DISS. ETH NO. 17013

MODELS AND METHODS FOR THE AUTOMATED ANALYSIS OF SECURITY PROTOCOLS

A dissertation submitted to
ETH ZURICH
for the degree of
Doctor of Sciences

presented by
SEBASTIAN ALEXANDER MÖDERSHEIM
Dipl.-Inf., Universität Freiburg, Germany
born December, 28th 1974
citizen of Germany

accepted on the recommendation of
Prof. David Basin, examiner
Prof. Alessandro Armando, co-examiner
Prof. Luca Viganò, co-examiner

2007

To all honest intruders.

Acknowledgments

I want to thank my supervisor David Basin for guiding and supporting me and at the same time giving me the freedom to pursue my ideas. He has shaped my scientific work and style by forcing me to abstract and to "crispify" my ideas.

I want to thank Luca Viganò, who has also supported me throughout my doctoral research. Writing papers and discussing ideas with him was not only highly productive, it was also a pleasure.

I want to thank David Basin, Achim Brucker, Paul Hankes Drielsma, Felix Klaedtke, Pascal Lafourcade, Patrik Schaller, and Luca Viganò for helpful comments on the first draft of my thesis. I also want to thank all past and present members of the AVISPA team for many inspiring discussions and the productive teamwork.

Most importantly, I want to thank my parents for their love and support.

Thank you all very much!

Zurich, Switzerland
November, 5th 2006

Sebastian Alexander Mödersheim

Zusammenfassung

Die automatische Analyse von Sicherheitsprotokollen erfordert effiziente Methoden, um die spezifischen Probleme anzugehen, die sich in der Verifikation von Sicherheitsprotokollen ergeben, wie beispielsweise die Deduktionen eines Angreifers. In dieser Arbeit entwickeln wir neue Methoden und Verbesserungen existierender Methoden in diesem Bereich. Diese Methoden basieren auf symbolischem Model-Checking mit Constraints, um den Angreifer effizient zu repräsentieren. Wir integrieren Ideen aus dem Bereich der Partial-Order-Reduction in diesen constraint-basierten Ansatz, um das Problem des Parallelismus anzugehen, was zu einer neuen Technik namens Constraint-Differenzierung führt. Ausserdem benutzen wir Techniken des modularen Rewritings, um auch algebraische Eigenschaften der kryptographischen Operatoren betrachten zu können.

Wir haben diese Techniken in dem Werkzeug OFMC (On-The-Fly Model-Checker for security protocols) implementiert. Wie wir in einer Reihe von Fallstudien und Experimenten an komplexen Sicherheitsprotokollen zeigen, hat OFMC den Stand der Technik verbessert, sowohl im Hinblick auf Geschwindigkeit als auch im Hinblick auf die Klasse der Sicherheitsprotokolle, die betrachtet werden können. Als Teil des AVISPA-Tools wird OFMC von Dutzenden Forschern und Protokolldesignern weltweit benutzt, um Sicherheitsprotokolle zu validieren.

Neben der Entwicklung von Methoden und Werkzeugen geht es in dieser Arbeit auch um subtile Fragen der Modellierung von Sicherheitsprotokollen. Wir definieren einen Formalismus um Sicherheitsprotkolle, ihre Ziele und das Angreifermodell eindeutig zu spezifizieren. Darüber hinaus führen wir einen formalen Vergleich durch zwischen unserem Modell und anderen Modellen, welche auf Überapproximation basieren, und beschreiben ihre Beziehung formal. Neben einem besseren Verständnis dieser Modelle erlaubt uns dies zu beweisen, dass aus der Sicherheit eines Protokolls in der Überapproximation die Sicherheit im Standardmodell folgt.

Abstract

The automated analysis of security protocols requires efficient methods to address the specific problems that arise in the verification of security protocols, such as the deductions of an intruder. In this thesis, we develop new such methods and improve existing ones. Our proposed methods are built upon symbolic model-checking with constraints to efficiently represent the intruder. We integrate into the constraint-based approach ideas from partial-order reduction to deal with the problem of parallelism, leading to a novel technique called constraint differentiation. Moreover, we use modular rewriting techniques to allow for the consideration of algebraic properties of cryptographic operations.

We have implemented these techniques into the tool OFMC (On-The-Fly Model-Checker for security protocols). As we demonstrate by a number of case studies and experiments on complex security protocols, OFMC has improved the state of the art both in terms of performance and in terms of the class of security protocols that can be considered. As part of the AVISPA tool, OFMC is used by dozens of researchers and protocol designers world wide to validate the design of security protocols.

Besides the development of methods and tools, this thesis is also concerned with the subtle issues of modeling security protocols. We define a formalism for unambiguously specifying security protocols, their goals, and the intruder model. Moreover, we formally compare our model with other models based on over-approximation and formally describe their relationship. Besides a better understanding of these models, this allows us to prove that safety of a protocol in the over-approximation implies its safety in the standard model.

Contents

1 Introduction **1**

I Modeling **5**

2 Alice and Bob **9**
2.1 Message Exchange of Yahalom . . . 9
2.2 Between the Lines . . . 10
2.3 An Attack to Yahalom . . . 11
2.4 Formal Semantics . . . 12

3 Messages as Terms **15**
3.1 Free Algebra . . . 16
3.2 Substitutions, Matching, Unification, Rewriting . . . 17
3.3 Rewriting . . . 18
3.4 Dolev-Yao Intruder Deduction . . . 19
3.4.1 Functions . . . 21
3.4.2 Discussion . . . 21
3.5 Algebraic Properties . . . 22
3.5.1 Equational Theories . . . 22

4 Protocols as State Transition Systems **25**
4.1 Set Rewriting . . . 26
4.1.1 Extensions . . . 26
4.2 The Intermediate Format . . . 27
4.2.1 Syntax . . . 27
4.2.2 Semantics . . . 29
4.3 Agents Model . . . 30
4.3.1 Relationship Between Sessions . . . 32
4.3.2 An Unbounded Number of Sessions and Agents . . . 33
4.4 Communication Model . . . 34
4.4.1 Agent Names . . . 35
4.4.2 The Intruder as a Dishonest Agent . . . 36
4.4.3 Step Compression . . . 37
4.4.4 Typed Protocol Models . . . 39
4.5 Protocol Goals . . . 40
4.5.1 Secrecy . . . 40
4.5.2 Authentication . . . 41

II The Lazy Intruder Framework **45**

5 Exploring the Transition System **47**
5.1 A Semi-decision Algorithm . . . 48
5.2 Using Lazy Evaluation . . . 49
5.3 Feasibility . . . 49

6 The Lazy Intruder **51**
6.1 Constraints . . . 51
6.2 Constraint Reduction . . . 52
6.2.1 Reduction without Analysis . . . 54
6.2.2 Analyzing the Intruder Knowledge . . . 60
6.2.3 Integration of Generation and Analysis . . . 62
6.2.4 Improvements to the Analysis . . . 67
6.3 Inequality Constraints . . . 68

7 An ADT for the Lazy Intruder **71**
7.1 Implementation of the Constraint Store . . . 72

8 The Symbolic Transition System **75**
8.1 The Symbolic Successor Relation . . . 76
8.2 Symbolic Attack States . . . 79
8.3 Organizing Symbolic State Exploration . . . 79
8.4 Symbolic Sessions . . . 80

9 Constraint Differentiation **85**
9.1 A Common POR Situation . . . 86
9.2 Constraint Reduction with Constraint Differentiation . . . 88
9.3 Integrating Constraint Differentiation . . . 94
9.4 Experimental Results . . . 95

III Algebraic Properties **97**

10 Finite Theories **101**
10.1 Two Classes of Theories . . . 101
10.2 Restriction to a Bounded Variable Depth Model . . . 102
10.3 Unification modulo Finite Theories . . . 104
10.4 Intruder Deduction Modulo Finite Theories . . . 108

11 Cancelation equations **111**
11.1 Cancelation as Analysis . . . 112
11.2 Undecidability of Analysis . . . 113
11.3 Integration into IF . . . 114
11.4 The Lazy Algebraic Intruder . . . 116
11.4.1 Using Finite Theory Decomposition . . . 116
11.4.2 Cancelation Theories . . . 117

12 Application to Modeling Dictionary Attacks **119**
12.1 Intuition . . . 120
12.1.1 Decryption with a Wrong Key . . . 120
12.2 A Formal Model for Off-Line Guessing . . . 122
12.3 A Concrete Example: The MS-CHAPv2 Protocol . . . 125

IV Abstraction **127**

13 Models Based on Set Rewriting and Message Traces **131**
13.1 Message Patterns . . . 131
13.2 The Set Rewriting Model . . . 133
13.3 The Message Trace Model . . . 134

14 The Over-Approximation and Persistent Set Rewriting **135**
14.1 Persistent Set Rewriting . . . 136
14.2 Relation to Set Rewriting . . . 137
14.3 Inclusion in the Inductive Message Trace Model . . . 137

15 Reachable Facts and Events **141**

16 Goals **147**
16.1 Secrecy . . . 147
16.2 Authentication . . . 149
16.3 A Subtle Problem About Authentication . . . 151
16.4 Alternative Formulation of Authentication . . . 152

V Experimental Results, Related Work, and Conclusions **157**

17 Case Studies **159**
17.1 H.530 . . . 159
17.2 Asokan-Shoup-Waidner . . . 161
17.3 Secure Remote Passwords . . . 166

18 Protocol Libraries **169**
18.1 The Clark/Jacob Library . . . 169
18.2 AVISPA-Library . . . 171
18.2.1 The Methods of the Back-Ends . . . 173
18.2.2 The Flaws . . . 173

19 Conclusions and Related Work **175**
19.1 Modeling . . . 175
19.1.1 Contributions . . . 176
19.2 Automated Methods . . . 177
19.3 Algebraic Properties . . . 178
19.4 Abstraction-Based Analysis . . . 180
19.5 Experiments and Case Studies . . . 181
19.6 Future Work . . . 182

Chapter 1

Introduction

Security protocols are omnipresent in electronic communication, since communication media such as the Internet or wireless networks cannot be assumed to be secure. For instance, on-line banking cannot be performed without counter-measures against fraud. Firstly, we need to ensure *authentication* and *integrity* of the transmitted data: the bank needs to be sure that the customer who seemingly wants to perform a certain transaction indeed intends so. Secondly, we need *secrecy*: the details of transactions and the customer's bank-account must not be visible to a (possibly malicious) intermediate node of the Internet between the customer and bank. In particular, when using a password for the authentication of the customer, this password must be protected. There are further goals we want to achieve, such as *non-repudiation* (the customer cannot later deny that he performed some transaction). Failure to meet these goals can obviously lead to a financial disaster.

To achieve these goals, we use security protocols, based on cryptographic operations. The cryptographic operations are the basic building blocks which guarantee that certain messages cannot be created, read, or altered without knowing a certain secret. Security protocols conduct secure communication on this basis by prescribing how such cryptographic messages are exchanged between participants. For instance, in the above banking example we can imagine a protocol that is based on a shared secret, or password, between the customer and the bank. To perform a transaction, it is not sufficient that a customer simply sends an electronic form encrypted with the password: an intruder may observe this message and replay it several times, so that the bank would execute the transfer of money accordingly often. Therefore, we need a more sophisticated protocol to ensure that messages cannot be replayed, e.g. using a sequence number or a time-stamp.

We have here assumed (and we will assume throughout this thesis, unless noted otherwise) that the cryptography is perfect, i.e. an intruder cannot break the cryptography and thus create, read, or alter messages for which he does not know an appropriate secret. Also, we have assumed that customer and bank have such a secret that nobody else knows. However, the example of a simple on-line banking protocol demonstrates that, even under such assumptions, one can easily make subtle mistakes in the design of security protocols that lead to serious vulnerabilities. The most prominent example is probably the Needham-Schroeder Public-Key protocol (NSPK) that we will consider several times during the thesis. The core of this protocol consists of only 3 simple messages, and it was considered to be safe for about 17 years, until an attack was discovered in a formal analysis.

There are several reasons why the security of protocols can be so subtle. One reason can be that the circumstances and silent assumptions are unclear, e.g. that certain participants are trusted. Another is that protocols are distributed systems where the events of several runs of a protocol can be *interleaved*. In particular, in the presence of an intruder who can hold back, replay, or redirect messages, some interleavings of events may evade the attention of the human mind when reasoning about such a protocol. It is therefore necessary to have methods to analyze security protocols in a rigorous, formal way.

The main goal of this thesis is therefore to improve existing methods, and develop new ones, for

the automated analysis of security protocols. The aim is to provide designers of security protocols with an efficient, flexible, and widely applicable tool to validate, or find flaws in, their protocols before deployment.

To achieve this goal, we need efficient methods to address the core problems that arise in the analysis of security protocols, such as the intruder deduction problem: an intruder can construct an infinite number of different messages from his knowledge, i.e. from the set of messages he has observed so far on the communication medium. Even under restrictions, methods based on a naïve enumeration do not scale well to protocols with complex messages. In this thesis, we will describe the methods we have developed to address these problems and that we have implemented in a tool called the *On-The-Fly Model-Checker for security protocol analysis* (OFMC), which is among the tools with the best performance and coverage that are currently available. Besides the development of methods and tools, this thesis is also concerned with the subtle issues of modeling security protocols, such as the model of an intruder, as well as formal comparisons between different ways of modeling. The models and techniques behind OFMC and its deployment in protocol analysis are described in eleven papers at international conferences and workshops [7, 8, 20, 24, 25, 26, 27, 48, 83, 84, 85], as well as a journal paper [28]. The thesis is structured into five parts, and we give now a more detailed overview of each.

Part I: Modeling

The first part of the thesis is concerned with the modeling of security protocols. Besides reviewing the mathematical preliminaries, this describes in detail the many subtle aspects of modeling security protocols and their goals, along with the design choices and assumptions we made in this work. This part also serves for describing the problems we consider in a precise, unambiguous way.

The contributions of this part are as follows:

- We define the syntax and semantics of the *Intermediate Format* (IF) [14], a formalism for describing security protocols and their goals. IF is the input language of OFMC and a number of other protocol analysis tools. This formalism was designed as part of the EU-projects AVISS [8] and AVISPA [7].

- Using IF, we describe in detail how we model security protocols, the intruder, and goals. Note that modeling itself is not a formal process—only the result is. We give examples and justify our design decisions and discuss other models.

- We consider *step compression*, a common optimization of protocol models that can be described as "the intruder *is* the network". We show formally that step compression is correct in the sense that the step-compressed model is safe iff the original model is.

Part II: The Lazy Intruder Framework

The second part is dedicated to the core techniques used in OFMC. We begin with a general method of exploring *on-the-fly* an infinitely large tree, as which we can represent the protocol insecurity problem. We then consider a method to efficiently represent the intruder. We avoid the enumeration of the (in general infinite) set of messages that the intruder can construct from his knowledge by using a symbolic representation with constraints. We call this representation the *lazy intruder* since constraints are evaluated in a demand-driven, lazy way.

This symbolic technique was first proposed by [91] and has proven successful in the analysis of security protocols, e.g. [6, 37, 38, 52, 60, 77, 111]. Our contributions to this symbolic technique are as follows.

- We give a precise algorithmical account of the lazy intruder technique that is close to the real implementation and show its correctness, i.e. that the results of the lazy intruder algorithm are equivalent to the standard intruder model.

- We extend previous approaches to a larger class of protocols and goals by including inequalities, which allows us to handle protocol descriptions with negative conditions in transitions.
- We formalize the integration of the lazy intruder technique with the analysis of security protocols. We perform this in an abstract way using an abstract data-type for constraint stores. This allows us to separate the search space of the lazy intruder from higher levels of the search, hiding technicalities like the condition of well-formed constraints. Moreover, the definition of this abstract data-type underlines the general applicability of the lazy intruder technique. We also describe how to efficiently implement the lazy intruder without sacrificing declarativity, using lazy evaluation in functional programming languages like Haskell, the implementation language of OFMC.
- We consider the problem of exploring different protocol scenarios, i.e. sets of *sessions* where different agents assume different roles in the interleaved protocol executions. We develop the *symbolic sessions* technique that exploits the symbolic representation of the lazy intruder and thereby avoids enumerating all possible session instances associated with a bounded number of sessions.
- We formalize a novel technique called *constraint differentiation*, which integrates the symbolic approach with ideas from partial-order reduction. This addresses the explosion of the search space that arises from the large number of interleavings when considering a larger number of parallel sessions. Partial-order reduction is a standard technique in model checking to exploit redundancies in the interleavings of parallel processes. This technique is not directly applicable to the symbolic state space that results from the lazy intruder; constraint differentiation integrates ways of exploiting such redundancies into the constraint reduction of the lazy intruder. We demonstrate with experiments that this substantially improves the performance and scope of OFMC for large numbers of parallel sessions.

Part III: Algebraic Properties

In the third part, we consider the integration of algebraic properties into the reasoning. Throughout the second part we interpret terms representing the cryptographic messages in the free term algebra, i.e. syntactically different terms are interpreted as being different. This interpretation, however, makes it impossible to handle a large variety of modern protocols that exploit algebraic properties of the cryptographic operations, such as associativity and commutativity. Integrating algebraic properties into the reasoning is non-trivial, since many basic problems like the equality of terms or the intruder deduction problem are in general undecidable under algebraic properties.

While several problems related to algebraic properties have been intensively studied, the problem of intruder deduction modulo such properties is relatively new. Many approaches are specialized to particular properties of an operator, e.g. [49, 50, 112].

Our contribution is to provide a framework for intruder deductions modulo algebraic theories with the following properties:

- Our framework is not specialized to a particular set of algebraic equations (such as associativity and commutativity), but can uniformly handle a large class of algebraic properties. A large number of the algebraic properties that are required in protocol analysis fall into this class. This allows the users of OFMC to specify custom algebraic theories under which the protocol is checked.
- The framework also allows for the integration of existing efficient algorithms specialized to particular algebraic theories, so that the specialized algorithm can be used whenever available and the general algorithm otherwise.
- Due to the undecidability of the general problem, we have to make several restrictions, e.g. bounding intruder deductions. For each restriction that we consider, we show that at least one basic problem of protocol analysis is undecidable without the restriction.

- We formalize the integration of the framework into the protocol analysis and give a summary of the ideas behind our integration with the lazy intruder technique which is implemented in OFMC.
- Finally, we show a major application for algebraic properties in the formalization of off-line guessing attacks. When weak passwords (that are likely to be guessed) are used in security protocols, we may obtain a vulnerability that the intruder can find out such a password if there is anything in the messages that tells him which of many guesses was the correct one. Several protocols have been designed to prevent guessing attacks, and verifying that they achieve this goal requires a formalization of guessing.

Part IV: Abstraction

A common problem for the automated analysis of security protocols arises when considering an unbounded number of executions (sessions) of the protocol in parallel, since this gives rise to an infinite state-space. For this reason, several approaches have considered over-approximations of protocols, i.e. defining models that admit at least as many states or traces as the real protocol. Such an over-approximation is often easier to prove correct than the original model, and proving the over-approximation is safe implies that the original model is safe—provided that the models are indeed in an over-approximation relation.

