. *STM'05, Proc. of the first international workshop on security and trust management*. Electronic Notes in Theoretical Computer Science. Elsevier ScienceDirect, Italy, September 2005.

[64] C.J.F. Cremers and S. Mauw. Checking secrecy by means of partial order reduction. In D. Amyot and A.W. Williams, editors, *SAM 2004: Security Analysis and Modelling*, volume 3319 of *Lecture Notes in Computer Science*, pages 177–194, Ottawa, Canada, September 2004. Springer.

[65] C.J.F. Cremers and S. Mauw. Operational semantics of security protocols. In S. Leue and T. Systä, editors, *Scenarios: Models, Transformations and Tools, International Workshop, Dagstuhl Castle, Germany, September 7-12, 2003, Revised Selected Papers*, volume 3466 of *Lecture Notes in Computer Science*. Springer, 2005.

[66] C.J.F. Cremers and S. Mauw. Generalizing Needham-Schroeder-Lowe for multi-party authentication, 2006. Computer Science Report CSR 06-04, Eindhoven University of Technology.

[67] C.J.F. Cremers, S. Mauw, and E.P. de Vink. Defining authentication in a trace model. In T. Dimitrakos and F. Martinelli, editors, *FAST 2003*, Proc. of the first international Workshop on Formal Aspects in Security and Trust, pages 131–145, Pisa, September 2003. IITT-CNR technical report.

[68] C.J.F. Cremers, S. Mauw, and E.P. de Vink. Formal methods for security protocols: Three examples of the black-box approach. *NVTI newsletter*, 7:21–32, 2003. Newsletter of the Dutch Association for Theoretical Computing Scientists.

[69] C.J.F. Cremers, S. Mauw, and E.P. de Vink. A syntactic criterion for injectivity of authentication protocols. In P. Degano and L. Vigano, editors, *ARSPA 2005*, volume 135 of *Electronic Notes in Theoretical Computer Science*, pages 23–38. Elsevier ScienceDirect, July 2005.

[70] C.J.F. Cremers, S. Mauw, and E.P. de Vink. Injective synchronisation: an extension of the authentication hierarchy. *Theoretical Computer Science*, 2006. In preparation.

[71] A. Datta, A. Derek, J.C. Mitchell, and D. Pavlovic. Secure protocol composition. In *Proc. 1st ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 11–23. ACM Press, 2003.

[72] A. Datta, A. Derek, J.C. Mitchell, and D. Pavlovic. Abstraction and refinement in protocol derivation. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 30–45. IEEE Computer Society, 2004.

[73] A. Datta, A. Derek, J.C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13(3):423–482, 2005.

[74] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[75] X. Didelot. COSP-J: A compiler for security protocols, 2003. `http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Casper/COSPJ/secu.pdf`.

[76] W. Diffie, P.C. van Oorschot, and M.J. Wiener. Authentication and authenticated key-exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[77] D. Dill. The Murϕ verification system. In *Proc. Computer Aided Verification'96 (CAV)*, volume 1102 of *Lecture Notes in Computer Science*, pages 390–393. Springer, 1996.

[78] S. Dogmi, J.D. Guttman, and F.J. Thayer. Skeletons and the shapes of bundles. `http://www.ccs.neu.edu/home/guttman/skeletons.pdf`, 2006.

[79] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(12):198–208, March 1983.

[80] B. Donovan, P. Norris, and G. Lowe. Analyzing a library of security protocols using Casper and FDR, July 1999.

[81] N. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[82] N.A. Durgin and J.C. Mitchell. Analysis of security protocols. In M. Broy and R. Steinbruggen, editors, *Calculational System Design*, pages 369–395. IOS Press, 1999.

[83] N.A. Durgin, J.C. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 241–272, 2001.

[84] A.G. Engels, S. Mauw, and M.A. Reniers. A hierarchy of communication models for Message Sequence Charts. *Science of Computer Programming*, 44(3):253–292, September 2002.

[85] R. Focardi, R. Gorrieri, and F. Martinelli. A comparison of three authentication properties. *Theoretical Computer Science*, 291(3):285–327, 2003.

[86] R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *World Congress on Formal Methods (1)*, volume 1708 of *Lecture Notes in Computer Science*, pages 794–813. Springer, 1999.

[87] International Organization for Standardization. Information technology - security techniques - entity authentication - part 1: General model. ISO/IEC 9798-1, September 1991.