Our contributions in this part are the following:

- We show that several models are indeed over-approximations of our IF protocol model in a sense that we define precisely.
- We thereby clarify in a formal way the subtle relationship between several models with different degrees of over-approximation.
- From these results, it is immediate that all secrecy goals that hold in the over-approximation of a protocol also hold in the original model. However, for authentication goals, the same reduction is not possible as we show with a counter-example. We finally show how to correct this problem, so that over-approximations can also be used for authentication.

Part V: Experimental Results

While in the main exposition of techniques and models we restrict ourselves to small academic examples of protocols for simplicity, we consider in the final part some "real world" protocols and describe in detail several case studies we have performed with OFMC. We see these case studies as a contribution in itself, as we gain insights on the protocols themselves and on many issues of modeling. In particular, we describe an attack that we have found with OFMC on the H.530 protocol [92]. Our detection of an attack has led to a change of this protocol [93].

We also describe our analysis of two major libraries of protocols, the Clark/Jacob library [54] and the AVISPA library [13]. We have used both libraries during the development of OFMC as test-suites to asses the effects of changes and improvements, in particular, to prevent bugs in the development. We use the AVISPA library in this thesis to compare OFMC with the other tools developed during the AVISPA project, CL-AtSe, SATMC, and TA4SP.

We conclude the fifth part with a summary of the entire thesis, a comparison with related work, and an outlook on future work.

Part I

Modeling

Designing a formal model of "reality" (or aspects thereof) is a challenging task and one that partially lies outside of mathematics. A first fundamental problem—shared with all empirical sciences—is to figure out whether our view of reality (if we accept that there is one) is correct and our perception has not been mislead. This problem can only be addressed by a repeated process of building a formal model or theory, deriving new statements from that theory and checking by experiments whether these statements do (seem to) hold; if so, the theory *may* be fine, but otherwise the theory is clearly wrong and must be changed to reflect all empirical data.

In the realm of formal verification of hardware or software, we cannot be sure that the assumptions of the verification are actually met by the real implementation of the basic building-blocks: in hardware verification, we rely on, e.g. transistors working correctly (which is a matter of physics), in software verification, in turn, we must rely on the hardware to work correctly. This problem carries over to the verification of cryptographic algorithms (which are implemented as pieces of hardware or software) and of security protocols. In addition, the security of cryptography usually relies on assumptions about computational hardness. Such an assumption is not a genuinely empirical problem since, e.g. the question whether P equals NP is mathematically well-defined and might be answered one day.[1] The empirical aspect of this assumption is the argumentation that nobody has found a proof for equality or for inequality yet, even though the problem has been known for a long time and has been studied by many of the most brilliant minds.

A second problem of modeling comes into play whenever, due to the complexity of reality, simplifying assumptions have to be made or only some aspects of reality are modeled. The question whether the model is appropriate and includes all relevant aspects for the questions at hand clearly lies beyond mathematics as well. However, a good way to detect mistakes and shortcomings of the modeling process is to have two (or more) levels of models, i.e. first have a *meta-model* that is close to ones intuition but complex or unsuitable for reasoning, and then mathematically prove the correctness of a simpler, low-level model with respect to the meta-model. Still, this does not ultimately solve the problem.

An inappropriate model of security protocols can lead to two kinds of problems. First, we may miss vulnerabilities of a flawed protocol and verify it as correct; we call this situation *false negative* (with respect to finding attacks). Second, a correct protocol may count as vulnerable in the formal model; we call this situation *false positive*. False positives are especially problematic when working with automated analysis tools for protocol analysis: even if the user is able to decide which of the potential attacks reported by the analysis tool are relevant in reality, it is impractical to find a relevant attack in a "hey-stack" of thousands of nonsensical attacks.

For all of these reasons, we want to be explicit about the motivations, ideas, and design choices behind the model we use in this work; in particular, we discuss differences with respect to other protocol models. Also, we briefly discuss the design of formal protocol specification languages. These languages can at least be helpful to avoid a certain amount of modeling mistakes in the same way as advanced high-level programming languages can help to avoid typical mistakes of low-level programming languages. Another benefit of defining and using a formal specification language is when several analysis tools take the same language (with the same semantics) as input. This allows one to compare analysis tools and their results, and easily benefit from their complementary strengths—without specifying the protocol several times in different formalisms.

We proceed as follows:

- In Chapter 2, we start with a discussion of a semi-formal notation often used in textbooks and articles about security protocols, the so-called *Alice-and-Bob notation*. The questions that arise from this notation are our motivation and starting point for the definition of a formal model.

- In Chapter 3, we start with the building blocks of security protocols, the messages that are exchanged. We represent messages by terms and we discuss the implications of this modeling. We summarize the standard concepts of terms that we need in the thesis. Also

[1] Even mathematics itself is of course not beyond any doubt, as the fundamental axioms or theories like set theory may eventually turn out to be inconsistent (and therefore useless).

we introduce the intruder deduction problem, i.e. what the intruder can deduce from a set of messages that he has seen. Finally, we define what it means to interpret messages within an algebra defined by a set of equations such as commutativity of an operator.

- In Chapter 4, we formally define the syntax and semantics of the *Intermediate Format IF* [14] that we use throughout the thesis to describe protocols. We then describe in detail how honest agents, the intruder, and the goals can be modeled within this formalism. Note that this is only one of many possible ways to model protocols, and we discuss alternatives and justify our choices. The modeling itself is therefore not a definition, but an example for using IF. The following parts of the thesis are independent from the details of modeling and work for every IF protocol description. Also, we describe in Chapter 4 the step compression technique, a common optimization of protocol models. We show for every protocol that it has an attack in the step-compressed model iff it has one in the original model.

Chapter 2

Alice and Bob

In the literature on security protocols, one often finds descriptions in a semi-formal notation, called the *Alice-and-Bob notation*, or A&B for short. The beauty of this notation lies in its simplicity and succinctness. Its popularity stems from the fact that most protocols can be described that way; and even when a protocol has aspects that cannot be described in A&B, such aspects are often just informally noted at the side of an A&B description. Clark and Jacob collected a large library of authentication protocols in A&B [54], including the protocol *Yahalom* (which was first described in [43]) that is displayed in Figure 2.1 and that we will use as an example for our discussion of this notation.

2.1 Message Exchange of Yahalom

This protocol describes the communication between three parties, A, B, and S, where S stands for *server*. Intuitively, the purpose of the protocol is to establish a key between A and B with the help of S; we will later define the goals and assumptions of this protocol more precisely.

In the messages exchanged, the notation $\{|M|\}_K$ represents *symmetric encryption* of a message M with a key K.[1] For now it suffices to say that a symmetric encryption means that an agent is able to decrypt such a message (and obtain the clear-text M) iff he knows the key K. We will shortly come to the assumptions of this protocol about which party should know which keys. Concatenated messages are separated by commata.

One issue that might be confusing for an unexperienced reader is the fact that the notation of a protocol step of the form $n.\, A \to B : M$ seems like an *atomic* action. However, this is not meant by this notation: if there were no danger of an intruder reading, intercepting, or altering the message, then why would one bother to encrypt messages⁉ Actually, each line in A&B represents two different kinds of actions: Firstly, the party A constructs the message M and sends it (via whatever communication medium is used). Secondly, B receives a message of the form of M and performs necessary decryption operations—where we still have to clarify how B determines whether the received message has the appropriate form.

For instance, in Yahalom, A starts the protocol by sending the message 1, waits until she[2] receives the message of form 3 (seemingly coming from S), and then reacts by sending message 4. So in fact, this protocol describes three separate *programs*, called the *roles* of the protocol in the following.

We now describe the protocol message by message, and clarify the assumptions underlying it.

1. In the first message, A creates a random number NA, often called a *nonce* (for "*n*umber used

[1] The notation for the cryptographic primitives may vary from author to author. We prefer for instance to distinguish symmetric and asymmetric encryption in our notation (the latter being expressed as $\{M\}_K$), because it is in some cases unclear how to derive from the context which kind of encryption is meant.

[2] We adopt the following convention: A (Alice) is female, B (Bob) and all dishonest parties are male, and servers and trusted third parties like S are neutral.

1. $\mathsf{A} \rightarrow \mathsf{B}: \mathsf{A}, \mathsf{NA}$
2. $\mathsf{B} \rightarrow \mathsf{S}: \mathsf{B}, \{\!|\mathsf{A}, \mathsf{NA}, \mathsf{NB}|\!\}_{\mathsf{k(B,S)}}$
3. $\mathsf{S} \rightarrow \mathsf{A}: \{\!|\mathsf{B}, \mathsf{KAB}, \mathsf{NA}, \mathsf{NB}|\!\}_{\mathsf{k(A,S)}}, \{\!|\mathsf{A}, \mathsf{KAB}|\!\}_{\mathsf{k(B,S)}}$
4. $\mathsf{A} \rightarrow \mathsf{B}: \{\!|\mathsf{A}, \mathsf{KAB}|\!\}_{\mathsf{k(B,S)}}, \{\!|\mathsf{NB}|\!\}_{\mathsf{KAB}}$

Figure 2.1: The Yahalom protocol in Alice-and-Bob notation.

only *once*"), and sends it together with its name to B. Observe that, in reality, this message is just a bit-string and the protocol description does not tell us how this is decomposed into an agent name and a nonce; in a real implementation, this must be determined by defining the message format in more detail (e.g. fixed lengths).

2. B next also creates a nonce, NB, and encrypts it with the key $\mathsf{k}(\mathsf{B},\mathsf{S})$. This key shall be a secret shared only between B and S. Also, B adds his name in clear-text; this is necessary since otherwise S would have to try to decrypt the message with all keys that he knows. A subtle question is how S can distinguish a valid message from an arbitrary bit-string (of appropriate length): if S tries to decrypt some arbitrary bit-string with the key $\mathsf{k}(\mathsf{B},\mathsf{S})$, the result will be some garbage, but the only thing S can check about the obtained "clear-text" is whether the first part constitutes an agent name A that he knows (otherwise he will not be able to send the next message anyway). It is at least very unlikely that a random bit-string when decrypted with $\mathsf{k}(\mathsf{B},\mathsf{S})$ gives an agent name that S knows. Also, often the concrete implementation may include a special bit-string, called *tag*, into the clear-text, and if on decryption the receiver does not find this tag, it throws the message away. We will come back to this problem to recognize a correct decryption; for instance this is a central issue in guessing attacks in Chapter 12.

3. In step 3, S creates a new key KAB which, as the name suggests, shall be a shared key for A and B. Together with nonces and agent names, S produces two encryptions, one for A using the key $\mathsf{k}(\mathsf{A},\mathsf{S})$ which is a shared secret between A and S, and one for B using again $\mathsf{k}(\mathsf{B},\mathsf{S})$. Receiving this message, A decrypts the first part, checks that it contains the nonce NA she created earlier, and the name B of the agent with whom she wants to communicate. Also, she learns the key KAB from this message. The second part of the message is indecipherable for A as she does not know the key $\mathsf{k}(\mathsf{B},\mathsf{S})$. Therefore, A accepts simply any bit-string (of an appropriate length) for this part of the message.

4. In step 4, A forwards this bit-string from the server to B, along with a proof that she now possesses the new key KAB, namely encrypting B's nonce NB with this key. Finally, B will check that the first part of the message contains the name of agent A that he wants to communicate with, learns the key KAB and checks the second part of the message, namely that it contains his nonce NB.

2.2 Between the Lines

Besides the mentioned subtleties that agents must have some way to determine if decryptions are correct, there are actually several assumptions necessary for the protocol to make sense. The server S must have shared symmetric keys with all other agents. The nonces and the symmetric key KAB are random numbers created by the first person who uses them. Also, the purpose of this protocol is not entirely clear from the A&B description itself; it is that the server S creates and distributes the *session key* KAB for A and B that they can use for future communication. Therefore, the security goals we attach with the protocol are that nobody except A, B, and S should be able to obtain the key KAB and nobody should be able to trick A or B into accepting a different session key than the one created by S. Any violation of these goals we call an *attack*. In Section 2.3, we show that there is indeed an attack against this protocol.

1. $\mathsf{i} \rightarrow \mathsf{b}: \mathsf{i}, \mathsf{na}$
2. $\mathsf{b} \rightarrow \mathsf{s}: \mathsf{b}, \{|\mathsf{i}, \mathsf{na}, \mathsf{nb}|\}_{\mathsf{k(b)}}$
3. $\mathsf{s} \rightarrow \mathsf{i}: \{|\mathsf{b}, \mathsf{kab}, \mathsf{na}, \mathsf{nb}|\}_{\mathsf{k(i)}}, \{|\mathsf{i}, \mathsf{kab}|\}_{\mathsf{k(b)}}$
4. $\mathsf{i} \rightarrow \mathsf{b}: \{|\mathsf{i}, \mathsf{na}, \mathsf{nb}|\}_{\mathsf{k(b)}}, \{|\mathsf{nb}|\}_{\mathsf{na,nb}}$

Figure 2.2: An attack to the Yahalom protocol.

Besides the security goal, there is the question if the protocol is *executable*. Suppose the communication medium is reliable (does not loose, introduce, or alter messages) and all agents behave exactly as the protocol prescribes. Is it then guaranteed that each agent can perform the necessary operations to compose and decompose messages so that both A and B finally get the new shared key KAB? We have convinced ourselves in the above description that this is the case for Yahalom. In general, this question is not directly relevant for security, but suppose, by mistake, we specify a different key in some of the messages when formalizing the protocol; then the protocol could possibly not be executed to the end. As a result, attacks may trivially be excluded from the formalization since the protocol cannot be executed up to the point where a goal is violated. Checking the executability of the protocol minimizes this risk.

We close the discussion of the protocol with the question of what we assume about the abilities of an intruder (or several intruders), and what happens if one of the parties does not stick to the protocol. As already said, we assume the communication medium to be unreliable; it is common to assume the worst-case, namely that an intruder can control the communication medium entirely, i.e. read and intercept all messages and insert messages under any sender identity. Also, if there are several intruders, the worst-case is that they all work together and can thus be modeled as just one powerful intruder. In the following, we will therefore often just talk about *the* intruder. Also, it is obvious that the protocol cannot give any guarantees, if the cryptography is broken, i.e. when the intruder has found a (feasible) way to obtain M from $\{|M|\}_K$.

We have a similar situation, if the server is not honest, in the sense that he does not behave as the protocol demands; in the worst case he is the intruder or a marionette controlled by the intruder (because the intruder, e.g. has hacked the server). In this case, the intruder can obviously read all messages that A and B encrypt with KAB. Moreover, he can for instance pose as B towards A by creating an appropriate message of step 3 for A. We thus need to assume that the cryptography is at least computationally hard to break, and that the server is honest and not corrupted by the intruder. It is valuable to consider also the case of dishonest agents: it is realistic that some parties of a protocol may try to cheat or that their computer has been infected with some kind of malware. Only parties like trusted servers (which are maybe controlled carefully by legal authorities) should be assumed honest. In fact, many of the most subtle attacks that took quite a while to be discovered, are based on dishonest participants, namely the attacks on Yahalom described in the following section and the one on the Needham-Schroeder Public-Key protocol which we explain in Subsection 4.4.2.

2.3 An Attack to Yahalom

As a concrete example of an attack, we describe an attack on Yahalom that was found using the OFMC tool. Figure 2.2 shows a sequence of exchanged messages between concrete agents i, b and s, playing the roles A, B, and S respectively. Here, i stands for *intruder*, who controls the communication medium and can e.g. see the message 2 from b to s. b and s are honest and execute the protocol exactly as described.

Note that all identifiers used in the A&B description that start with an upper-case letter, e.g. A, B and S, are essentially variables or placeholders for the values used in a real protocol execution; we use identifiers that start with lower-case letters, e.g. i, b, and s to denote the concrete values that are used in the attack trace. Namely, na and nb represent the concrete nonces created by i

and b for the protocol variables NA and NB, respectively, during this execution of the protocol. Similarly, kab is the concrete key created by the server as KAB of the protocol. The shared keys of i and b with the server s are denoted by k(i) and k(b), respectively. The function k is in this attack trace (and all subsequent examples) parameterized over only one agent name as we will later assume for simplicity that there is one server s used for role S in all protocol runs.

The protocol execution is in accordance with the specification of the protocol, except for the last message sent by the dishonest agent i. Rather than forwarding the second part of message 3, namely $\{|\mathsf{i}, \mathsf{kab}|\}_{\mathsf{k(b,s)}}$, he uses a part of message 2 that b has sent to s, namely $\{|\mathsf{i}, \mathsf{na}, \mathsf{nb}|\}_{\mathsf{k(b,s)}}$. Note that this makes only sense if we assume that the pair na, nb can be interpreted by the recipient b as a concrete value for the key KAB according to the protocol. More specifically, this assumption refers to details of the implementation of the protocol that are not part of the A&B specification. For instance, if nonces and keys in the implementation of the protocol have a fixed length, then our assumption means that nonces have half the size of a key. Also, we have to assume that there are no other notions of distinction, e.g. *tags* in the messages that indicate whether a given bit-string is meant as a nonce or as a key. Since the attack exploits possible ambiguities in the interpretation of a message, namely the types, it is sometimes called a *type-flaw attack*.

Under the assumption that b can interpret the pair na, nb as KAB, then message 4 can be received by b, since the second part of message 4 has indeed the required form, namely the encryption of b's nonce nb with what appears to be the new shared key, na, nb. Note that i can generate this second part of message 4, since he knows nb from the first part of message 3.

The result of this manipulation by i is that b has now accepted na, nb as the key generated by the server for KAB although this is not the case. This is a violation of an authentication goal (we will formally define authentication goals below).

As a historical remark, note that a very similar attack was displayed in the Clark/Jacob Library [54] with the difference that all participants are honest and the intruder is an outsider who intercepts message 4 and replaces it in a similar way as in our attack. This is a mistake, however: in the variant with honest principals the attack does not work, because the intruder is unable to read the nonce nb unless he is either playing in role A or B. The authors of [73] point out this mistake and have not found any attacks on Yahalom in their analysis with Casper/FDR.

The practical relevance of the attack we have found with OFMC is limited by the fact that it requires a dishonest agent i to participate in the protocol run, so that manipulating the shared key does not give him a significant advantage (as compared to using the key generated by the server). However, this attack is an example that we can use to describe several subtle issues about security protocols, for instance intruder deduction with composed keys. Therefore we will use it as a running example in several parts of this thesis.

2.4 Formal Semantics

While the common A&B notation is often used without a formal definition, it is of course possible to define protocol description languages based on this kind of notation and to define a formal semantics for such a language.

One of the first comprehensive solutions is the Casper/FDR system by Lowe [102]. It has an A&B-style input language that is translated into the process calculus CSP [89], and the FDR model-checker for CSP can be applied to automatically analyze the protocol. This system turned out to be feasible for a large number of protocols and has discovered a number of previously unknown attacks [73, 99].

Millen et al. have devised the CAPSL Integrated Protocol Environment (CIPE) [70], which consists of an A&B-style high-level input language, CAPSL, that is translated into an low-level intermediate format, CIL, based on multi-set rewrite rules. The idea of this intermediate format is that a variety of different analysis tools can be connected to CIPE based on the same input language CIL, so that they do not have to interpret the more high-level CAPSL. One detail—which is semantically important—shows that CAPSL was inspired by Casper: the so-called *Lowe operator*. This operator is used for messages like step 3 of Yahalom, where it is not obvious how

the protocol can be executed, because the receiver cannot fully decrypt the message; the Lowe operator allows to specify explicitly how the message should be parsed by the receiver.

This parsing problem was solved in [94], which is the basis of the Casrul system. For instance, the message exchange of Yahalom can be formulated in this system exactly as in Figure 2.1. Like CIPE, the approach is also based on a high-level language and an intermediate format also based on multi-set rewriting. The semantics that [94] give for A&B notation has become standard (see [44]).

In a nutshell, this semantics can be described as follows. Suppose that all agents, including the intruder, can decrypt messages iff they have the necessary key. Similar to static analysis, we read all messages in the message exchange as concrete messages (as opposed to regarding every atom as a place-holder for the concrete messages exchanged in a real protocol run). Step-by-step we compute the set of messages that an agent can derive from its initial knowledge and the messages it received so far when applying all possible operations it can perform on messages (both encryption and decryption). We formally define this so-called *Dolev-Yao-closure* later in Definition 3.6. From the messages that the agent can derive, we obtain the information how far an agent can decompose an incoming message, and thus we obtain a message pattern describing the format of messages acceptable for this agent at this point. In particular, indecipherable message parts are replaced by fresh variables (so any concrete message term is acceptable in this place). Another complication of message parsing is also correctly handled by this semantics: when an agent receives an indecipherable term M in one message, and the decryption key in a later message, then he can check that M has indeed the appropriate format. The next step is to compute if (and how) the agent can construct the next message of the protocol from the knowledge that it has gathered so far. If it is impossible, the protocol description is refused by this semantics as *unexecutable*.

From this semantics (that refers to the Dolev-Yao closure), it may seem that A&B notation is based on the perfect cryptography assumption. However, this is not the case, as A&B notation only describes how a protocol is supposed to be executed (by honest participants); this is independent from the question whether cryptography may be broken or what an intruder is able to do.

The high-level and low-level language of Casrul were later used in the EU-project AVISS [8] under the names HLPSL (High-Level Protocol Specification Language) and IF (Intermediate Format); several analysis tools were connected, including the OFMC tool described in this work. IF, which has undergone several extensions to handle more sophisticated protocols, is the basic formalism we use to describe protocols and it will be described in detail in Chapter 4. In a successor EU-project, AVISPA [7], HLPSL was replaced by a different formalism, also called HLPSL [48], which is no longer based on A&B notation, but specifies each role individually like IF. As the difference between new HLPSL and IF in terms of abstraction is relatively small, we focus in this thesis on IF.

Chapter 3

Messages as Terms

We start the exposition of our model at the basis: the messages and their term representation, and use this opportunity to review the relevant fundamental concepts.

Messages are represented by terms, where agent names, atomic keys and nonces are represented by constants and all (cryptographic) operations are represented by function symbols. For instance, the function symbol $\{|m|\}_k$ represents the symmetric encryption of message m with key k. The formal concepts behind this are as follows:

Definition 3.1. *The* alphabet Σ *is a finite or countably infinite set of symbols. To each symbol we associate a natural number, its* arity*. Symbols of arity* 0 *are called* constant symbols*, all others are called* function symbols*. The set of symbols of arity* n *is denoted as* Σ_n*.* $\mathcal{V}$ *denotes a countable set of variable symbols disjoint from* Σ*.*

For a set $V \subseteq \mathcal{V}$*, let* $\mathcal{T}_\Sigma(V)$ *denote the set of terms that can be built using symbols of* Σ *and* V*, namely the least set that contains* Σ_0 *and* V*, and that is closed under the following rule:*

$$\text{If } t_1, \ldots, t_n \in \mathcal{T}_\Sigma(V) \text{ and } f \in \Sigma_n \text{ then } f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma(V) \ .$$

In the case that $V = \emptyset$*, we simply write* $\mathcal{T}_\Sigma$*, and this is the set of* ground terms*. We also write* $ground(t)$ *if* $t \in \mathcal{T}_\Sigma$*. For a term* t*, let* $vars(t)$ *be the set of variables that occur in* t*:*

$$vars(t) = \begin{cases} \{t\} & \text{if } t \in \mathcal{V} \\ vars(t_1) \cup \ldots \cup vars(t_n) & \text{if } t = f(t_1, \ldots, t_n) \text{ for some } f \in \Sigma_n \ . \end{cases}$$

Term positions *are represented as sequences of natural numbers where* ϵ *represents the empty sequence, and* $a : s$ *represents the sequence of an element* a *followed by sequence* s*. Then the set of positions of a term* t *is defined as*

$$pos(t) = \begin{cases} \epsilon & \text{if } t \in \Sigma_0 \cup \mathcal{V} \\ \{i : p \mid i \in \{1, \ldots, n\} \wedge p \in pos(t_i)\} & \text{if } t = f(t_1, \ldots, t_n) \ . \end{cases}$$

The symbol at position ϵ *of a term* t *is also called the* root symbol *and we write* $root(t)$*. For a term* t *and a position* $p \in pos(t)$*, we write* $t|p$ *for the subterm of* t *at position* p*:*

$$t|p = \begin{cases} t & \text{if } p = \epsilon \\ t_i|p' & \text{if } p = i : p' \wedge t = f(t_1, \ldots, t_n) \ . \end{cases}$$

Also, we define the replacement of a subterm t *at position* $p \in pos(t)$ *with a term* s *as*

$$t[s]p = \begin{cases} s & \text{if } p = \epsilon \\ f(t_1, \ldots, t_{i-1}, t_i[s]p', t_{i+1}, \ldots, t_n) & \text{if } p = i : p' \wedge t = f(t_1, \ldots, t_n) \ . \end{cases}$$

Symbol	Arity	Meaning	Intrudable
i	0	name of the intruder	yes
$\mathsf{inv}(\cdot)$	1	private-key of a given public-key	no
$\{\cdot\}_\cdot$	2	asymmetric encryption	yes
$\mathsf{sign}(\cdot,\cdot)$	2	digital signature	yes
$\mathsf{open}(\cdot)$	1	obtain message in signature	yes
$\{\!\|\cdot\|\!\}_\cdot$	2	symmetric encryption	yes
$\langle\cdot,\cdot\rangle$	2	pairing/concatenation	yes
$\pi_1(\cdot)$	1	projection to first component of a pair	yes
$\pi_2(\cdot)$	1	projection to first component of a pair	yes
$\cdot(\cdot)$	2	function application	yes
$\mathsf{exp}(\cdot,\cdot)$	2	exponentiation modulo fixed prime p	yes
$\cdot^{-1}$	1	inverse of an exponent	yes
$\cdot \oplus \cdot$	2	bitwise exclusive or	yes
e	0	bit-string of zeros	yes

Table 3.1: Signature Σ used in this thesis for message terms (omitting all constants without special meaning).

We use the convention to typeset symbols of Σ in lower-case sans-serif, e.g. $\mathsf{i} \in \Sigma$ is the name of the intruder, and symbols of $\mathcal{V}$ in upper-case sans-serif, e.g. M. Meta-variables are displayed in standard mathematical italic. Table 3.1 shows the signature Σ that we use for message terms in this thesis, omitting all constants to which no special meaning is attached.

A subset of the symbols are declared as *intrudable* in this table. Intuitively that means whether the intruder can use this symbol to generate terms. This is made more precise below in Definition 3.6 where we define the intruder model. Note that all constants are by default unintrudable, unless said otherwise. We also assume that the signature includes all natural numbers, $\mathbb{N} \subseteq \Sigma_0$, and that they are intrudable.

Modeling terms as messages implies a quite abstract view of the world: random numbers, for instance, which are in reality a sequence of bits, are considered as atoms in the term world. This neglects many issues of the real-world cryptography, where we might consider properties of single bits (e.g. whether the clear-text's last bit is 1). Also there is no notion whatsoever of the length of the sequence of bits which might be important in reality (e.g. for issues like padding or the feasibility of brute-force attacks). And finally, there is no notion of *entropy*, i.e. whether a constant is a really randomly chosen bit-sequence or not: e.g. agent-names can (with high probability) be distinguished from random numbers. We will discuss below when we come to the model of the intruder, how this abstract view can be justified as part of the so-called perfect cryptography assumption.

3.1 Free Algebra

We now define how the terms of $\mathcal{T}_\Sigma$ are interpreted, were our focus is first on the free term algebra:

Definition 3.2. *A* Σ-algebra $\mathcal{A}$ *consists of a set* A*, called the* universe *or* carrier*, and an* interpretation $\mathcal{I}$ *that maps each* $f \in \Sigma_n$ *to a function* $f^\mathcal{I}$ *of type* $A^n \to A$ *(and to an element of* A *in the case* $n = 0$*). The interpretation* $\mathcal{I}$ *is extended homomorphically to terms as follows:*

$$f(t_1, \ldots, t_n)^\mathcal{I} = f^\mathcal{I}(t_1^\mathcal{I}, \ldots, t_n^\mathcal{I}) \ .$$

We write $t_1 \approx_\mathcal{A} t_2$ *if* $t_1^\mathcal{I} = t_2^\mathcal{I}$*. We may omit the index* $\mathcal{A}$ *when it is clear from the context. Given a class of algebrae* $\mathcal{C}$*, we say that algebra* $\mathcal{A} \in \mathcal{C}$ *is* free in $\mathcal{C}$*, iff* $t_1 \approx_\mathcal{A} t_2$ *implies* $t_1 \approx_\mathcal{B} t_2$ *for each* $\mathcal{B} \in \mathcal{C}$*.*

The free (term) algebra *is a Σ-algebra with the universe $\mathcal{T}_\Sigma$ and the following interpretation $\mathcal{I}$: $c^\mathcal{I} = c$ for each $c \in \Sigma_0$ and $f^\mathcal{I}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ for each $f \in \Sigma_n$ where $n > 0$. This algebra is itself also often denoted as $\mathcal{T}_\Sigma$.*

Note that the free term algebra is, as the name suggests, free in the class of all Σ-algebrae. An important property is that $t_1 \approx_{\mathcal{T}_\Sigma} t_2$ iff $t_1 = t_2$, i.e. terms are equivalent in the free term algebra iff they are syntactically identical.

It is standard to interpret the message terms of protocols in the free algebra. This has several consequences:

1. If we have two different constant symbols n_1 and n_2 that represent for instance two random numbers created by honest agents, then these constants are interpreted as different values. Thus, if we use a *fresh* constant symbol (that did not appear before) for every fresh value that an agent creates, then it is guaranteed (by definition) to be different from any value that any agent has created before. The same holds of course also for the constants symbols that represent the names of agents like $\mathsf{a}, \mathsf{b}, \mathsf{i}$: they all represent different names.

2. In the free algebra, it holds that $\{\!|m|\!\}_k \approx \{\!|m'|\!\}_{k'}$ iff $k \approx k'$ and $m \approx m'$. That means two cipher-texts are identical iff they result from encrypting the same clear-text with the same key.[1] In particular, we will model hash functions as function symbols of Σ. Therefore, in the free algebra, hash functions are *collision-free*, i.e. $h(m_1) \not\approx h(m_2)$ for any function symbol h and terms $m_1 \not\approx m_2$.

3. Two terms cannot be equal if they have different root symbols, e.g. $\{\!|m|\!\}_k \not\approx n$ for any terms k and m and constant n. In other words, any constant (like a random number) is different from any cipher-text.

4. We cannot model a decryption operation explicitly as a function symbol of Σ, since clearly for any function symbol dec it holds $\mathsf{dec}(k, \{\!|m|\!\}_k) \not\approx m$—while the defining property of a decryption operation is that, when applied with the correct key, one obtains the original clear-text.

5. Several properties of operators that one may expect do not hold in the free algebra. For instance, one may expect that the concatenation of two messages m_1 and m_2, denoted as $\langle m_1, m_2 \rangle$, is associative, i.e.

$$\langle m_1, \langle m_2, m_3 \rangle \rangle \approx \langle \langle m_1, m_2 \rangle, m_3 \rangle \ .$$

   This is not the case in the free algebra. We will for simplicity use the notation m_1, m_2 for concatenation (without any form of parenthesis around it), a notation which suggests that the operator is associative. For the free algebra, we thus need to fix the associativity for the term m_1, m_2, m_3; we choose to associate to the right, i.e. interpreting this term as $\langle m_1, \langle m_2, m_3 \rangle \rangle$, unless m_1 is itself a concatenation.

Though unrealistic, the first three of the mentioned consequences do make sense in the context of the perfect cryptography assumption discussed below. While in reality there is indeed a small chance of collisions, it should be impossible to systematically exploit the collisions for attacks if the cryptography is strong. We will examine the forth and fifth consequence in the context of algebraic properties in Section 3.5.

3.2 Substitutions, Matching, Unification, Rewriting

Variables in terms are of uttermost importance, both for formulating any kinds of rules on terms and for symbolic approaches, where one summarizes a (usually infinite number) of terms by a single one with variables. This requires a notion of *substitution*, i.e. replacement of variables with

[1] It is straightforward to also model probabilistic encryption e.g. by including fresh nonces into the clear-text.

terms, and a notion of *unification*, i.e. whether for two terms exists a substitution, called *unifier*, under which they are equal, and how to represent the set of all unifiers. An important special case is *matching* where one of the terms to be unified is ground.[2] These problems are well-understood for the free algebra, so we just quickly summarize the most important definitions and results now, see e.g. [17, 96] for more details.

Definition 3.3. *A* substitution σ *is a function from* $\mathcal{V}$ *to* $\mathcal{T}_\Sigma(\mathcal{V})$*. The* domain *of* σ*, denoted by* $dom(\sigma)$*, is the set of variables* v *for which* $\sigma(v) \neq v$ *holds. The* range *of* σ*, denoted* $ran(\sigma)$*, is the set of terms* $\sigma(v)$ *for which* $v \in dom(\sigma)$*. As we only consider substitutions with finite domains, we represent a substitution* σ *with* $dom(\sigma) = \{v_1, \ldots, v_n\}$ *by* $[v_1 \mapsto \sigma(v_1), \ldots, v_n \mapsto \sigma(v_n)]$*. We restrict the notion of substitutions to those with the property* $dom(\sigma) \cap vars(ran(\sigma)) = \emptyset$*.*[3] *The* identity substitution *id is the substitution with* $dom(id) = \emptyset$*. We say that a substitution* σ *is* ground*, and write* $ground(\sigma)$*, if* $\sigma(v)$ *is a ground term for all* $v \in dom(\sigma)$*. We extend* σ *to a homomorphism on terms as expected. We also write* $t\sigma$ *for* $\sigma(t)$*.*

We say two substitutions σ_1 *and* σ_2 *are* compatible*, written* $\sigma_1 \sim \sigma_2$*, if* $v\sigma_1 \approx v\sigma_2$ *for every* $v \in dom(\sigma_1) \cap dom(\sigma_2)$*. The* composition *of* σ_1 *and* σ_2 *is denoted as* $\sigma_1\sigma_2$*. Note that* $\sigma_1\sigma_2 \approx \sigma_2\sigma_1$ *for compatible ground substitutions. For two sets of ground substitutions* S_1 *and* S_2*, we define their* intersection modulo the different domains *as*

$$S_1 \sqcap S_2 = \{\sigma_1\sigma_2 \mid \sigma_1 \in S_1 \ \wedge \ \sigma_2 \in S_2 \ \wedge \ \sigma_1 \sim \sigma_2\}.$$

Since the composition of compatible ground substitutions is associative and commutative, so is the $\sqcap$ *operator.*

Given two substitutions σ *and* τ*, we say that* σ *is an* instance *of* τ *with respect to the set of variables* V*, denoted* $\sigma \succeq_V \tau$ *if there exists a substitution* χ *such that* $x\sigma \approx x\tau\chi$ *for all* $x \in V$*. If* $V = dom(\sigma) \cup dom(\tau)$*, we simply write* $\sigma \succeq \tau$*.*

A unification problem *is given by a finite set of pairs of terms* $\{(s_1, t_1), \ldots, (s_n, t_n)\}$*. The following definitions are with respect to a given unification problem. A substitution* σ *is called a* unifier*, iff* $s_i\sigma \approx t_i\sigma$ *for each* $i \in \{1, \ldots, n\}$*. A unifier* σ *is a* most general unifier *if there is no unifier* τ *such that* $\sigma \succeq \tau$*. A set of unifiers* S *is called* complete *if each unifier* τ *is an instance of a unifier in* S*. A* matching problem *is the special case of a unification problem where all the* s_i *are ground.*

A standard result for unification in the free algebra is that each unification problem has either no unifier or one unique *most general unifier* (*mgu*, for short) such that all unifiers are an instance of the mgu. Moreover, the mgu can be computed by an algorithm *unify* with the property that for a given unification problem $P = \{(s_t, t_1), \ldots, (s_n, t_n)\}$

- $unify(P) = \emptyset$ if P has no unifier, and
- $unify(P) = \{\sigma\}$ where σ is the mgu of P if there is a unifier. Also, σ has in this case the property that no new variables are introduced by unification in the free algebra:

  $$dom(\tau) \cup vars(ran(\tau)) \subseteq vars(P) \ .$$

  Finally, σ has the property that the variables in its domain and range are disjoint:

  $$dom(\tau) \cap vars(ran(\tau)) = \emptyset \ .$$

3.3 Rewriting

We define rewriting in two steps, first abstract rewrite systems and then rewriting based on a set of rewrite rules. This has the advantage that we can define properties like confluence independent of the formalism (like a set of rules) that is used to define the rewrite relation.

[2] One may allow also matching between two symbolic terms, but then matching is not a special case of unification anymore [96]. We will however not consider this in this work.

[3] This is a slight restriction, as we cannot express substitutions like $[x \mapsto f(x)]$ or $[x \mapsto y, y \mapsto x]$. It is however well-known that for matching and unification we never need such substitutions [96].

Definition 3.4. *An* abstract rewrite system *is a pair* $(A, \rightarrow)$ *where* A *is a set and* $\rightarrow$ *a binary relation on* A. $\rightarrow^*$ *and* $\leftrightarrow^*$ *denote the reflexive transitive closure and the symmetric reflexive transitive closure of* $\rightarrow$*, respectively. Also we write* $\leftarrow$ *for the inverse relation.*

$a \in A$ *is called* reducible *iff there is an* $a' \in A$ *such that* $a \rightarrow a'$. *We say that* a' *is a* normal-form of a *iff* $a \rightarrow^* a'$ *and* a' *is not reducible.*

$\rightarrow$ *is called* terminating *or* noetherian *if there is no infinite chain* $a_1 \rightarrow a_2 \rightarrow \dots$, *and* $\rightarrow$ *is called* confluent *iff* $a_1 \leftarrow^* a_0 \rightarrow^* a_2$ *implies that there is an element* $a \in A$ *with* $a_1 \rightarrow^* a \leftarrow^* a_2$.[4]

$\rightarrow$ *is called* convergent *iff it is terminating and confluent. In this case, every element* $a \in A$ *has a unique normal form which is denoted by* $a{\downarrow}$.