[88] C. Fournet, A.D. Gordon, and S. Maffeis. A type discipline for authorization policies. In M. Sagiv, editor, *Proc. 14th European Symposium on Programming (ESOP)*, volume 3444 of *Lecture Notes in Computer Science*, pages 141–156, Edinburgh, UK, 2005. Springer.

[89] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proc. 19th International Conference on Automated Deduction (CADE)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 271–290. Springer, 2000.

[90] D. Gollmann. What do we mean by entity authentication. In *1996 IEEE Symposium on Security and Privacy*, pages 46–54. IEEE Computer Society, 1996.

[91] L. Gong. Variations on the themes of message freshness and replay—or the difficulty of devising formal methods to analyze cryptographic protocols. In *Proc. 6th IEEE Computer Security Foundations Workshop (CSFW)*, volume 3835, pages 131–136. IEEE Computer Society, 1993.

[92] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocl analysis. In *1990 IEEE Symposium on Security and Privacy*, pages 234–248. IEEE Computer Society, 1990.

[93] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Proc. of the 5th International Working Conference on Dependable Computing for Critical Applications (DCCA-5)*, pages 44–55, 1995.

[94] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes '94*, Workshops in Computing Series, pages 26–62. Springer, 1995.

[95] J.D. Guttman and F.J. Thayer. Protocol independence through disjoint encryption. In *Proc. 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 24–34. IEEE Computer Society, 2000.

[96] J.D. Guttman and F.J. Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

[97] C. He and J.C. Mitchell. Analysis of the 802.11i 4-way handshake. In *WiSe '04: Proc. of the 2004 ACM workshop on Wireless security*, pages 43–50. ACM Press, 2004.

[98] C. He, M. Sundararajan, A. Datta, A. Derek, and J.C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *Proc. 12th ACM Conference on Computer and Communications Security*, pages 2–15. ACM Press, 2005.

[99] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[100] N. Heintze, J.D. Tygar, J.M. Wing, and H.-C. Wong. Model checking electronic commerce protocols. In *Proc. USENIX 1996 Workshop on Electronic Commerce*, pages 147–166. USENIX Association, 1996.

[101] J.C. Herzog. The Diffie-Hellman key-agreement scheme in the strand-space model. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 234–247. IEEE Computer Society, 2003.

[102] G. Hollestelle. Systematic analysis of attacks on security protocols. Master's thesis, Technische Universiteit Eindhoven, Department of Mathematics and Computer Science, 2005.

[103] G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[104] FET Open Project IST-2001-39252. AVISPA: Automated validation of internet security protocols and applications, 2003.

[105] ITU-TS. *Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, 1999.

[106] A. Jeffrey and A. Gordon. Typing One-to-One and One-to-Many Correspondences in Security Protocols. In Mitsuhiro Okada, Benjamin C. Pierce, Andre Scedrov, Hideyuki Tokuda, and Akinori Yonezawa, editors, *Proc. ISSS '02*, volume 2514 of *Lecture Notes in Computer Science*, pages 418–434. Springer, 2002.

[107] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proc. 5th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society. `citeseer.lcs.mit.edu/burch90symbolic.html`.

[108] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*, volume 1163 of *Lecture Notes in Computer Science*, pages 36–49. Springer, 1996.

[109] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In *Proc. of Crypto'03*, volume 2729 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2003.

[110] C. Kaufman, R. Perlman, and M. Speciner. *Network security: private communication in a public world (2nd ed.)*. Computer Networking and Distributed Systems. Prentice Hall PTR, 2002.

[111] J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In B. Christianson, B. Crispo, T.M.A. Lomas, and M. Roe, editors, *Proc. 5th International Workshop on Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 1997.

[112] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7:79–130, 1994.

[113] S.T. Kent. Encryption-based protection for interactive user/computer communication. In *SIGCOMM '77: Proc. of the fifth symposium on Data communications*, pages 5.7–5.13. ACM Press, 1977.

[114] Y. Kim, A. Perrig, and G. Tsudik. Group key agreement efficient in communication. *IEEE Transactions on Computers*, 53, 2004.

[115] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[116] N.Y. Lee and M.F. Lee. Comments on multiparty key exchange scheme. *Operating Systems Review*, 38(4):70–73, 2004.

[117] Y.R. Lee, H.S. Sook, and H.K. Lee. Multi-party authenticated key agreement protocols from multi-linear forms. *Applied Mathematics and Computation*, 159(2):317–331, December 2004.

[118] Y. Li, W. Yang, and C. Huang. On preventing type flaw attacks on security protocols with a simplified tagging scheme. *Journal of Information Science and Engineering*, 21(1):59–84, 2005.