The standard way to define a rewrite relation is to use a set of *rewrite rules*; however, there are several extensions and variants of rule-based rewriting. Since the above definitions are based on the general concept of abstract rewriting systems, they are independent from the particularities of rule-based formalisms.

Definition 3.5. *A* rewrite rule *has the form* $l \rightarrow r$ *where* $l, r \in \mathcal{T}_\Sigma(\mathcal{V})$. *A set* R *of rewrite rules induces a rewrite system* $(\mathcal{T}_\Sigma(\mathcal{V}), \rightarrow_R)$ *where the rewrite relation is defined as follows.* $t \rightarrow_R t'$ *holds iff there is a rule* $(l \rightarrow r) \in R$*, a position* $p \in pos(t)$*, and a substitution* σ*, such that* $t|p = l\sigma$ *and* $t' = t[r\sigma]p$. *When the term* t *contains variables, we require that the variables of the rule* $l \rightarrow r$ *are renamed so that* $(vars(l) \cup vars(r)) \cap vars(t) = \emptyset$.

3.4 Dolev-Yao Intruder Deduction

We have defined the term representation for messages and covered basic questions like equality and unifiability of messages. The next step is to define what can be done with messages, namely how they can be composed and decomposed. In particular, the focus is on what new messages the intruder can deduce from a given set of messages that he has for instance observed by eavesdropping on a network.[5] Dolev and Yao [72] have defined such a deduction system, based on the assumption of *perfect cryptography*. This Dolev-Yao model, with several extensions and modifications, has become the standard model for the cryptographic abilities of an intruder for all approaches based on an abstract representation of terms.

The deduction is defined using an operator $\mathcal{DY}(\cdot)$ (the abbreviation stands for Dolev-Yao). This operator takes as input a (usually finite) set of ground terms, representing the set of messages that the intruder already knows, and it yields an infinite set of ground terms as a result, the set of messages that the intruder can deduce. Thus $t \in \mathcal{DY}(M)$ means that the intruder can deduce the term t from his knowledge M.

Definition 3.6. *For a set* M *of ground terms,* $\mathcal{DY}(M)$ *is the least set closed under the rules of Figure 3.1, and* $\mathcal{DY}^{gen}(M)$ *is the least set closed under the rules* G_{axiom} *and* G_{comp} *of Figure 3.1.*

Their names put the rules into two categories: generation rules ($G.$) and analysis rules ($A.$); the closure $\mathcal{DY}^{gen}$ under only the generation rules is a special case that we also refer to as *generate-only deductions*. The G_{axiom} expresses that the given messages M are contained in the closure.

The G_{comp} rule expresses that, for every *intrudable* function symbol f of arity n, the intruder can compose the term $f(t_1, \dots, t_n)$ for arbitrary terms $t_1, \dots, t_n$ that he can derive. Recall that in Table 3.1, we have defined for every function symbol of the standard signature Σ whether it is intrudable or not. Note that the constants are not intrudable, with the exception of the natural numbers (which are also part of Σ) and the special constants i and e. This is necessary, because otherwise the intruder can generate almost every term using the G_{comp} rule.

The actual meaning of the operators and the perfect cryptography assumption become apparent with the analysis rules:

[4] Confluence is equivalent to the *Church-Rosser property*, namely that $a_1 \leftrightarrow^* a_2$ implies that there is an element $a \in A$ with $a_1 \rightarrow^* a \leftarrow^* a_2$.

[5] As said before, one may also use the same deduction system as the basis for a model of honest agents.

$$\frac{}{m \in \mathcal{DY}(M)}\ G_{\text{axiom}}\ (m \in M) \qquad \frac{t_1 \in \mathcal{DY}(M) \quad \cdots \quad t_n \in \mathcal{DY}(M)}{f(t_1, \ldots, t_n) \in \mathcal{DY}(M)}\ G_{\text{comp}}\ (\mathit{intrudable}(f))$$

$$\frac{\langle m_1, m_2 \rangle \in \mathcal{DY}(M)}{m_i \in \mathcal{DY}(M)}\ A_{\text{pair}_i}\ (i \in \{1,2\}) \qquad \frac{\{\!|m_2|\!\}_{m_1} \in \mathcal{DY}(M) \quad m_1 \in \mathcal{DY}(M)}{m_2 \in \mathcal{DY}(M)}\ A_{\text{scrypt}}$$

$$\frac{\{m_2\}_{m_1} \in \mathcal{DY}(M) \quad \mathsf{inv}(m_1) \in \mathcal{DY}(M)}{m_2 \in \mathcal{DY}(M)}\ A_{\text{crypt}} \qquad \frac{\mathsf{sign}(\mathsf{inv}(m_1), m_2)}{m_2 \in \mathcal{DY}(M)}\ A_{\text{sign}}$$

Figure 3.1: The standard Dolev-Yao intruder deduction rules (for the free algebra).

A_{pair_i}: A pair can be decomposed into its components.

A_{scrypt}: A symmetric encryption can only be analyzed, if the intruder also knows the term that was used as the encryption key.

A_{crypt} and A_{sign}: These rules clarify the characteristics of *asymmetric cryptography*, namely encryption and digital signatures. Each term that is encrypted with a *public key* m_1 can only be decrypted using the corresponding *private key* $\mathsf{inv}(m_1)$. Note that in this model, the intruder can always retrieve the clear-text of a digitally signed message, even if he does not know the public key that corresponds to the private key with which the message is signed. This is realistic, as in most implementations of digital signatures, the signed message is included in clear-text, and only a hash-value is actually signed. Note that *verifying* a signature, i.e. checking that it is signed with the proper key, is not part of the intruder model, as it is performed only by honest agents. Due to the free algebra assumption, we cannot have the property $\mathsf{inv}(\mathsf{inv}(m)) \approx m$. $\mathsf{inv}(\cdot)$ is not intrudable: the intruder cannot obtain the private key from a known public key by any rule.[6]

Function Application: Here, the point is the absence of analysis rules, i.e. the intruder cannot get m from $f(m)$. This makes sense for cryptographic hash-functions: it should be computationally hard to find pre-images. Also, as we have described before, due to the free-algebra interpretation, these functions are collision free. This reflects again the perfect cryptography assumption.

open, exp, $\oplus$, e, π_1, and π_2: These symbols make sense only in the context of algebraic properties, therefore we will not consider these operators when working under the free algebra assumption.

A proof that $t \in \mathcal{DY}(M)$ can be presented in form of a tree:

Definition 3.7. *A* Dolev-Yao proof *for the statement* $t \in \mathcal{DY}(M)$ *is a tree with root node* t*, such that all leaf nodes are terms of* M *and each non-leaf node is a term that can be obtained from its successor nodes by one application of a rule of Figure 3.1. For convenience, we may also label the nodes with the name of the rule.*

Example 3.1. In contrast to many other approaches [6, 102, 116] that are limited to atomic keys, our approach can correctly handle composed keys. Let $M = \{\, \{\!|\mathsf{m}|\!\}_{\mathsf{f}(\mathsf{k}_1,\mathsf{k}_2)}, \mathsf{k}_1, \mathsf{k}_2, \mathsf{f} \,\}$, then we have

[6] Some approaches, e.g. [116], use the $\mathsf{inv}(\cdot)$ function also for symmetric keys, with $\mathsf{inv}(k) = k$. This however requires a typed model like the one in Subsection 4.4.4 so that from the type of the key-term the kind of encryption can be determined.

$\mathsf{m} \in \mathcal{DY}(M)$ as follows:

$$\dfrac{\dfrac{}{\{|\mathsf{m}|\}_{\mathsf{f}(\mathsf{k}_1,\mathsf{k}_2)}}\,G_{\text{axiom}} \quad \dfrac{\overline{\mathsf{f}}\,G_{\text{axiom}} \quad \dfrac{\overline{\mathsf{k}_1}\,G_{\text{axiom}} \quad \overline{\mathsf{k}_2}\,G_{\text{axiom}}}{\mathsf{k}_1,\mathsf{k}_2}\,G_{\text{comp}}}{\mathsf{f}(\mathsf{k}_1,\mathsf{k}_2)}\,G_{\text{comp}}}{\mathsf{m}}\,A_{\text{scrypt}}$$

□

3.4.1 Functions

Example 3.1 also shows the use of the function application operator. It may seem confusing at first that we can use any term as a "function symbol". For instance, using the operator $\cdot(\cdot) \in \Sigma$, the intruder may build the term $\mathsf{i}(\mathsf{i})$, i.e. using his own name as a function. Such a term is nonsensical in that the intruder may never achieve anything with it, but the derivability of such terms is not problematic if we consequently see each term and subterm as representing a bit-string to which a priori no type is attached: in this view the intruder (or an honest agent by mistake) could interpret an arbitrary bit-string as the machine code of a function and try to execute it. The result will most certainly be garbage, but it is not necessary to forbid the construction of such terms. The same holds for terms like $\{\mathsf{i}\}_\mathsf{i}$, i.e. when the intruder uses an arbitrary term as an asymmetric key—probably nobody knows the inverse key.

The fact that we can easily define a constant f as above and put it into the intruder knowledge, or even transmit it in messages, allows for a high flexibility; from a logic point of view, this gives us a second-order concept (functions on terms are itself terms) in a first-order term world. Note that if we were to go beyond the function application operator, we could no longer use standard (first-order) unification for term comparison.

The use for function application goes beyond hash-functions. Firstly, Message Authentication Codes (MACs), which are essentially keyed hash-functions like $f_k(m)$, can be represented as $f(k, m)$. Secondly, it is very convenient to use functions to represent key-tables:

Example 3.2. In the case of the Yahalom protocol, we define a constant k that represents a function mapping agents (including the intruder) to their shared keys with the server s, i.e. $\mathsf{k}(\mathsf{a})$ is shared key of a and s. Recall that for simplicity we use only one global server s. (It is also possible to use keytables for pairs of agents, e.g. $\mathsf{k}(\mathsf{a},\mathsf{s})$.) The set of messages that the server s can accept as step 2 is thus described by the message pattern $\mathsf{B}, \{|\mathsf{A}, \mathsf{NA}, \mathsf{NB}|\}_{\mathsf{k}(\mathsf{B})}$. The new shared key KAB generated by the server is not of the form $\mathsf{k}(\cdot)$, however. □

3.4.2 Discussion

We conclude this section with a discussion of the implications of the Dolev-Yao model in the free algebra. One could describe this model as considering all cryptographic operations as black boxes that work perfectly according to an abstract specification, namely that no derivations are possible for an intruder except those that we have explicitly defined—this is reflected by considering the *least* closure under deduction rules. In general, it is however unclear what abstract specification we should require for cryptographic operators so that the verification of protocols within the Dolev-Yao model implies that the protocol is indeed cryptographically sound [5, 21].

As mentioned before, we cannot have explicit decryption operators in the free algebra. As a consequence we have decryption rules of the form "if the intruder knows the key, he can obtain the clear-text". In this model it is impossible that the intruder or an honest agent attempt to decrypt a message with a wrong key. The result of such a decryption should be just some garbage, but an honest agent may continue with whatever results out of the decryption. Moreover, the intruder may be able to distinguish garbage from meaningful messages and in this way find out which of several keys is the right one (see also Chapter 12). Next, we consider how to integrate algebraic properties into the model.

$$x_1 \oplus x_2 \approx x_2 \oplus x_1 \quad (3.1)$$
$$(x_1 \oplus x_2) \oplus x_3) \approx x_1 \oplus (x_2 \oplus x_3) \quad (3.2)$$
$$\mathsf{exp}(\mathsf{exp}(x_1, x_2), x_3) \approx \mathsf{exp}(\mathsf{exp}(x_1, x_3), x_2) \quad (3.3)$$
$$\mathsf{exp}(\mathsf{exp}(x_1, x_2), {x_2}^{-1}) \approx x_1 \quad (3.4)$$
$$({x_1}^{-1})^{-1} \approx x_1 \quad (3.5)$$
$$\mathsf{inv}(\mathsf{inv}(x_1)) \approx x_1 \quad (3.6)$$
$$\mathsf{sign}(\mathsf{inv}(x_1), \{x_2\}_{x_1}) \approx x_2 \quad (3.7)$$
$$\mathsf{open}(\mathsf{sign}(\mathsf{inv}(x_1), x_2)) \approx x_2 \quad (3.8)$$
$$\{|\{|x_2|\}_{x_1}|\}_{x_1} \approx x_2 \quad (3.9)$$
$$x_1 \oplus x_1 \approx \mathsf{e} \quad (3.10)$$
$$x_1 \oplus \mathsf{e} \approx x_1 \quad (3.11)$$
$$\pi_1(\langle x_1, x_2 \rangle) \approx x_1 \quad (3.12)$$
$$\pi_2(\langle x_1, x_2 \rangle) \approx x_2 \quad (3.13)$$

Figure 3.2: The example theory E_{ex}.

3.5 Algebraic Properties

Many security protocols fundamentally depend on the algebraic properties of cryptographic operators, in the sense that the properties are required for the agents to carry out the steps prescribed by their protocol roles, see [62] for a survey. For example, protocols based on the Diffie-Hellman key-exchange, such as the Station-to-Station [71] and IKE [78, 86] protocols, exploit the property of modular exponentiation that $(g^x)^y \bmod p = (g^y)^x \bmod p$. Without this property, these protocols could not even be executed. Hence, unlike the practice of abstracting from the concrete behavior of cryptography, we cannot ignore the algebraic properties on which such protocols are based. Another, less central, aspect of incorporating algebraic properties is that it gets us closer to the real cryptography, and may allow one to detect attacks for which the free algebra model is blind. For instance, [62] gives an example of a protocol that was proven correct in the free algebra, but that has an attack when taking algebraic properties of a cryptographic operator into account.

The incorporation of algebraic properties makes several basic problems like term comparison and unification considerably more difficult, which is the reason why many formal approaches to protocol analysis stick with the free algebra and thus a smaller scope of protocols.

3.5.1 Equational Theories

Definition 3.8. *An equation has the form $s \approx t$ where $s, t \in \mathcal{T}_\Sigma(\mathcal{V})$. The* equational theory $\approx_E$ *induced by a set of equations E is the congruence relation $\leftrightarrow^*_{R(E)}$ induced by the set of rewrite rules $R(E) = \{l \to r \mid (l \approx r) \in E \vee (r \approx l) \in E\}$.*[7] *For brevity, we may call E itself an equational theory.*

The equivalence class $[t]_{\approx_E}$ *of a term t is the set $\{s \mid t \approx_E s\}$. For a set of terms T, we also write $[T]_{\approx_E}$ for $\bigcup_{t \in T}[t]_{\approx_E}$. The* quotient algebra *of the term algebra under the congruence on terms, written $\mathcal{T}_\Sigma(\mathcal{V})/_{\approx_E}$, has as the carrier the set of equivalence classes $[t]_{\approx_E}$ for each $t \in \mathcal{T}_\Sigma(\mathcal{V})$, and interprets symbols as follows:*

$$f^{\mathcal{I}}([t_1]_{\approx_E}, \ldots, [t_n]_{\approx_E}) = [f(t_1, \ldots, t_n)]_{\approx_E} .^{8}$$

[7] Birkoff's theorem states that $\approx_E$ is equivalent to the least reflexive relation that contains E (interpreted as a relation on terms) and that is closed under symmetry, transitivity, congruence, and substitutivity [32].

[8] This is well-defined, even though it is ambigous which representative t_i of the equivalence class $[t_i]_{\approx_E}$ is chosen,

The ground word problem *for a theory E is the problem of deciding $s \approx_E t$ for arbitrary $s, t \in \mathcal{T}_\Sigma$. All concepts defined in Definition 3.3 that are based on the equality $\approx$ of the considered algebra are defined accordingly for an equational theory E, i.e. with respect to $\approx_E$. We prepend the theory as in E-instance and E-unifier to indicate the respective algebra.*

Given a set of equations E and a rewrite system $(A, \to)$, we define the modular *rewrite system $(\{[a]_{\approx_E} \mid a \in A\}, \to_E)$ on E-equivalence classes of elements of A as follows:*

$$[t]_{\approx_E} \to_E [s]_{\approx_E} \text{ iff } t' \to s \text{ for some } t' \in [t]_{\approx_E}$$

For a rule based rewrite relation $\to_R$, we write $\to_{R/E}$ for its modular extension. For notational simplicity, we may also write $t \to_{R/E} s$, for terms s and t, rather than $[t]_{\approx_E} \to_{R/E} [s]_{\approx_E}$. Similarly, we may also write $t{\downarrow}$ for $[t]_{\approx_E}{\downarrow}$.

Observe that in the quotient algebra two terms are equal iff they are equivalent due to $\approx_E$. More formally, the quotient algebra is free in the class of algebrae that interpret all E-equivalent terms as equal. This is exactly what we want for the Dolev-Yao intruder: different terms are interpreted differently unless the theory E implies their equivalence. Therefore, like in the free algebra, there are no "collisions" between different terms except for the equivalences that we have deliberately introduced.

Example 3.3. Figure 3.2 displays an algebraic theory formalizing relevant properties used in many protocols, including those based on the Diffie-Hellman key-exchange. We will refer to this theory as E_{ex} (see also the description of the signature Σ in Table 3.1), and use it as a standard example throughout this work. Also, we use the convention to denote variables in algebraic equations as $x, y, \ldots$, to distinguish them from other variables in our presentation.

The property required for the Diffie-Hellman key-exchange for instance is (3.3). Note that, as is often done, we leave implicit the modulus of exponentiation in E_{ex}: instead of $\mathsf{exp}(g, x)\ mod\ p$ we write simply $\mathsf{exp}(g, x)$, assuming that exponentiation is always performed using the same (publicly known) modulus. Another equation for exponentiation is (3.4), which expresses that two exponentiations with inverse exponents cancel each other out. Note that we use $\cdot^{-1}$ to denote the inverse of an exponents; in contrast, the inverse of a public key is denoted by $\mathsf{inv}(\cdot)$. Both these functions have the property that two times inverting is the identity ((3.5) and (3.6)). Recall that $\cdot^{-1}$ is intrudable (while $\mathsf{inv}(\cdot)$ is not), and it depends on the protocol one wants to analyze, if $\cdot^{-1}$ should indeed be intrudable: in Diffie-Hellman for instance, all exponentiations are performed modulo a commonly known prime number and in this scenario everybody can compute inverse exponents. In contrast, this does not hold e.g. when using RSA encryption where the modulus is not a prime number and one can construct inverse exponents only when one knows the prime factors of the modulus.

The equations (3.1), (3.2), (3.10), and (3.11) describe the properties of bitwise exclusive OR. The rest of the equations express the properties of the explicit decryption and decomposition operations, e.g. applying projection functions on a pair yields the components of the pair. There are, of course, other properties one may include, such as the associativity of concatenation.[9] Note that the OFMC tool is not specialized to such an algebraic theory, but allows the user to specify a theory from a large class of theories that is defined in Part III. □

On this basis, we define a variant of the Dolev-Yao intruder, the *algebraic Dolev-Yao intruder*, which is parameterized over a set of equations E:

Definition 3.9. *For a set M of ground terms and a set of equations E, we define*

$$\mathcal{DY}_E(M) = [\mathcal{DY}^{gen}(M)]_{\approx_E} .$$

since the equivalence class on the right-hand side is independent of this choice.

[9] We suggest in general not to include this property. The reason is that it will lead to meaningless attacks in almost every protocol, because the length of components is abstracted away in the term based models. In every reasonable implementation there is a method to determine the length of fields such as fixed lengths, offsets, or special tags. If the decomposition of a message into fields is uniquely defined, associativity does not play a role.

Note that the definition is built using the $\mathcal{DY}^{gen}$ closure which refers to terms, not to their interpretations in some algebra. In other words, the closure does not take any algebraic properties into account, but rather algebraic properties are "added after the closure". Such a separation between deduction and algebraic properties were in general not possible if we had included analysis rules in the closure.

The analysis rules of $\mathcal{DY}$ are redundant as they are entailed by the algebraic theory E_{ex}:

Theorem 3.1. *For any set M of ground terms, $\mathcal{DY}_{E_{ex}}(M)$ is closed under the analysis rules of Figure 3.1.*

Proof. Let $k, \{\!|m|\!\}_k \in \mathcal{DY}_{E_{ex}}(M)$ for some ground terms k and m. Then there exist terms $k', c \in \mathcal{DY}^{gen}(M)$ with $k' \approx_{E_{ex}} k$ and $c \approx_{E_{ex}} \{\!|m|\!\}_k$. Since $\{\!|\cdot|\!\}_\cdot$ is intrudable, we thus have also $\{\!|c|\!\}_{k'} \in \mathcal{DY}^{gen}(M)$. Since $\{\!|c'|\!\}_{k'} \approx_{E_{ex}} \{\!|\{\!|m|\!\}_k|\!\}_k \approx_{E_{ex}} m$, we have $m \in \mathcal{DY}_{E_{ex}}(M)$. Thus $\mathcal{DY}_{E_{ex}}$ is closed under the analysis rule A_{scrypt}. The proof for the other analysis rules goes similar, namely A_{pair_i} can be derived using the operators π_1 and π_2, respectively, A_{crypt} using the operator $\mathsf{sign}(\cdot,\cdot)$, and A_{sign} using the operator $\mathsf{open}(\cdot)$. □

We thus have a model of an algebraic Dolev-Yao intruder that subsumes the model of the free algebra and includes equalities, and in particular allows for the modeling of explicit decryption operators.

Chapter 4

Protocols as State Transition Systems

Up to this point, we have considered the question of how to model messages and the abilities of an intruder to synthesize and analyze messages. This is the basis of the problem of modeling a protocol, namely the way honest agents and intruder interact with each other over some communication medium that is (at least partially) controlled by the intruder. The formal-methods view on such a situation is to regard protocols as a *distributed system*, i.e. a number of *processes* that run in parallel and that interact with each other. In our case, these processes are the honest agents who execute the protocol and the intruder; but also the communication medium itself may be regarded as a process.

There has been proposed a large variety of formalisms for modeling distributed systems and the properties we want to prove about them. Even on a very basic level, there are different ways how to define a system, e.g. as a state transition system, as a Kripke structure, or as a set of traces. On a higher level, there are many ways to describe a system such as temporal logics, process calculi, or rewrite rules.

The choices that we have made in this work to model a system are the following. We define an infinite set of terms to be the *state space*. One state in this state space is the *initial state* S_0. There is a binary relation $\rightarrow$ on states, the *transition relation*. The set of all states S such that $S_0 \rightarrow^* S$ is the set of *reachable states*. Finally, to express the goals, i.e. the properties we want to prove or disprove about the protocol, we specify a subset of states, the *attack states*, in which the desired property is violated. The verification question is then, whether the set of reachable states and the set of attack states have an empty intersection or not. If yes, the protocol has the desired property, otherwise we have a counter-example, which can be represented by an attack trace, i.e. a sequence of $S_0 \rightarrow S_1 \ldots \rightarrow S_n$ of states where S_n is an attack state.[1] For defining the transition relation and the set of attack states, we will use rewrite rules. More specifically, we will use a particular form of rewriting, *set rewriting*, which we will introduce next.

There are several reasons for our choices. Firstly, we already use terms to model messages, using concepts from "the term world" like substitution and unification. Rewriting is a technique that is closely related to this area. We thus get a very expressive formalism to model state transition systems based on rewriting for a relatively small extra fee so to speak (in terms of new concepts we have to introduce). Secondly, for many applications we need algebraic properties and, moreover, we later turn to symbolic approaches, where a set of terms is represented by a term with variables and constraints. Both concepts are closely related to rewriting and their integration is painless in the sense that we do not have to introduce and integrate concepts that are in some sense alien to the basic approach. For the above reasons, many researchers have used rewriting for protocol analysis, including [68, 94, 110]. The EU-projects AVISS and AVISPA [8, 7], as part

[1] It is usually sufficient (and convenient) for the user of a protocol analysis tool that simply the sequence of messages that were exchanged along the trace is displayed, rather than the entire states.

of which our tool OFMC has been developed, have built on this work and used a rewrite based formalism, IF [14], as the common input language to the analysis tools.

4.1 Set Rewriting

Several approaches have been based on *multi-set rewriting*, for instance Maude [105]. The basic idea is to introduce an operator "." which is associative and commutative (AC), i.e. for which we have the following algebraic equations:

$$\begin{aligned} (\mathsf{A}.\mathsf{B}).\mathsf{C} &\approx \mathsf{A}.(\mathsf{B}.\mathsf{C}) \\ \mathsf{A}.\mathsf{B} &\approx \mathsf{B}.\mathsf{A} \end{aligned}$$

A state is then a term of the form $t_1.t_2.\cdots.t_n$ with subterms t_i that do not contain the "."-operator. We call these subterms *facts*. Note that $t.t \not\approx t$, thus a state is a *multi-set* of facts.

Multi-sets are in several cases not so convenient for our formal argumentation as normal sets, since we must always care about duplicates and number of occurrences. So we have chosen to add a third equality for the "." operator, namely idempotence (I):

$$\mathsf{A}.\mathsf{A} \approx \mathsf{A}$$

This allows us to consider each state as a *set* of facts. While an ACI operator is more difficult to handle in rewriting than an AC operator, the advantage is the conceptual simplicity of sets (as opposed to multi-sets). Also the form of facts in states and rules that we will introduce below (e.g. a fact cannot be a variable) makes the matching and unification a lot easier.

The rewrite rules R that we will define will be applied modulo an equational theory E containing at least the above ACI equations for the set-operator (which we will call the "." from now on), i.e. we will use the modular rewrite relation $\rightarrow_{R/E}$. This allows us to formalize rules that can be applied to a state S iff their left-hand side matches a subset of S, and that on application replace this subset with the right-hand side under the matching substitution.

Example 4.1. Using the operator $\mathsf{iknows}(m)$ to indicate that the intruder knows the message m, we can encode the Dolev-Yao rule G_{comp} for operator $f = \{\cdot\}_\cdot$ by the rewrite rule

$$\mathsf{iknows}(\mathsf{M}).\mathsf{iknows}(\mathsf{K}) \Rightarrow \mathsf{iknows}(\mathsf{M}).\mathsf{iknows}(\mathsf{K}).\mathsf{iknows}(\{\mathsf{M}\}_\mathsf{K}) \ .$$

This means that whenever a state contains the fact that the intruder knows some messages that match with M and K (i.e. any messages), then we can make a transition to a state where he additionally knows $\{M\}_K$. Here, the matched facts of the intruder knowledge $\mathsf{iknows}(M)$ and $\mathsf{iknows}(K)$ are not necessarily different (as we are not using multi-set rewriting). For instance we have that

$$\mathsf{iknows}(\mathsf{i}) \rightarrow_{R/E} \mathsf{iknows}(\mathsf{i}).\mathsf{iknows}(\{\mathsf{i}\}_\mathsf{i}) \ .$$

□

Since we will not have any rules where messages are deleted from the intruder knowledge, we define the following syntactic sugar: for the special case of $\mathsf{iknows}(\cdot)$ facts, every left-hand side occurrence is implicitly repeated on the right-hand side, so we do not have to repeat it there. Thus we may write the above rule as

$$\mathsf{iknows}(\mathsf{M}).\mathsf{iknows}(\mathsf{K}) \Rightarrow \mathsf{iknows}(\{\mathsf{M}\}_\mathsf{K}) \ .$$

4.1.1 Extensions

Before we define the precise syntax and semantics of the Intermediate Format, we integrate a number of extensions, which make the formalization of many protocols easier. Rewriting without extensions is already a powerful, expressive concept (e.g. combinatory logic [65] can be encoded by rewrite rules), but without extensions, the realization of some relevant concepts like negative conditions for transitions is technically involved and not suitable for automated analysis.

$$
\begin{array}{rcl}
\mathit{ProtocolDescr} & ::= & \mathit{State}, \mathit{Rule}^*, \mathit{AttackRule}^* \\
\mathit{Rule} & ::= & \mathit{LHS} \; =[\mathcal{V}^*]\Rightarrow \; \mathit{RHS} \\
\mathit{AttackRule} & ::= & \mathit{LHS} \\
\mathit{LHS} & ::= & \mathit{State}\ \mathit{NegFacts}\ \mathit{Condition} \\
\mathit{RHS} & ::= & \mathit{State} \\
\mathit{State} & ::= & \mathit{Fact}\ (\ .\ \mathit{Fact})^* \\
\mathit{NegFacts} & ::= & (\ .\ \mathsf{not}(\mathit{Fact})\ )^* \\
\mathit{Fact} & ::= & \mathcal{F}\ (\ \mathcal{T}_\Sigma(\mathcal{V})\ (\ ,\ \mathcal{T}_\Sigma(\mathcal{V}))^*\ ) \\
\mathit{Condition} & ::= & (\ \wedge\ \mathcal{T}_\Sigma(\mathcal{V})\ \not\approx\ \mathcal{T}_\Sigma(\mathcal{V})\ )^*
\end{array}
$$

Figure 4.1: The Syntax of the Intermediate Format.

The first extension concerns the creation of fresh data during the protocol execution. For this reason, we formalize rules of the form $l =[V]\Rightarrow r$ where $V \subseteq \mathcal{V}$, which expresses that the variables of V are substituted during the transition with fresh constants, i.e. constants that have never been used before.

The other two extensions concern (negative) conditions under which a rule can be applied. Firstly, the left-hand side of a rule may contain inequalities of the form $s \not\approx t$ for terms s and t, excluding rule matches σ for which $s\sigma \approx_E t\sigma$. Secondly, the left-hand side of a rule may also contain negative facts of the form $\mathsf{not}(f)$ for a fact f, excluding a rule application to a state S under the match σ, if $f\sigma \in S$.

Finally, the way Example 4.1 formalizes the intruder deduction has drawbacks. Every deduction step of the intruder takes one transition. For most automated analysis methods, the intruder deduction must be handled somewhat differently in order to obtain a terminating algorithm. We will therefore assign a special meaning to this fact-symbol in the formal semantics below: if $\mathsf{iknows}(m)$ appears on the left-hand side of a rule, then any message in the $\mathcal{DY}$-closure of the messages that the intruder knows can be matched with m. In this way, we can avoid explicit deduction rules of the intruder in the style of Example 4.1 entirely.

4.2 The Intermediate Format

4.2.1 Syntax

We first define the syntax formally by the grammar of Figure 4.1 and additional conditions, and then give an intuition.

Definition 4.1. *Figure 4.1 shows a context-free grammar describing the syntax of the Intermediate Format (IF), which is parameterized over two disjoint signatures Σ and $\mathcal{F}$ (see Figure 4.2 for an example), and a set of variables $\mathcal{V}$. To distinguish parentheses of IF from those of the grammar exposition, we have used bold font for the latter kind. In the following, we will refer to the language described by the non-terminals of the grammar as* IF protocol description*,* IF rule*, etc., and omit the prefix "IF" when there is no danger of confusion. Also, we will refer to lists of rules, facts etc. simply as sets, as the order and repetitions in lists are immaterial due to the semantics.*

As part of the syntax of IF, we also require the following conditions to be met:

1. *For every fact f, we require that it is contained in $\mathcal{T}_{\Sigma\cup\mathcal{F}}(\mathcal{V})$, i.e. that the arity of the employed fact symbol agrees with the length of the list of arguments.*

2. *The initial state is a list of ground facts.*

3. *For each transition rule $L =[V]\Rightarrow R$ it must hold that $vars(L) \cap V = \emptyset$ and $vars(R) \subseteq vars(L) \cup V$.*

Symbol	Arity	Meaning
iknows	1	Message known to the intruder
msg	1	A message on the network
state.$(\cdot)$	2	Local state of an honest agent, as index the role the agent is playing in
contains	2	An element is contained in a set
secret	2	Declare that a message shall be a secret between a set of agents.
witness	4	Declare that a message is meant in a certain context
request	5	Declare that a message is accepted within a certain context
attack	0	Indicates that an attack has occurred

Figure 4.2: The signature $\mathcal{F}$ of fact symbols.

4. *For each left-hand side consisting of positive facts P, negative facts N, and conditions C, we require that $vars(P) \supseteq vars(N) \cup vars(C)$.*

5. *The negative facts may not contain an* iknows *fact.*

6. *We require that* msg *facts occur neither in the initial state nor in the attack rules. Moreover, there appears at most one* msg *fact on each side of a rule, and only in positive form.*

As syntactic sugar, we write simply $L \Rightarrow R$ for $L =[\emptyset]\Rightarrow R$. Also, as discussed above, since the intruder never forgets messages in our models, an iknows$(\cdot)$ *fact on the left-hand side is implicitly repeated also on the right-hand side.*

An IF protocol description consists of three components, namely an initial state, a set of rules that describes a transition relation on states, and a set of attack rules that describe the attack states. A state is a set of facts. (As already said, we refer to the lists defined in the syntax as sets and use standard set operations on them. Also, note that all such sets are finite according to the syntax.)

A fact is a special kind of term where the root symbol is from the special signature $\mathcal{F}$ (that is disjoint from Σ and $\mathcal{V}$) and the subterms are standard message terms from $\mathcal{T}_\Sigma(\mathcal{V})$. The signature $\mathcal{F}$ used in this thesis is displayed in Figure 4.2. Note that these symbols have no built-in meaning, except iknows$(\cdot)$ for which we define a particular interpretation in the semantics of IF. The "meaning" displayed in Figure 4.2 is merely to give an intuition of how these facts are used, and will be explained in detail at hand of concrete examples below. An important property of our syntax of states is that the set-operator "." can only occur on the "toplevel" of states, i.e. linking facts which must have a symbol of $\mathcal{F}$ as their root symbol and thus cannot be variables themselves. Not allowing arbitrary terms composed with the set-operator allows for an easier handling of states as sets of facts.

A rule consists of a left-hand side, a set of variables, and a right-hand side. Both sides of a rule consists of state, and, additionally, the left-hand side may have negative facts and conditions. We will also call the facts of the state on the left-hand side the positive facts of the left-hand side. A negative fact has the form not(f) for a fact f. A condition is a conjunction of inequalities of messages. Negative facts intuitively mean that the rule can only be applied to a state that does not contain any of these facts (in positive form). Conditions intuitively mean that the rule can only be applied if the inequalities of terms hold. Attack rules are a variant of rules, namely they do not have a right-hand side. Intuitively, attack rules describe a subset of the set of states, namely the states to which the attack rule can be applied.

The additional conditions on the syntax ensure that we can define the semantics of an IF protocol description as a transition system where all states are ground (in particular, the transition rules do not introduce variables into the state-space). Condition 6 is needed for the step compression technique in Subsection 4.4.3.

With respect to the definition of IF in [14], we do not allow for negative iknows facts and msg facts; also, we require that all variables in negative facts and conditions must also occur in positive facts of the left-hand side of the respective rule. These restrictions are crucial for the

decidability and semi-decidability results in Section 5.1 and Section 11.3. Also, we do not consider type annotations and only give an overview of this issue in Subsection 4.4.4

4.2.2 Semantics

We now define the rewrite relation on ground terms that is induced by a set R of IF rules, which is the core of the semantics. We do this first for the free algebra interpretation of terms; since we have extended rules with concepts like negation, we cannot directly use rewriting modulo the equations (ACI) for the set-operator (and neither can we do that for taking arbitrary algebraic theories into account).

Definition 4.2. *Let $r = L -[V]\Rightarrow R$ be an IF rule. Let further P be the set of positive facts of L without the* iknows *facts; let M be the set of message terms m such that* iknows$(m) \in L$*; let N be the set of negative facts of L; and let C be the set of conditions of L.*

For the free algebra, we define the rewrite relation $\Rightarrow_r$ on ground states as follows: $S \Rightarrow_r T$ holds iff there is a ground substitution σ with $dom(\sigma) = vars(L)$ such that

- $P\sigma \subseteq S$,
- $M\sigma \subseteq \mathcal{DY}(\{m \mid \mathsf{iknows}(m) \in S\})$,
- $N\sigma \cap S = \emptyset$,
- *for each $s \neq t \in C$ it holds that $s\sigma \not\approx t\sigma$, and*
- $T = (S \setminus P\sigma) \cup R\sigma\tau$,

where τ is an injective substitution with $dom(\tau) = V$ and $ran(\tau)$ is a set of fresh constants that are not intrudable.[2]

For a set R of rewrite rules, we define $\Rightarrow_R$ as the union of $\Rightarrow_r$ for each $r \in R$.

This definition expresses that there must be a match σ such that the rule is applicable to the state S under the match, i.e. the positive facts are contained in S (where the iknows facts of the rule must be contained only in the $\mathcal{DY}$-closure of the intruder knowledge of S), the negative facts are not contained in S, and all inequalities hold. The successor state is obtained by removing the positive facts (without iknows facts) and adding the right-hand side facts, where all fresh variables V are substituted for fresh constants. As no iknows facts are removed, we have ensured that the intruder knowledge monotonically grows, even when iknows facts are not repeated on the right-hand side. Finally observe that for $S \rightarrow_R T$, the state T must be ground if S is, because the only new facts in T (that were not present in S) are those of $R\sigma\tau$, which is ground. Since the initial state of an IF protocol description is a ground state (see the restrictions in Subsection 4.2.1), it follows by induction that all reachable states are ground.

Next we define the transition relation with respect to an algebraic theory E:[3]

Definition 4.3. *Let r be an IF rule, and L, V, R, P, M, N, and C be as in Definition 4.2. Let E be an equational theory, then we define the rewrite relation $\Rightarrow_{r/E}$ on ground states as follows: $S \Rightarrow_{r/E} T$ holds iff there is a ground substitution σ with $dom(\sigma) = vars(L)$ such that*

- $[P]_{\approx_E}\sigma \subseteq [S]_{\approx_E}$,
- $[M\sigma]_{\approx_E} \subseteq \mathcal{DY}_E(\{m \mid \mathsf{iknows}(m) \in S\})$,

[2] We assume that there is a deterministic mechanism to create fresh constants, e.g. based on a global counter.