[119] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[120] G. Lowe. Some new attacks upon security protocols. In *Proc. 9th IEEE Computer Security Foundations Workshop (CSFW)*, pages 162–169. IEEE Computer Society, 1996.

[121] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 18–30. IEEE Computer Society, 1997.

[122] G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 31–44. IEEE Computer Society, 1997.

[123] G. Lowe. Towards a completeness result for model checking of security protocols. In *Proc. 11th IEEE Computer Security Foundations Workshop (CSFW)*. IEEE Computer Society, 1998.

[124] Formal Systems (Europe) Ltd. *Failure-Divergence Refinement — FDR2 Users Manual*, 2000.

[125] M. Maffei. Tags for multi-protocol authentication. In *Proc. of the 2nd International Workshop on Security Issues in Coordination Models, Languages, and Systems*, volume 128(5) of *Electronic Notes in Theoretical Computer Science*, pages 55–63. Elsevier ScienceDirect, August 2005.

[126] P. Maggi and R. Sisto. Using SPIN to verify security properties of cryptographic protocols. In *Proc. of the 9th International SPIN Workshop on Model Checking of Software*, volume 2318 of *Lecture Notes in Computer Science*, pages 187–204. Springer, 2002.

[127] S. Malladi, J. Alves-Foss, and R. Heckendorn. On preventing replay attacks on security protocols. In *Proc. International Conference on Security and Management*, pages 77–83. CSREA Press, 2002.

[128] S. Malladi, J. Alves-Foss, and S. Malladi. What are multi-protocol guessing attacks and how to prevent them. In *Proc. 7th International Workshop on Enterprise Security*, pages 77–82. IEEE Computer Society, June 2002.

[129] F. Martinelli. Analysis of security protocols as open systems. *Theoretical Computer Science*, 290(1):1057–1106, 2003.

[130] S. Mauw and V. Bos. Drawing Message Sequence Charts with LaTeX. *TUGBoat*, 22(1-2):87–92, March 2001.

[131] S. Mauw, W.T. Wiersma, and T.A.C. Willemse. Language-driven system design. *International Journal of Software Engineering and Knowledge Engineering*, 14(6):625–663, 2004.

[132] C. Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Proc. 4th European Symposium on Research in Computer Security (ESORICS)*, volume 1146 of *Lecture Notes in Computer Science*, pages 351–364. Springer, 1996.

[133] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[134] C. Meadows. Analysis of the Internet Key Exchange Protocol using the NRL protocol analyzer. In *Proc. 20th IEEE Symposium on Security & Privacy*, pages 216–231. IEEE Computer Society, 1999.

[135] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. *Lecture Notes in Computer Science*, 2052:21, 2001.

[136] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Inc., 5th edition, 2001.

[137] D. Micciancio and S. Panjwani. Optimal communication complexity of generic multicast key distribution. In Jan Camenisch and Christian Cachin, editors, *Advances in cryptology - EUROCRYPT 2004, Proc. of the internarional conference on the theory and application of cryptographic techniques*, volume 3027 of *Lecture Notes in Computer Science*, pages 153–170. Springer, May 2004.

[138] J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*. IEEE Computer Society, 2003.

[139] J.K. Millen. A necessarily parallel attack. In N. Heintze and E. Clarke, editors, *Workshop on Formal Methods and Security Protocols*, Trento, Italy, 1999.

[140] J.K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 166–175. ACM Press, 2001.

[141] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part 1. *Information and Computation*, 100(1):1–40, 1992.

[142] C.J. Mitchell and L. Chen. Comments on the s/key user authentication scheme. *Operating Systems Review*, 30(4):12–16, 1996.

[143] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murϕ. In *1997 IEEE Symposium on Security and Privacy*, pages 141–151. IEEE Computer Society, 1997.

[144] D. Nalla and K. Reddy. Id-based tripartite authenticated key agreement protocols from pairings. Cryptology ePrint Archive, Report 2003, 2003. `eprint.iacr.org/`.

[145] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[146] P.C. van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *Proc. 1st ACM Conference on Computer and Communications Security*, pages 232–243. ACM Press, 1993.

[147] S. Owre, J.M. Rushby, and N. Shankar. PVS: A prototype verification system. In *Proc. 11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer, 1992.

[148] N. Palm. Bewijzen van security protocollen in een trace model. Master's thesis, Technische Universiteit Eindhoven, Department of Mathematics and Computer Science, 2004.