[3] We cannot do this in the style of Definition 3.8, by simply considering E-equivalence classes of terms (i.e. states) due to the negative facts and inequalities in the rules. For instance, consider a rule which has on the left-hand side a positive fact f and a negative fact f', such that $f \neq f'$, but $f \approx_E f'$. Then according to our intuition, the rule is not applicable modulo E to any state S. However, if we had defined $\Rightarrow_{R/E}$ based on $\Rightarrow_R$ as in definition Definition 3.8, then the rule were applicable to a state S that contains a term that is an instance of f but not of f' (since $S \in [S]_{\approx_E}$).

- $[N\sigma]_{\approx_E} \cap [S]_{\approx_E} = \emptyset$,
- *for each* $s \not\equiv t \in C$ *it holds that* $s\sigma \not\approx_E t\sigma$, *and*
- $T = (S \setminus [P\sigma]_{\approx_E}) \cup R\sigma\tau$,

where τ *is an injective substitution with* $dom(\tau) = V$ *and* $ran(\tau)$ *is a set of fresh constants that are not intrudable.*[4]

For a set R *of rewrite rules, we define* $\Rightarrow_{R/E}$ *as the union of* $\Rightarrow_{r/E}$ *for each* $r \in R$.

We have defined the algebraic transition relation not on E-equivalence classes of states, but rather on states themselves. Observe that also T is a state (and not an E-equivalence class of states), obtained from S by removing all facts that are E-equivalent to fact of $P\sigma$ and adding the facts of $R\sigma\tau$. Like in the free algebra case, if S is a ground state, then so is T, because $R\sigma\tau$ is ground.

We now define what it means that a protocol description is safe:

Definition 4.4. *For a set of IF rules* R, *and an IF initial state* S_0, *we define the set of reachable states in the free algebra as*

$$reach(I, R) = \{S \mid S_0 \Rightarrow_R^* S\} .$$

The set AttackStates is the set of all IF states that contain the fact symbol attack.

An IF protocol description (I, R, A) *is called* safe in the free algebra *iff*

$$reach(I, R \cup \overline{A}) \cap AttackStates = \emptyset ,$$

where $\overline{A}$ *transforms every IF attack rule* $a \in A$ *into the IF rule* $a \Rightarrow a$.attack. *For an equational theory* E, *the definitions are adapted to* reachable states modulo E *and* safety modulo E *as expected, by using the* $\Rightarrow_{R/E}$ *relation.*

Finally, we say that a protocol description (I, R, A) *has a* bounded number of sessions (modulo E), *iff* $\Rightarrow_R$ ($\Rightarrow_{R/E}$) *is terminating.*

This definition expresses that we extend attack rules to normal rules by repeating all positive facts from the left-hand side on the right-hand side, and adding the fact attack. Then the question of safety is simply whether a reachable state contains the fact attack.

4.3 Agents Model

After our exposition of the syntax and semantics of IF, we now describe how we can model protocols within IF. Note that the following sections only give examples of possible ways to model security protocols. This gives us the opportunity to discuss in detail the assumptions and ideas that are implicit in the protocol models and that one usually lacks the space to discuss. Note that the methods and theoretical results presented in the following parts are independent of the choices made in the modeling and hold for every IF protocol description.

The first step in modeling the protocol is the model of the honest agents, i.e. those who execute the protocol exactly as it is defined, not trying to cheat. Figure 4.3 shows the IF rules for the example of the Yahalom protocol; at hand of this example we discuss our model of honest agents.

Let us first consider an example of an initial state, since it may otherwise be unclear how these rules can ever be applied:

Example 4.2. For the Yahalom protocol, we can for instance define the following initial state:

$$\mathsf{state}_{\mathcal{A}}(0, \mathsf{sess1}, \mathsf{a}, \mathsf{b}).\mathsf{state}_{\mathcal{B}}(0, \mathsf{sess1}, \mathsf{b}).\mathsf{state}_{\mathcal{A}}(0, \mathsf{sess2}, \mathsf{a}, \mathsf{i}).\mathsf{state}_{\mathcal{B}}(0, \mathsf{sess3}, \mathsf{b}).$$
$$\mathsf{iknows}(\mathsf{a}).\mathsf{iknows}(\mathsf{b}).\mathsf{iknows}(\mathsf{i}).\mathsf{iknows}(\mathsf{k}(\mathsf{i}))$$ □

[4] Again, we assume that there is a deterministic mechanism to create fresh constants, e.g. based on a global counter.

$$\begin{array}{ll}
\text{Role } \mathcal{A}: & \mathsf{state}_{\mathcal{A}}(0,\mathsf{ID},\mathsf{A},\mathsf{B},) \\
& \quad =[\mathsf{NA}]\Rightarrow \mathsf{state}_{\mathcal{A}}(1,\mathsf{ID},\mathsf{A},\mathsf{B},\mathsf{NA}).\mathsf{msg}(\mathsf{A},\mathsf{NA}) \\
& \\
& \mathsf{state}_{\mathcal{A}}(1,\mathsf{ID},\mathsf{A},\mathsf{B},\mathsf{NA}).\mathsf{msg}(\{|\mathsf{B},\mathsf{KAB},\mathsf{NA},\mathsf{NB}|\}_{\mathsf{k(A)}}.\mathsf{BsPart}) \\
& \quad \Rightarrow \mathsf{state}_{\mathcal{A}}(2,\mathsf{ID},\mathsf{A},\mathsf{B},\mathsf{NA},\mathsf{NB},\mathsf{KAB}).\mathsf{msg}(\mathsf{BsPart}.\{|\mathsf{NB}|\}_{\mathsf{KAB}}) \\
& \\
\text{Role } \mathcal{B}: & \mathsf{state}_{\mathcal{B}}(0,\mathsf{ID},\mathsf{B}).\mathsf{msg}(\mathsf{A},\mathsf{NA}) \\
& \quad =[\mathsf{NB}]\Rightarrow \mathsf{state}_{\mathcal{B}}(1,\mathsf{ID},\mathsf{B},\mathsf{A},\mathsf{NA},\mathsf{NB}).\mathsf{msg}(\mathsf{B},\{|\mathsf{A},\mathsf{NA},\mathsf{NB}|\}_{\mathsf{k(B)}}) \\
& \\
& \mathsf{state}_{\mathcal{B}}(1,\mathsf{ID},\mathsf{B},\mathsf{A},\mathsf{NA},\mathsf{NB}).\mathsf{msg}(\{|\mathsf{B},\mathsf{KAB},\mathsf{NA},\mathsf{NB}|\}_{\mathsf{k(B)}}.\{|\mathsf{NB}|\}_{\mathsf{KAB}}) \\
& \quad \Rightarrow \mathsf{state}_{\mathcal{B}}(2,\mathsf{ID},\mathsf{B},\mathsf{A},\mathsf{NA},\mathsf{NB},\mathsf{KAB}) \\
& \\
\text{Role } \mathcal{S}: & \mathsf{msg}(\mathsf{B},\{|\mathsf{A},\mathsf{NA},\mathsf{NB}|\}_{\mathsf{k(B)}}) \\
& \quad =[\mathsf{KAB}]\Rightarrow \mathsf{msg}(\{|\mathsf{B},\mathsf{KAB},\mathsf{NA},\mathsf{NB}|\}_{\mathsf{k(A)}}.\{|\mathsf{B},\mathsf{KAB},\mathsf{NA},\mathsf{NB}|\}_{\mathsf{k(B)}})
\end{array}$$

Figure 4.3: IF Rules for the Yahalom protocol.

Each state fact represents the local state of one process participating in the protocol. These processes do not necessarily represent different agents, as one agent can be involved in several *sessions* (protocol executions) at the same time.[5] As an index of each state fact, we have an identifier of the role the agent is playing; although these role identifiers are constants (of Σ_0), we use calligraphic upper-case such as $\mathcal{A}$ for distinction from other constants. The second argument is a concatenation of message terms representing the agents local state. As a convention, the first three elements in this list are the *step number*, the *session identifier*, and the agent's name. The step number distinguishes the different steps of the protocol execution by the agent. The session identifiers, sess1, sess2, sess3, are used to distinguish several state facts with otherwise identical arguments in the initial state. Note that we can give the same session identifiers to different state facts if they are playing in different roles, to indicate which "belong to the same session". This is not further reflected in the transition rules, but helpful for the output of attack traces. For the agent names we use constants like $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$ (and i is the intruder). Moreover, the initial state of agents may include the name of the agent with whom to start a session, e.g. we have above two sessions of an agent a in role $\mathcal{A}$, one for communication with b, and one with i. (Observe that for that session, there is no state fact for role $\mathcal{B}$ as the intruder is not modeled by a state fact.) For the agents in role $\mathcal{B}$ in the Yahalom example, it is not pre-determined who will talk to them, they rather take as the sender name anything that is sent in the first message. Finally, observe that there is no state fact for the key-server, for reasons to made clear below. Also the iknows facts will be discussed later.

Let us now consider the first transition rule of role $\mathcal{A}$. Whenever a state contains a state fact that matches the left-hand side, $\mathcal{A}$ can start a new run of the protocol by creating a fresh nonce NA, updating its state (storing the nonce and increase the step number) and send out the first message of the protocol. The msg fact represents that the message is currently on the communication medium. Note that the application of the rule leads to a replacement of the matched state fact: the old state fact is removed from the state and the updated one is inserted.

The second rule of $\mathcal{A}$ can only be executed when some agent in role $\mathcal{A}$ is indeed in the required local state, namely with step number 1, some agent names A and B, and some nonce NA. Further, there needs to be a message of the correct form on the network—as far as $\mathcal{A}$ can see:

- The message must be a concatenation.
- The first part of the concatenation is a symmetric encryption with the shared key k(A) of

[5] A general definition of the term *session* is difficult, as this notion may depend on the viewpoint: in many attacks, an agent may believe to be communicating with another agent who does not exists at all, or (some) messages of one session are forwarded by the intruder to another session. We therefore use the term *session* in a broad sense to denote the execution of the protocol from the point of view of one or more honest agents. Only the term *bounded number of sessions* is used formally to denote a terminating transition relation (Definition 4.4).

$\mathcal{A}$ and $\mathcal{S}$. Recall that for simplicity, we have assumed here that there is only one server s playing the role $\mathcal{S}$ in all sessions. Therefore the shared keys of an agent A with s is simply k(A). Observe that this term is in fact a function application $\cdot(\cdot)$, where the first argument is the constant k and the second the variable A that must agree the first agent name in the state-fact. In other words, when we have a fixed infrastructure of keys, we do not need to bother about storing all keys in the state facts as well, but can rather have a "global constant" k that is not part of the intruder knowledge.

- The clear-text of the encrypted message must be of the form B, KAB, NA, NB. Here, B and NA must agree with what the state fact contains. KAB and NB, on the other hand, can be any terms: $\mathcal{A}$ *learns* (and stores) these values in this transition.
- The second part of the concatenation, BsPart can also be anything, as $\mathcal{A}$ cannot check that it is a valid message of the protocol as it is supposed to be encrypted with the shared key of $\mathcal{B}$ and $\mathcal{S}$.

Upon reaction, $\mathcal{A}$ will send out the next message of the protocol, simply copying the message BsPart which she cannot decrypt (she does not even store it in her local state, as she never needs it again). Note also that during such a transition the *incoming message*, i.e. the msg fact on the left-hand side of the rule, is removed from the state (i.e. it is now no longer present on the communication medium) and the *outgoing message* is added.

As role $\mathcal{B}$ is very similar, we do not discuss it here, but only observe a point about the rule for role $\mathcal{S}$: there is no state fact for the server. The reason becomes clear from the fact that the outgoing message contains only variables that already occur in the incoming message, except for the key KAB that is freshly created by the server during the transition. In other words, the "output" of the server is a function of the "input" and some fresh data; thus the server does not really need to store anything.[6]

The reader may convince himself that with the rules of Figure 4.3 and the above initial state, there is a reachable state that contains facts of the form $\mathsf{state}_{\mathcal{A}}(2, \mathsf{sess1}, \ldots)$ and $\mathsf{state}_{\mathcal{B}}(2, \mathsf{sess1}, \ldots)$, i.e. one where $\mathcal{A}$ and $\mathcal{B}$ have each finished their part of the protocol. Also, the reader may observe that the given rules are not yet sufficient to reach a state containing a fact of the form $\mathsf{state}_{\mathcal{A}}(2, \mathsf{sess2}, \ldots)$, since there are no rules yet that allow the intruder to send and receive messages. In contrast, $\mathsf{state}_{\mathcal{B}}(2, \mathsf{sess3}, \ldots)$ can be reached, since it differs from the other $\mathcal{B}$-state fact only in the session number (which is not important for the transition rules). In order to detect possible mistakes in protocol specifications, OFMC optionally performs a check whether the "final state" of every honest agent is reachable.

4.3.1 Relationship Between Sessions

In the Yahalom example, the different sessions an agent participates in are in some sense unrelated. More precisely, any transition rule affects a single state fact of the current state and leaves the others untouched; the only potential side-effect on other sessions is through the msg facts. In fact, for such protocols, the communication and the possible interaction of an intruder (that is considered below) is the only reason why distinct sessions cannot be considered independently.

This holds for most protocols, but there are also some important exceptions where we model a relationship between the different sessions an agent participates in. An important example is the contract-signing protocol ASW that we will discuss in detail in Section 17.2. In a nutshell, there is a trusted server that is contacted in case of failures or disputes; this server issues messages for aborting and resolving contracts. In order to fulfill his tasks, the server must maintain a data-base of contracts that have been aborted or resolved, and check for each incoming abort or resolve request, whether the contract in question is already contained in his data-base. Obviously, it would be inappropriate to model the server using independent state facts for each protocol run.

[6] Note that the server rule alone can lead to a non-terminating relation $\Rightarrow_R$, i.e. an unbounded number of sessions. For many approaches it is therefore necessary to restrict this rule to finitely many applications, e.g. by state facts similar to the ones of the other roles.

With set rewriting and concepts like negative facts in rules, it is straightforward to model a data-base maintained by an agent like the one we have just sketched, using the contains facts. To make the illustration easier, we do not describe the ASW protocol here in detail and rather consider an extension of the Yahalom protocol with a similar data-base. Although the example is much simpler, the modeling paradigm is the same.

The extension we consider for Yahalom is a simple replay check: each agent maintains a data-base of session keys KAB that it has already seen in some protocol run; before they accept a new session key, they first check that it is not contained in their data-base.[7]

The replay check could for instance be formulated for role $\mathcal{B}$ of the Yahalom by replacing $\mathcal{B}$'s second rule with the following (where the modification has been underlined):

$$\mathsf{state}_{\mathcal{B}}(1,\mathsf{ID},\mathsf{B},\mathsf{A},\mathsf{NA},\mathsf{NB}).\mathsf{msg}(\{|\mathsf{B},\mathsf{KAB},\mathsf{NA},\mathsf{NB}|\}_{\mathsf{k(B)}}.\{|\mathsf{NB}|\}_{\mathsf{KAB}}).\ \underline{\mathsf{not}(\mathsf{contains}(\mathsf{seenkeys}(\mathsf{B}),\mathsf{KAB}))}$$
$$\Rightarrow \mathsf{state}_{\mathcal{B}}(2,\mathsf{ID},\mathsf{B},\mathsf{A},\mathsf{NA},\mathsf{NB},\mathsf{KAB}).\underline{\mathsf{contains}(\mathsf{seenkeys}(\mathsf{B}),\mathsf{KAB})}$$

In this example, seenkeys is a new constant symbol that is used with the function application operator to model a function from agent names to sets of known keys. The set itself is described by the contains facts as follows: the set s contains the element e in a state S iff $\mathsf{contains}(s,e) \in S$. With this concept, we can describe sets that can change during state transitions, i.e. elements can be added and deleted. The contains fact on the right-hand side of the above rule for instance adds the key KAB just received to the set, if the transition is taken. The fact $\mathsf{not}(\mathsf{contains}(\mathsf{seenkeys}(\mathsf{B}),\mathsf{KAB}))$ on the left-hand side ensures that the transition can only be taken, if the set $\mathsf{seenkeys}(\mathsf{B})$ does not currently contain the key KAB just received.

To model sets (of any kind of messages) that are maintained by honest agents, we have thus exploited that a state itself is a set of facts. This is a powerful concept as we can formulate several operations on sets:

- To check that a set contains a particular element, the respective contains fact is positively stated on the left-hand side, and repeated on the right-hand side.
- To delete an element from a set (e.g. to model a set of TAN numbers that are used only once), the respective contains fact is positively stated on the left-hand side, but not on the right-hand side, so that the matched contains fact gets removed from the state during the transition.
- To modify an element of a set, the element is stated in the original form on the left-hand side, and in the modified form on the right-hand side.
- By combinations of positive and negative contains facts on the left-hand side, we can formulate that elements are contained in the intersection, union, or difference of several sets.

However this model of sets is also limited, as we cannot formulate that some operation is performed on all elements of a set in one transition, such as creating a message for every agent in a set. This would be desirable for group protocols, although such operations can be formulated in a slightly different way by distributing the operation over several transitions, e.g. creating only one message per transition and in this way iterate over the elements of a set.

4.3.2 An Unbounded Number of Sessions and Agents

In the example seen before, the number of possible protocol sessions of honest agents (except for the server rule) and the number of honest agents is limited by the number of state facts that are contained in the initial state. (Such a limitation is often necessary for analysis tools based on exhaustive search to eventually terminate.) In this section, we demonstrate how we can specify a protocol with an unbounded number of agents and sessions in IF.

[7] Although this example may seem contrived for the Yahalom protocol, since the fresh nonces NA and NB should prevent replay of messages, there is an attack to Yahalom [118] that can be prevented by this replay check.

For a finite set A of agents, one can formulate an unbounded number of sessions by a finite set of *initialization rules* that create new state facts without limit, rather than specifying a finite set of them in the initial state. For instance, for the Yahalom protocol, we may have the following set of initialization rules:

$$\begin{array}{lll} \text{Role } \mathcal{A}: & =[\mathsf{ID}]\Rightarrow \mathsf{state}_{\mathcal{A}}(0,\mathsf{ID},a,b) & \text{for each } a,b\in A, a\neq \mathsf{i} \\ \text{Role } \mathcal{B}: & =[\mathsf{ID}]\Rightarrow \mathsf{state}_{\mathcal{B}}(0,\mathsf{ID},b) & \text{for each } b\in A, b\neq \mathsf{i} \end{array}$$

As a fresh ID is created, this can produce an arbitrary number of new state facts between any of the agents and thus induce an unbounded number of protocol sessions. Observe that the initial state facts do not contain any data except for the step number, the fresh identifier and the agent names. In contrast, several other protocol formalisms such as the new HLPSL [48] require the explicit declaration of the long-term keys for the initial state, rather than having them as a function of the agent names; it is then only possible to formulate a fixed number of parallel sessions.

Unfortunately, we cannot have the agent names as the fresh variables in the above initialization theory, and therefore need to require a finite set of agents. To overcome this restriction, we can introduce a new fact symbol $\mathsf{agent}(\cdot)$ for creating agent names by initialization rules itself:

$$\begin{array}{ll} & =[\mathsf{A}]\Rightarrow \mathsf{agent}(A) \\ \text{Role } \mathcal{A}: & \mathsf{agent}(\mathsf{A}).\mathsf{agent}(\mathsf{B}) \wedge \mathsf{A}\neq \mathsf{i} \\ & \quad =[\mathsf{ID}]\Rightarrow \mathsf{state}_{\mathcal{A}}(0,\mathsf{ID},\mathsf{A},\mathsf{B}) \\ \text{Role } \mathcal{B}: & \mathsf{agent}(\mathsf{B}) \wedge \mathsf{B}\neq \mathsf{i} \\ & \quad =[\mathsf{ID}]\Rightarrow \mathsf{state}_{\mathcal{B}}(0,\mathsf{ID},\mathsf{B}) \end{array}$$

Here, we assume that the initial state contains the fact $\mathsf{agent}(\mathsf{i})$. Any number of new agent names can be created on the fly and for each pair of agent names, an unbounded number of sessions can be introduced. As mentioned earlier, we do not create state facts for the intruder, i.e. his behavior is not limited by transition rules as this is the case for the honest agents, but rather only by the Dolev-Yao model; therefore we exclude in the creation of state facts for $\mathcal{A}$ that A is the intruder (but B can be the intruder), and we have a similar exclusion for $\mathcal{B}$.

Two remarks on initialization rules are in order. First, observe that one can split rule applications into two phases without restriction: in the first phase (which is a kind of initialization), we create a finite scenario consisting of an arbitrary number of agents and sessions, and in the second phase (the normal search), we apply the transition rules of honest agents (and intruder as introduced below). Second, these rules are not directly feasible for automated analysis. We will describe a symbolic method in Section 8.4 to handle an unbounded number of agents, and methods for handling an unbounded number of sessions in Part IV.

4.4 Communication Model

So far, we have considered how honest agents behave, in particular how they send messages to, and receive messages from, the communication medium. We now come to the question how this communication medium is modeled and in particular, what the intruder can do on this communication medium. We focus here on the standard model where the intruder entirely controls the network, and then discuss what implications this has for integrating the intruder model (that we have so far only defined in terms of the ability to compose and decompose messages).

The standard communication model is that the intruder controls the communication medium entirely. We can think of a small network where honest agents are connected to a *hub* (i.e. there are no direct links between honest agents) and where the intruder controls this hub. Then he is able to read (and store) any message that is sent by an honest agent on this network; further he is able to remove the message from the network so that it does not reach its intended destination; and finally, he can send to an any honest agent any message that he can decompose and compose according to the $\mathcal{DY}$ model.

If we think of the Internet as the communication medium, then it is of course very unlikely that there is such a powerful intruder: he would have to control every single hub and router of the entire Internet. However, the model of the intruder controlling the entire communication medium makes sense as a worst-case assumption since it subsumes everything that is possible when the intruder controls any subset of the Internet. If there are no attacks in the worst-case, then there are none in case of a less powerful intruder; if there is an attack, then we have a statement of the form "if the intruder controls at least a particular subset of the network, then he can attack the protocol". Similar to the abstractions that we already have when we model messages as terms (e.g. no length information), we abstract with such a model from a lot of technical details, like the concrete topology of the network and the concrete subset that the intruder controls. Such a model may however be too abstract to reasonably analyze certain kinds of protocols (such as secure routing where the topology of the network matters) or particular goals like a low vulnerability to Denial-of-Service attacks (as the intruder can easily block the entire communication in our model).

We model the three intruder actions read, intercept, and send by the two following (protocol independent) IF rules:

$$\mathsf{msg}(\mathsf{M}) \Rightarrow \mathsf{iknows}(\mathsf{M}) \tag{4.1}$$

$$\mathsf{iknows}(\mathsf{M}) \Rightarrow \mathsf{msg}(\mathsf{M}) \tag{4.2}$$

The first rule removes the respective message from the network, so it models a combination of read and intercept. Since the intruder can put any message back onto the network with the second rule, he can easily simulate a pure read (without intercept) in two transitions. As the intruder knowledge is monotonically growing and we have forbidden negative iknows facts in IF rules and attack rules, it is never a restriction that the intruder adds intercepted messages to his knowledge. Thus it is not necessary to simulate a pure intercept (without reading).

It is possible to formulate in IF other models of the communication medium. For instance, one may model a medium on which the intruder cannot read or on which he cannot write. It is however difficult to prevent the intruder from "intercepting" messages for the following reason. Suppose we simply drop the rule (4.1). Then there is still no guarantee that messages will arrive at the intended destination, since nothing in the IF model enforces that an honest agent must ever receive a message sent on the communication medium. In Part IV, we analyze the formal relationship of the IF model with one that does not have an intercept rule and that models interception implicitly by the fact that messages may be ignored. To model an interception-free communication medium, one should thus use a synchronous communication model, where sending and receiving of a message always happens in one transition. A subtlety is then that a sender may be blocked if the receiver currently cannot receive the message. A weaker assumption is that an intruder cannot block the communication medium forever, i.e. messages will eventually reach their destination. We consider such an example in Section 17.2.

4.4.1 Agent Names

An important aspect that we have not discussed so far concerns information like sender and receiver address fields in messages. In the model we have exposed so far, we have used the predicate $\mathsf{msg}(\cdot)$ which does not contain any such information, but only the actual message transmitted. The A&B notation in contrast seems to suggest that such fields are present, since the steps have the form $n.\, A \to B : M$. We argue now that this is a slight misconception.

First observe that in reality there are indeed often such fields, but they contain a technical form of address (such as an IP-address) which are related to the low-layer protocols (such as IP) to route the messages, and which are usually not related to the identity of an agent and to their long-term keys. For instance, the Yahalom protocol can be used by any agent who has a shared-key with the server, even when he is traveling with a notebook and working temporarily under a different IP-address. The identity (and with it the long-term keys) are in most protocols completely unrelated to the low-level IP-addresses.

Looking at the handling of agent names in several protocols such as Yahalom, these names appear in several messages in encrypted parts or in the clear-text, but obviously the protocol

1.	$A \to B:$	$\{NA, A\}_{pk(B)}$	1.	$a \to i:$	$\{na, a\}_{pk(i)}$
2.	$B \to A:$	$\{NA, NB\}_{pk(A)}$	1′.	$i \to b:$	$\{na, a\}_{pk(b)}$
3.	$A \to B:$	$\{NB\}_{pk(B)}$	2′.	$b \to a:$	$\{na, nb\}_{pk(a)}$
			3.	$a \to i:$	$\{nb\}_{pk(i)}$
			3′.	$i \to b:$	$\{nb\}_{pk(b)}$

Figure 4.4: The Needham-Schroeder Public-Key Protocol (without key server) and Lowe's attack.

designers try to minimize the amount of agent names that have to be transmitted. For instance, recall that the message of step 2. of Yahalom contains in clear-text the name (not the IP address) of the person acting in role $\mathcal{B}$, since otherwise the server would not know which key to use for decryption. On the other hand, neither message contains in clear-text the name of the intended receiver, as this information is obviously not needed to decrypt the message if it has been encrypted with the correct key according to the protocol. The reason for including agent names as rarely as possible may have had only efficiency reasons in the case of the Yahalom protocol, but several other protocols such as Station-to-Station [71] and its successors like IKE [78, 86] explicitly state that the reason is to protect the real identities of agents. We therefore suggest not to include the sender's and the receiver's real names automatically in messages, but only as far as the protocol designers have explicitly mentioned them in the description of the protocol messages.

There are however protocols, where agent identities and low-level routing addresses are related. An example is the mentioned IKE which is used as part of IPSec for securing, amongst other things, some low-level information like IP addresses. In these cases the requirement to precisely model the relevant aspects of protocols is even more important, since failure to do so can lead to nonsensical attacks, see [109] for an example.

We close this discussion of agent names with the remark that demanding the real sender name to be included in all messages can rule out attacks: [107] describes an attack to the Needham-Schroeder Public-Key protocol, which we consider in the following section. In this attack an agent generates a fresh nonce that should be kept secret from the intruder. By a type-confusion, the intruder can make another agent accept this message as a different step of the protocol, and interpreting the contained nonce as the sender name. This agent will then ask the key-server for the public-key of the agent with that "name", and thereby transmit the nonce in clear-text, so the intruder can see it. This attack were not possible if the intruder had to first fill in the "name" of the agent who seemingly sent that message, because he does not know the nonce at this point.

4.4.2 The Intruder as a Dishonest Agent

There are a number of protocols that would be secure if all participating agents were honest, i.e. if the intruder does not act under his real name in any of the protocol roles. One example is the Yahalom protocol, for which we have discussed an attack in case that the intruder is participating in role $\mathcal{A}$ (in Section 2.3). Another example is the Needham-Schroeder Public-Key Protocol (NSPK) [113] which is shown in Figure 4.4 together with a man-in-the-middle attack due to Lowe [99]. (Note that for both Yahalom and NSPK, there are no attacks known when all participants are honest.)

In this attack, an honest agent a starts a session with the dishonest agent i (the public key of an agent A is denoted as $pk(A)$). The nonce that a has created is supposed to be kept secret between a and i (since it may later be used to generate a shared secret key). However, i uses this nonce in a second parallel session he starts with the honest agent b, and in which i claims to be a. Agent b now answers with a nonce of his own and sends it to a according to the protocol. i cannot decrypt this message. For a, this message looks just like the reply to her first message to i. So a answers to the i according to the protocol, thereby telling him the nonce nb that was created by b for a and which should thus be secret between a and b. To perform this attack, i does not even have to control the network.

This is the standard example of how a session with a dishonest participant can destroy the

security guarantees of the sessions between honest agents. (Actually, the term session may be confusing in this case, as a is not even aware of b. As a consequence, it has become standard to include dishonest agents into models for protocol verification. Intuition may easily rule out such considerations, and this is probably the reason why the attack against NSPK has not been spotted before 1996, while the protocol was published in 1978 and had already been a common example for formal analysis in the 80s and 90s.

In fact, there is an overwhelming potential for dishonest participants in reality: Firstly, in communication over the Internet one often executes security protocols with participants one does not personally know and who might not be trustworthy at all. For the attack on NSPK, suppose for instance i runs some Internet service offering free art work "via a secure NSPK connection", and a has just stepped into this trap. Secondly, an honest agent's computer may have been hacked by an intruder. The worst case is, of course, that all dishonest agents and intruders work together, so we generally summarize them all under the single identity i.

4.4.3 Step Compression

In the model we have presented, where an intruder controls the communication medium entirely, we can perform an important optimization, namely not to distinguish the communication medium and the intruder. Although one can see this optimization as irrelevant for the modeling, there are several approaches that incorporate this optimization as part of their model [6, 37, 53, 60, 111]; also the HLPSL2IF translator of the AVISPA project produces IF files which incorporate this optimization. We will first informally give the idea behind step compression and then formally show that it is without restriction in the sense that no attacks are excluded (and also, no new attacks are introduced).

The model we have described so far allows for states that contain an arbitrary number of msg facts, i.e. messages that have been sent but not yet received. For instance, the intruder could send out message after message and nobody reacts to them. For this reason, we have introduced in Subsection 4.2.1 some restrictions for the occurrence of msg facts in IF protocol descriptions (see condition 6). The idea is now that the following two restrictions to our state-space are without loss of attacks:

- Whenever an honest agent sends a message, we allow as the next transition only that the intruder intercepts this message with the rule (4.1).
- The intruder can only send messages to the network (using rule (4.2)), if there is a rule of an honest agent containing a positive left-hand side msg fact that matches the message from the intruder. The next transition must be a transition of an honest agent that uses this msg fact.

The restrictions can be expressed by compiling several transition rules together into one rule as follows:

- The first restriction can be expressed by compiling every honest agent rule that has a msg fact on the right-hand side with the rule (4.1). This results in replacing all right-hand side msg facts with corresponding iknows facts.
- The second restriction is the same on the left-hand side, namely compiling the rule (4.2) with every honest agent rule that has a msg fact on the left-hand side. This results in replacing the left-hand side msg fact of this rule with a corresponding iknows fact. In particular this ensures that the message sent by the intruder matches the form of message expected by the receiver.

This compiling of steps makes clear why we call this optimization *step compression*: the state-space gets drastically reduced by compressing several steps together, ruling out the intermediate states and several redundant interleavings. Put another way, the intruder *is* the network now, he absorbs all sent messages, and honest agents just receive messages he wants them to receive.

In the step-compressed model, there is no reason to include the intruder rules (4.1) and (4.2), since there are no msg facts in the initial state and attack states (by the definition of IF) and none in the transition rules for honest agents (by step compression).

Formally, we define step compression as follows:

Definition 4.5. *For a rule r we denote its step compression as $sc(r)$, i.e. the replacement of all* msg *facts of r with* iknows *facts. Similarly, we define $sc(\cdot)$ also for sets of rules and for states.*

For showing the completeness of step compression, we define a relation on states: we say that state S_2 is at least as good for the intruder *as S_1, denoted $S_1 \sqsubseteq S_2$, if $sc(S_1) \subseteq sc(S_2)$ and $sc(S_2) \setminus sc(S_1)$ contains only* iknows *facts.*

As a first step we show that for any reachable state of the standard model, we can reach one that is at least as good for the intruder in the step-compressed model.

Lemma 4.1. *Given an IF protocol description (I, R, A) and let R' be the union of R and the intruder rules (4.1) and (4.2), then for each $S_1 \in reach(I, R')$, there is a state $S_2 \in reach(I, sc(R))$ such that $S_1 \sqsubseteq S_2$.*

Proof. We show this by structural induction according to the definition of *reach* as the closure under the transition relation. The induction basis is trivial as the initial state I is the same in both transition systems, $\sqsubseteq$ is reflexive, and the initial state may not contain msg facts.

For the induction step we have to show the following. Given that $S_1 \in reach(I, R)$ and $S_2 \in reach(I, sc(R))$ with $S_1 \sqsubseteq S_2$, then for every state S_1' with $S_1 \Rightarrow_r S_1'$ (for some $r \in R$) there is a state S_2' such that $S_2 \Rightarrow^*_{sc(R)} S_2'$. We distinguish the three cases of the rule r:

Honest Agent Transitions. Let σ be the substitution under which r is applied to S_1 (including the freshly created values). Then $sc(r)$ of the step-compressed model is also applicable to S_2 under σ for the following reason.

From $S_1 \sqsubseteq S_2$ follows that S_1 and S_2 are identical except for msg facts and iknows facts. Also, for every msg fact and iknows fact in S_1, there is a corresponding iknows fact in S_2. As the syntax of IF rules forbids negative msg facts and iknows facts, and since in $sc(r)$ all msg facts in r are replaced with iknows facts, also $sc(r)$ must be applicable to S_2 under σ.

Also, the resulting state S_2' is identical with S_1' for all facts except msg facts and iknows facts. Moreover, for every new msg fact or iknows fact introduced in the transition from S_1 to S_1', there is a corresponding iknows fact in S_2', thus it holds that $sc(S_1') \subseteq sc(S_2')$, and thus $S_1' \sqsubseteq S_2$.

Rule (4.1). Simply take $S_2' = S_2$: as $S_1 \sqsubseteq S_2$ and S_2 does not contain any msg facts, the intercepted message must already be in the intruder knowledge. Thus we have $S_1 \sqsubseteq S_2$.

Rule (4.2). Again, take $S_2' = S_2$, since when replacing msg facts with iknows facts in S_1 and S_1', then the transition does not change anything. □

From that lemma, we can conclude that the two models are "attack equivalent":

Theorem 4.1. *For every IF protocol description (I, R, A) if holds that (I, R', A) is safe iff $(I, sc(R), A)$ is safe, where R' is the union of R and the intruder rules (4.1) and (4.2).*

Proof. The simulation of the step-compressed model in the original model is easy, as every step in the step-compressed model is simply a compilation of up to three steps of the original model. Therefore it holds that $reach(I, R' \cup \overline{A}) \supseteq reach(I, sc(R) \cup \overline{A})$, and therefore every attack state of the step-compressed model is also contained in the original model (where $\overline{A}$ is the transformation from IF rules to IF attack rules of Definition 4.4).

For the other direction, suppose that $S_1 \in reach(I, R \cup \overline{A})$ is an attack state, i.e. attack $\in S_1$. By Lemma 4.1, there is a state $S_2 \in reach(I, sc(R) \cup \overline{A})$ with $S_1 \sqsubseteq S_2$. By definition thus, attack $\in S_2$, therefore there is also an attack in the step-compressed model.

Therefore attacks in one model can always be recast in the other.

□

As a consequence, we can check every protocol under step compression and without intruder rules and get an attack iff there is an attack in the original model. As a final remark, observe that step compression is one step further on an optimization that is already incorporated in the model, namely that honest agents receive a message and send an reply in one step. This optimization was shown correct by [69].

4.4.4 Typed Protocol Models

We now come to a point about the interaction between intruder and honest agents that is related to both modeling and analysis methods. Observe that the attack on the Yahalom protocol in Section 2.3 is a *type-flaw* attack, i.e. based on the confusion of a pair of nonces NA, NB and a symmetric key KAB. The intruder exploits here the fact that both messages, in reality, are just strings of bits and may be interpreted in different ways, depending on the details of the implementation. Observe that the attack on Yahalom would not possible, if there is a step-number in every encrypted message, which would make the interpretation of each message unique.

Type-flaw attacks found within an abstract formal protocol model can be quite absurd, as they sometimes are only possible in an unreasonable implementation of the protocol, e.g. one that has no rule for determining the start and end of message parts in a concatenation (e.g. by fixed sizes, offsets to fields in the message, special tags, etc.) and rather randomly guesses where the parts start and end. In fact, when we consider the algebraic property that concatenation is associative, then almost every protocol will have some kind of authentication flaw in the formal protocol model, while in most cases this attack is not possible in implementations that have a deterministic way of parsing messages. We thus get false positives since we consider an abstract model that does not reflect all properties of real implementations.

One may thus want to exclude type-flaw attacks from formal protocol models entirely, by reverting to a *typed* model. A standard way to define such a typed model is by associating a unique type (like "agent" or "symmetric key") to all constants and variables and restricting substitutions to well-typed ones. In this way, we can easily exclude that for instance an honest agent ever accepts a pair of nonces as a symmetric key. However, this is often too restrictive, as no variables can be substituted for composed message. For instance, in our model of Yahalom where A cannot decrypt the second part of message 3 we use a variable for this part in the respective transition rule; even in the "normal" execution of the protocol, this variable should be substituted for a composed term.

There are several ways to deal with composed messages in such a model. The easiest is to allow untyped variables that may still be substituted for arbitrary terms. Another way is to use *sorted signatures* where also function symbols of arity greater 0 have a type, e.g. $\{\cdot\}_\cdot : \mathit{PublicKey} \times \mathit{Message} \to \mathit{Message}$. The problem with both these approaches is that they do not type messages as tightly as one may want; the result of most operators has to be of type *Message* that summarizes all terms.

Therefore, one can consider the concept of *composed types* [14, 87]. It is based on a set of atomic types such as *agent*, and the set of composed types is the set of ground terms that can be built from Σ when using only basic types as constants. For instance, if t_1 and t_2 are of types τ_1 and τ_2 already, then $\{t_2\}_{t_1}$ is of type $\{\tau_2\}_{\tau_1}$. This allows us to require a precise structure for every message.