[149] L.C. Paulson. *Isabelle: a generic theorem prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[150] L.C. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 84–95. IEEE Computer Society, 1997.

[151] L.C. Paulson. Proving properties of security protocols by induction. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 70–83, Rockport, Massachusetts, 1997. IEEE Computer Society.

[152] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[153] L.C. Paulson. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, August 1999.

[154] L.C. Paulson. Relations between secrets: Two formal analyses of the Yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.

[155] L.C. Paulson. SET cardholder registration: the secrecy proofs. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning — First International Joint Conference, IJCAR 2001*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 5–12. Springer, 2001.

[156] D. Peled. Ten years of partial order reduction. In *Proc. Computer Aided Verification'98 (CAV)*, pages 17–28. Springer, 1998.

[157] A. Perrig and D. Song. Looking for diamonds in the desert – extending automatic protocol generation to three-party authentication and key agreement protocols. In *Proc. 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 64–76. IEEE Computer Society, 2000.

[158] G.D. Plotkin. A structural approach to operational semantics. Technical Report DIAMI FN-19, Computer Science Department, Aarhus University, 1981.

[159] G.J. Popek and C.S. Kline. Encryption and secure computer networks. *ACM Computing Surveys*, 11(4):331–356, 1979.

[160] A.W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *Proc. 8th IEEE Computer Security Foundations Workshop (CSFW)*, pages 98–107, Kenmare, 1995.

[161] A.W. Roscoe. Intensional Specifications of Security Protocols. In *Proc. 9th IEEE Computer Security Foundations Workshop (CSFW)*, pages 28–38. IEEE Computer Society, 1996.

[162] M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *Theoretical Computer Science*, 299(1-3):451–475, 2003.

[163] P. Ryan and S. Schneider. An attack on a recursive authentication protocol. A cautionary tale. *Information Processing Letters*, 65(1):7–10, 1998.

[164] P. Ryan and S. Schneider. *Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001. With M.H. Goldsmith, G. Lowe and A.W. Roscoe.

[165] S. Schneider. Security properties and CSP. In *1996 IEEE Symposium on Security and Privacy*, pages 174–187. IEEE Computer Society, 1996.

[166] S. Schneider. Formal analysis of a non-repudiation protocol. In *Proc. 11th IEEE Computer Security Foundations Workshop (CSFW)*, pages 54–65. IEEE Computer Society, 1998.

[167] S. Schneider. Verification of authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, 1998.

[168] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C (2nd ed.)*. Wiley, 1995.

[169] I. Schnitzler. Model checking secrecy in security protocols. Master's thesis, Universiteit van Amsterdam, 2003.

[170] V. Shmatikov and J.C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283, 2002.

[171] D. Song. Athena: a new efficient automatic checker for security protocol analysis. In *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW)*, pages 192–202. IEEE Computer Society, 1999.

[172] D. Song. *An Automatic Approach for Building Secure Systems*. PhD thesis, UC Berkeley, December 2003.

[173] D. Song, S. Berezin, and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.

[174] Security protocols open repository (SPORE). `http://www.lsv.ens-cachan.fr/spore`.

[175] G. Steel. Coral project: Group protocol corpus, 2004. `http://homepages.inf.ed.ac.uk/gsteel/group-protocol-corpus`.

[176] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *Proc. 3rd ACM Conference on Computer and Communications Security*, pages 31–37. ACM Press, 1996.

[177] S.G. Stubblebine and R.N. Wright. An authentication logic with formal semantics supporting synchronization, revocation, and recency. *IEEE Transactions on Software Engineering*, 28:256–285, 2002.

[178] P.F. Syverson and P.C. van Oorschot. A unified cryptographic protocol logic. CHACS Report 5540-227, NRL, 1996.

[179] G. Tel. *Cryptografie: Beveiliging van de digitale maatschappij*. Pearson Education, 2002. In Dutch.

[180] F.J. Thayer, J.C. Herzog, and J.D. Guttman. Honest ideals on strand spaces. In *Proc. 11th IEEE Computer Security Foundations Workshop (CSFW)*, pages 66–77. IEEE Computer Society, 1998.

[181] F.J. Thayer, J.C. Herzog, and J.D. Guttman. Mixed strand spaces. In *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW)*, pages 72–82. IEEE Computer Society, 1999.

[182] F.J. Thayer, J.C. Herzog, and J.D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.

[183] F.L. Tiplea, C. Enea, and C.V. Birjoveneanu. Decidability and complexity results for security protocols. In E.M. Clarke, M. Minea, and F.L. Tiplea, editors, *VISSAS*, volume 1 of *NATO Security through Science Series D: Information and Communication Security*, pages 185–211. IOS Press, 2005.

[184] W.G. Tzeng and C.M. Hu. Inter-protocol interleaving attacks on some authentication and key distribution protocols. *Information Processing Letters*, 69(6):297–302, 1999.

[185] G. Wedel and V. Kessler. Formal semantics for authentication logics. In E. Bertino, H. Kurth, and G. Martella, editors, *Proc. 4th European Symposium on Research in Computer Security (ESORICS)*, volume 1146 of *Lecture Notes in Computer Science*, pages 219–241. Springer, 1996.

[186] T. Woo and S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, 28(3):24–37, 1994.

[187] H. Zhou and S. Foley. Fast automatic synthesis of security protocols using backward search. In M. Backes and D. Basin, editors, *Proc. 1st ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 1–10. ACM Press, 2003.

Index of subjects

$\sqsubseteq$ (subterm), 15

active run identifiers, 26
actor, 18
agent
    (un)trusted agents, 29
agreement
    injective agreement, 46
    non-injective agreement, 46
alive, 36
ambiguous authentication, 116
athena, 151
attack
    multi-protocol, 109

basic term, 13
bounded characterization algorithm, 92

Casper/FDR, 150
cast function, 38
characterization, 80
characterization algorithm, 88
claim
    alive, 36
    i-agree, 46
    i-synch, 44
    ni-agree, 46
    ni-synch, 42
    secret, 35
claim set, 14
ClaimRunEv, 25
communication
    label, 14
    order, 17
    relation, 17
complete characterization, 80
$\cdot$, 16, 56
concatenate
    role event sequences, 16
    role events, 16
    run event sequences, 56
connected roles, 109
Const, 13
constants, 13
constraint solver, 151
cont, 26
content of event, 26
contributions of this thesis, 156

decryption chain, 86
decryption unifier, 87
Dolev-Yao intruder model, 29

E, 29
event
    content of, 26
explicit trace pattern, 78

Func, 13
function names, 13
future work, 158

i-agree, 46
inferable
    from role knowledge set, 18
initial intruder knowledge, 30
initial network state, 27
injective agreement, 46
injective synchronisation, 44
instantiation
    instantiation of role term, 22
    symbolic instantiation, 70
    well-typedness of, 24
instantiation function, 21
intruder
    Dolev-Yao, 29

no intruder, 29
passive, 29
wireless communications, 29
intruder knowledge
initial intruder knowledge, 30
intruder rules, 28
Isabelle, 150
i-synch, 44

key, 2
knowledge
inference operator, 18

label
communication, 14
local constants, 13
LOOP property, 59
LySatool, 151

match, 24
MGDU, 87
MGU, 87
minimal number of messages, 135
most general decryption unifier, 87
most general unifier, 87
multi-protocol attack, 109

network state, 23
initial network state, 27
ni-agree, 46
ni-synch, 42
non-injective agreement, 46
non-injective synchronisation, 42
NRL, 151

OFMC, 151
operational semantics, 27

pattern
realizable trace pattern, 75
trace pattern, 71
possible runs, 26
protocol
execution, 21
multiple, 109
single, 109
specification, 16
protocol specification, 12
protocol update attack, 114

read enabled predicate, 75
readable role term, 19
ReadRunEv, 25
realizable trace pattern, 75
refinement
of trace pattern, 73
role
knowledge, 16
specification, 16
well-formed, 18
Role, 13
role consistent predicate, 74
role event, 14
instantiation of role event, 71
order, 17
role names, 13
role term, 13
instantiation of role term, 22
readable, 19
symbolic instantiation, 70
roles
connected, 109
run, 21, 22
active run identifiers, 26
possible runs, 26
run term, 21
RunEvent, 25

scytale, 1
secret, 35
security claim, 14
semantics, 27
SendRunEv, 25
static requirements, 18
substitution, 26
subterm, 15
swap property, 56
symbolic term instantiation, 70
synchronisation
injective synchronisation, 44
non-injective synchronisation, 42

term
basic term, 13

role term, 13
run term, 21
threat model, 27
agent threat model, 29
network threat model, 28
trace
realizable trace pattern, 75
trace pattern, 71
events, 77
explicit trace pattern, 78
protocol traces, 72
read enabled, 75
refinement, 73
role consistent, 74
traces defined by a trace pattern, 72
unique runs, 74
transition label, 25
trusted agent, 29
type, 23
type matching
simple type matching, 23
strict type matching, 23
typeless matching, 24

unifier, 87
unique runs predicate, 74
untrusted agent, 29
update attack, 114

Var, 13

well-formed protocol, 20
Welltyped, 24

Samenvatting

Recente technologieën hebben de weg vrijgemaakt voor de grootschalige toepassing van elektronische communicatie. Het open en gedistribueerde karakter van zulke communicatie heeft tot gevolg dat het communicatiemedium niet meer geheel, of zelfs helemaal niet, onder controle is van de communicerende partners. Deze ontwikkelingen hebben geleid tot een toenemende vraag naar onderzoek op het gebied van veilige communicatie over potentieel onveilige netwerken. Dit is te realiseren door toepassing van beveiligingsprotocollen.

Dit proefschrift ontwikkelt een formeel model voor de beschrijving en analyse van beveiligingsprotocollen op procesniveau. De analyse gebeurt onder de aanname dat cryptografische methodes perfect zijn en heeft tot doel om fouten en mogelijke risicopunten voor externe aanvallen in het protocol te vinden.

In het proefschrift wordt vanuit grondbeginselen een operationele semantiek ontwikkeld om beveiligingsprotocollen en hun gedrag te beschrijven. Het resulterende model heeft verschillende parameters, waaronder een aanvullend model dat de potentiële aanvallen op de protocollen beschrijft. Diverse voorbeelden van instanties van deze parameter worden gegeven, en omvatten onder andere passieve aanvallen, en het model dat in de literatuur als het sterkste aanvalsmodel bekend staat. Binnen het protocolmodel worden diverse beveiligingseigenschappen geformuleerd, in het bijzonder voor het bereiken van geheimhouding en voor nieuwe vormen van authenticatie. Diverse nieuwe resultaten over deze eigenschappen worden bewezen.

Op basis van het model wordt een automatische verificatiemethode ontwikkeld, die een significante verbetering vormt op in de literatuur bestaande ideeën. De verificatiemethode is geïmplementeerd in een prototype, dat betere resultaten levert dan de bestaande methodes.

Zowel de theorie als de verificatiemethode worden toegepast in twee nieuwe analyses. Met behulp van het verificatieprototype worden nieuwe resultaten bereikt op het gebied van protocolcompositie. De analyses leiden tot de ontdekking van een klasse van aanvallen die nog niet eerder waren beschreven. Daarnaast wordt het formele model gebruikt voor de ontwikkeling en analyse van een nieuw geparameteriseerd protocol. Het protocol wordt correct bewezen binnen het model.

Summary

Recent technologies have cleared the way for large scale application of electronic communication. The open and distributed nature of these communications implies that the communication medium is no longer completely controlled by the communicating parties. As a result, there has been an increasing demand for research in establishing secure communications over insecure networks, by means of security protocols.

In this thesis, a formal model for the description and analysis of security protocols at the process level is developed. At this level, under the assumption of perfect cryptography, the analysis focusses on detecting flaws and vulnerabilities of the security protocol.

Starting from first principles, operational semantics are developed to describe security protocols and their behaviour. The resulting model is parameterized, and can e.g. capture various intruder models, ranging from a secure network with no intruder, to the strongest intruder model known in literature. Within the security protocol model various security properties are defined, such as secrecy and various forms of authentication. A number of new results about these properties are formulated and proven correct.

Based on the model, an automated verification procedure is developed, which significantly improves over existing methods. The procedure is implemented in a prototype, which outperforms other tools.

Both the theory and tool are applied in two novel case studies. Using the tool prototype, new results are established in the area of protocol composition, leading to the discovery of a class of previously undetected attacks. Furthermore, a new protocol in the area of multiparty authentication is developed. The resulting protocol is proven correct within the framework.

Curriculum Vitae

	Born on 16 June 1974 in Geleen, the Netherlands.
1986-1993	Secondary School: Atheneum-B, Scholengemeenschap Sint Michiel, Geleen, the Netherlands.
1993-2002	M.Sc. degree in Computer Science, Technische Universiteit Eindhoven, The Netherlands. Title of Master's Thesis: "How best to beat high scores in Yahtzee: A caching structure for evaluating large recurrent functions", supervisor dr.ir. T. Verhoeff.
2002-2006	Ph.D. student in the Formal Methods group, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven the Netherlands.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-01

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-02

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-03

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-04

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-05

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-06

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-07

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-08

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-09

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Scheduler Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation*

and communication. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*.

Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models*. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications*. Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20