This way of typing can be enforced in IF without extensions to syntax or semantics. For instance, the atomic type *agent* is implemented by introducing a unary, unintrudable function symbol $\mathsf{agent}(\cdot)$, and we write $\mathsf{agent}(t)$ in the IF protocol description for every constant or variable t that has type *agent*. Then, in an IF transition rule, the variables of type *agent* can only be substituted for constants of type *agent*. Composed types can be handled similarly.

The kind of typing we have introduced here may seem quite counter-intuitive, especially for composed types, since agents seem to "magically" recognize the correct types of messages. However this kind of typing can be justified within formal protocol analysis when terms are interpreted in the free algebra [87].

The question of typing, however, is also related to the protocol analysis methods. Observe that, if all variables in a protocol specification are typed and there is a finite number of constants of each type, then the set of messages that can be accepted by an honest agent is finite. By the step compression discussed in Subsection 4.4.3, the intruder can only create messages that an honest agent can accept. Therefore, typing allows one to restrict the set of messages that can ever occur in a protocol description, which is essential for a variety of methods. For instance in Part III, we will develop a method for handling algebraic properties which requires a bound for the depth of terms that can be substituted for variables in transition rules. This is more permissive than typing, but a similar restriction. Also, in Part IV, we discuss methods for handling an unbounded number of sessions; in general, these methods are guaranteed to terminate only in a typed model.

4.5 Protocol Goals

We now turn to the question of formalizing the goals of the protocol. It may not be surprising that we focus on *safety* properties:

Definition 4.6. *A* safety property *over an alphabet Σ is a subset S of Σ^ω such that for every $w \notin S$ there is a finite prefix $w_0 \in \Sigma^*$ of w that is not prefix of any word in S. Dually, a* liveness property *is a subset L of Σ^ω such that every finite word $w \in \Sigma^*$ is a prefix of a word in L.*

Intuitively, a safety property expresses that *never anything bad* happens. Thus, whenever anything bad happens, then the safety property is violated no matter what happens later. A liveness property means that *something good is guaranteed to happen eventually*, and thus no finite trace can prove the violation of a liveness property.

A liveness property would be, for instance, that agents will eventually reach their final states, no matter what the intruder does. But obviously such a goal cannot be achieved in most communication models, as the intruder can block everything or the intended recipients may just not react. Thus, liveness properties usually do not make much sense in security protocols. There are however exceptions as described in Section 17.2: for the ASW protocol, each agent will eventually reach a final state under the assumption that the intruder cannot block the communication forever. We will see how, using meta-reasoning about our model, we can reduce this verification problem to one over safety properties.

Safety properties are relatively easy to handle, as they can be checked on finite traces: if they are violated, there must be a point where things have "gone wrong beyond repair". For a state transition system, the violation of a safety property can be characterized by a subset of the state space. For IF, we have done exactly this, i.e. describing attacks by attack rules, i.e. subsets of the state space for which the desired property is violated.

We describe how to realize the two standard goals of security protocols, secrecy and authentication (the latter in two variants). See also [40] for an overview of standard goals in security protocols. As in the previous sections, we consider here only examples to show how standard goals can be formulated in IF. The results of the following parts hold for any goals that can be expressed using the concept of IF attack rules.

4.5.1 Secrecy

Secrecy of a message means the guarantee that nobody but a specified set of agents can see the message. This natural language definition is vague in several regards. First, it is unclear what secrecy means if a dishonest agent is part of the set of agents who may know the secret: he is able to tell the secret to everyone he wants. This cannot be prevented and obviously should not be considered as a violation of the secrecy goal of the protocol in question—otherwise every security protocol would trivially count as flawed. As we have merged all dishonest participants and intruders into the single intruder i and everyone else is honest, our view of secrecy is simply that the intruder should be able to see a secret iff he is a member of the set of agents who are allowed to know the secret.

Secrecy:	$\mathsf{secret}(\mathsf{M}, \mathsf{SetID}).\mathsf{iknows}(\mathsf{M}).\mathsf{not}(\mathsf{contains}(\mathsf{SetID}, \mathsf{i}))$
Weak Authentication:	$\mathsf{request}(\mathsf{B}, \mathsf{A}, \mathsf{MsgID}, \mathsf{M}, \mathsf{ID}).\mathsf{not}(\mathsf{witness}(\mathsf{A}, \mathsf{B}, \mathsf{MsgID}, \mathsf{M}, \mathsf{ID})) \wedge \mathsf{A} \neq \mathsf{i}$
Strong Authentication (additionally):	$\mathsf{request}(\mathsf{B}, \mathsf{A}, \mathsf{MsgID}, \mathsf{M}, \mathsf{ID}).\mathsf{request}(\mathsf{B}, \mathsf{A}, \mathsf{MsgID}, \mathsf{M}, \mathsf{ID}') \wedge \mathsf{A} \neq \mathsf{i} \wedge \mathsf{ID} \neq \mathsf{ID}'$

Figure 4.5: The IF attack rules for secrecy and authentication.

Second, it is still not clear from this definition who specifies the set of agents that may know the secret. Recall the NSPK attack from Figure 4.4: if it is only a who specifies the set of agents who may know the two nonces, then there is no secrecy attack (since a thinks she is running the protocol with i). More generally, the problem is, if the protocol does not ensure authentication (deliberately or because of a flaw as in NSPK), then different agents may have different views about whom they share the secret with. Thus we recommend that *each* communication partner who shares the secret declares its secrecy (as soon as he knows the secret) between the people that he believes to share the secret with.

The declaration of secrecy works as follows. For Yahalom, we modify the second rule of both role $\mathcal{A}$ and role $\mathcal{B}$, as well as the rule for the server as follows. We augment the set of fresh variables by another one SetID to have a fresh identifier for a fresh set, and add the following facts to the right-hand side of these rules:

$$\mathsf{secret}(\mathsf{KAB}, \mathsf{SetID}).\mathsf{contains}(\mathsf{SetID}, \mathsf{A}).\mathsf{contains}(\mathsf{SetID}, \mathsf{B}) \quad .$$

Note that the server does not need to be included in the set, as he is already guaranteed to be honest in our model of Yahalom.

The attack rule for detecting secrecy flaws then expresses, independent of the concrete protocol, that it is a violation of secrecy if the intruder knows a message that was declared as a secret between a set of people he does not belong to. This attack rule is displayed in Figure 4.5

4.5.2 Authentication

While one might think that authentication is merely concerned with the identities of the communication partners, this is true only for a very basic form of authentication: the guarantee that the communication partner is alive (and we will not consider this goal in isolation). More elaborate forms of authentication are concerned also with the integrity of certain messages or, in other words, with *agreement* between agents on certain messages. There are several variants of authentication goals, for instance Lowe [101] presents a hierarchy of authentication goals. We will consider here only the two most widely used of them, namely *weak authentication* (*non-injective agreement* in the terminology of [101]) and *strong authentication* (called *injective agreement* in [101]).

The two authentication goals that we consider refer to certain data that shall be protected, while all other data of the protocol have a technical purpose and are not in the center of our interest for the goal. We thus call the *payload* the data that shall be agreed upon (although it may be something "technical" as well, like a key for future communication). Also authentication goals have a direction, namely from an *authenticator* to an *authenticated*, in the sense that the authenticator convinces himself of the identity of the authenticated. Mutual authentication and authentication between more parties can be achieved as a combination of these unilateral authentication goals.

Roughly speaking, weak authentication means the following: when the authenticator has reached the end of the protocol execution, he can be sure that the authenticated he believes to be talking with, agrees with him on their names and the payload data. There are several things to be made precise about this informal definition:

1. It leaves relatively open what the agreement precisely means. Consider for instance the case that an intruder has held back the last message of the protocol for a while, and as the

message reaches the recipient, the sender is already off-line, so the agreement may not be at the same time-point but refer to some point in the past.

2. Any party may be involved in several sessions with different people at the same time. Therefore agreement is to be understood existentially on the side of the authenticated party, namely that the authenticated agent was (at some point in the past) in *some* session where he agreed with the authenticator on their names and payload.

3. If there are several authentication goals checked in parallel, then there may be confusion for which purpose a certain message was authenticated, e.g. in Yahalom, whether the authenticated value was meant as NA, NB, or as KAB.

4. It is unclear what happens in a session with the intruder. We simply exclude the case of the intruder being authenticated.

Similarly to secrecy, we insert goal-related facts into the IF rules in order to express those aspects of the current state that are relevant for the goal. This allows us to abstractly define authentication independent of the concrete protocol.

In general, the authenticated issues a $\mathsf{witness}$ fact as soon as possible in his protocol execution, i.e. as soon as he has knows the name of the authenticator and the payload message.[8] The authenticator issues in his last rule a $\mathsf{request}$ fact, i.e. when he has executed the protocol to the end from his point of view.

The goal shown in Figure 4.5 now states that weak authentication is violated whenever an authenticator has issued a $\mathsf{request}$ fact (representing that he will from now on rely on the authentication) while there is no corresponding $\mathsf{witness}$ fact. The intruder is excluded by demanding that he cannot be the authenticated A; also, since the intruder sends messages without producing $\mathsf{request}$ facts, he can never be the authenticator.

Observe that this attack rule covers a large number of attack scenarios, namely

- the authenticated does not exist,
- the authenticated has a different identity then the authenticator thought,
- the authenticated meant to talk to a different person than the authenticator,
- the authenticated meant the payload for a different purpose,
- the authenticated has a different payload.

Example 4.3. Consider again the Yahalom protocol. Let the payload message be the new key KAB, and $\mathcal{B}$ wants to authenticate $\mathcal{A}$ and $\mathcal{S}$. Let pKAB be a new constant to distinguish the purpose KAB from possible other purposes. To the right-hand side of $\mathcal{A}$'s second rule, we add the fact

$$\mathsf{witness}(\mathsf{A},\mathsf{B},\mathsf{pKAB},\mathsf{KAB}) \ ,$$

and to the right-hand side of the rule of the $\mathcal{S}$ we add the fact

$$\mathsf{witness}(\mathsf{s},\mathsf{B},\mathsf{pKAB},\mathsf{KAB}) \ ,$$

expressing that A and s (as the agents to be later authenticated) want to talk with B using the key KAB for the purpose of pKAB.[9] Similarly, we add the following facts to the right-hand side of $\mathcal{B}$'s second rule:

$$\mathsf{request}(\mathsf{B},\mathsf{A},\mathsf{pKAB},\mathsf{KAB},\mathsf{ID}).\mathsf{request}(\mathsf{B},\mathsf{s},\mathsf{pKAB},\mathsf{KAB},\mathsf{ID}) \ ,$$

[8] This is debatable, e.g. [101] suggests to use the *last* possible transition. As this can lead to nonsensical attacks, we do indeed suggest to use the first possible transition.

[9] Recall that we have for simplicity assumed that there is only one server s.

expressing that B (as the authenticator) is now convinced to be talking with A, using the key KAB for the purpose pKAB. (The last parameter is not relevant for weak authentication and will be discussed below.)

By the attack trace in Figure 2.2, we reach a state that contains the following facts:

$$\mathsf{witness}(\mathsf{s},\mathsf{b},\mathsf{pKAB},\mathsf{kab}).\mathsf{request}(\mathsf{b},\mathsf{i},\mathsf{pKAB},\langle\mathsf{na},\mathsf{nb}\rangle,\mathsf{sess1}).\mathsf{request}(\mathsf{b},\mathsf{s},\mathsf{pKAB},\langle\mathsf{na},\mathsf{nb}\rangle,\mathsf{sess1})\ .$$

Observe that there is no witness fact generated by the intruder who plays $\mathcal{A}$, but there cannot be an attack based on that according to the attack rule of Figure 4.5 due to the inequality $\mathsf{A} \neq i$. The request facts from b contain the value $\langle\mathsf{na},\mathsf{nb}\rangle$ for the payload, as that is what he accepted at the end of the protocol run. Now we can apply the attack rule for weak authentication to this state, since the authenticated s has never issued a corresponding witness fact. Thus we have found an attack against the goal that b authenticates s on KAB. □

We now turn to the strong variant of authentication. The idea is that we additionally consider replay of the authenticated data as a violation of a protocol. For instance in the Yahalom protocol, $\mathcal{B}$ can be made accept the same key KAB again and again, if role $\mathcal{A}$ is again played by the intruder [118]. This would not be a violation of weak authentication from the $\mathcal{S}$ to $\mathcal{A}$ (since $\mathcal{S}$ has used the key KAB once in the past).

The idea to detect replay is straightforward: there shall not be more request facts than corresponding witness facts. One way to realize this is by adding the transition rule

$$\mathsf{request}(\mathsf{B},\mathsf{A},\mathsf{MsgID},\mathsf{M},\mathsf{ID}).\mathsf{witness}(\mathsf{A},\mathsf{B},\mathsf{MsgID},\mathsf{M},\mathsf{ID}) \Rightarrow \mathsf{iknows}(\mathsf{i})$$

which can be used to delete matching pairs of a witness fact and a request fact; if there are more request facts than corresponding witness facts in a state, by successively applying this rule, we can reach a state that contains a request fact and no matching witness facts, and thus the weak authentication attack rule fires.

We have, however, chosen a slightly different way, as adding such a rule is not very efficient. It is not a restriction to assume that the payload message in a strong authentication goal contains at least one subterm that is supposed to be generated freshly during the protocol execution (otherwise checking for replay would not be very meaningful). Then, replay detection can be simplified to checking that the exact same message is not accepted by the authenticator more than once. This is expressed by the additional rule for strong authentication in Figure 4.5; the identifier ID is used to distinguish in the set of facts two identical request facts stemming from different sessions. Note that a violation of strong authentication is defined as a violation of one of the two rules, i.e. the one for weak authentication and the additional rule for strong authentication.

Part II

The Lazy Intruder Framework

Chapter 5

Exploring the Transition System

In the previous part, we have discussed how to model security protocols as infinite-state transition systems and how the security properties can be expressed negatively by a subset of the state-space, the attack states. We now come to techniques to efficiently check the safety or insafety of a protocols. As an introduction, we consider the general problems with infinite state spaces, decidability and semi-decidability. After that, we give an overview of the techniques presented in the following chapters.

There are several factors that can cause the set of reachable states to be infinite:

- The Dolev-Yao intruder, even in the free algebra, can construct and send infinitely many different messages from his knowledge at any point. Using step compression, we exclude that the intruder sends out messages that no honest agent currently can process; however, the intersection between the set of messages that the intruder can construct and the set of messages that can currently be received is in most cases infinite.

- There can be an unbounded number of agents, participating in an unbounded number of sessions.

It is well-known that protocol safety is undecidable for an unbounded number of sessions [75, 88], even when bounding the set of messages that the intruder can create [74]. Though the proofs of undecidability are not written with respect to IF protocol descriptions, the undecidability results also apply in our case, since all considered protocol models can be expressed in IF. As we show in Section 5.1, we can obtain a "semi-decision algorithm" for IF protocol insafety in the free algebra, i.e. an algorithm that does not terminate in general, but that terminates and reports an attack, if there is one.

When considering algebraic properties, however, the situation can get even worse. There are many algebraic theories for which even the ground word problem is undecidable, for instance a theory characterizing combinatory logic [17, 65]. The problem of insafety modulo such a theory with an undecidable ground word problem is not even semi-decidable:

Theorem 5.1. *Consider an algebraic theory E for which the ground word problem is undecidable. Then neither safety nor insafety of IF protocol descriptions modulo E is semi-decidable.*

Proof. It is straightforward to see that in general the ground word problem modulo E is semi-decidable, as one can recursively enumerate the E-equivalence class of one of the terms and syntactically compare each element with the other term. We use the following reductions from the complement of the ground word problem modulo E to the protocol safety and insafety problem modulo E.

For any ground terms s and t, let $\Pi_{(s,t)}$ be a protocol description with initial state $\mathsf{state}_{\mathcal{A}}(s)$, an empty set of transition rules, and the attack rule $\mathsf{state}_{\mathcal{A}}(t)$ for some role name $\mathcal{A}$. Let $\Pi'_{(s,t)}$ be the same protocol description except that the attack rule is $\mathsf{not}(\mathsf{state}_{\mathcal{A}}(t))$.

It holds that $\Pi_{(s,t)}$ is safe in E iff $s \not\approx_E t$. Therefore, if protocol safety modulo E were semi-decidable, then the complement of the ground word problem modulo E were semi-decidable, and thus (together with the fact that the ground word problem is semi-decidable) the ground word problem modulo E were decidable. By contradiction, we conclude that protocol safety modulo E cannot be semi-decidable. Similarly, $\Pi'_{(s,t)}$ is unsafe in E iff $s \not\approx_E t$. With a similar argumentation the protocol insafety modulo E cannot be semi-decidable. Observe that this proof does not even require protocols with an unbounded number of sessions. □

For the rest of Part II, we now consider the free algebra.

5.1 A Semi-decision Algorithm

For the case of the free algebra, we now sketch a simple semi-decision algorithm for protocol insafety. The basis is the following lemma:

Lemma 5.1. *For any reachable state S of an IF protocol description, the set of successors of S (the set of all states t such that $s \Rightarrow_R t$ for IF rules R) is recursively enumerable.*

Proof. Consider a state S and a rule r (the extension to a finite set of rules is straightforward). Recall that every state is a *finite* set of *ground* facts. It is clear that the set of all ground substitutions for the variables of r's left-hand side is enumerable. Then we need to filter out those that do not fulfill the conditions of Definition 4.2. All of these checks are simple equality checks on ground terms (in the free algebra this is trivial), except for the check whether $M\sigma \subseteq \mathcal{DY}(IK)$ holds, where IK are the (ground) iknows facts of S and M are the left-hand side iknows facts of r. (Note that M is also ground.) Observe that $\mathcal{DY}(IK)$ is enumerable, since IK is finite and there can only be finitely many messages that can be derived in one step with any rule of the definition of $\mathcal{DY}(IK)$. As we do not care about efficiency here, we can easily integrate the two enumeration algorithms, the one for σ and the one for $\mathcal{DY}(IK)$, to obtain an enumeration algorithm for all solutions of $M\sigma \in \mathcal{DY}(IK)$.

The rest is simple: for each ground substitution σ under which r can be applied, and for given values for the fresh variables, the successor is uniquely determined. □

Since the successors of a state are enumerable, the set of all reachable states is also enumerable using different search strategies as we will now show.

We can view the transition system as an infinite tree, where the root represents the initial state and the children of a node are the successors of the state that the node represents. This tree has in general infinite branching (because there can be infinitely many successors) and infinite depth (because there is no bound on the number of sessions).[1]

The problem is thus to explore the infinite tree and thereby transform it into an infinite list of states. This is a standard algorithmic problem (see, e.g., [122]). However, depth-first or breadth-first search cannot be applied in general: since the tree can have infinite depth, the depth-first search may run into an infinite sub-tree and therefore never visit other parts of the tree; similarly, since the tree can have infinite branching, breadth-first search may never reach lower plies of the tree. Therefore one uses iterated variants of the search algorithms: one starts the search with bounds on depth and width up to which the tree is explored; when this bounded search terminates (and it is guaranteed to terminate), the search is re-started with increased bounds. In this way, every node will eventually be reached.

We thus can design a search algorithm that will recursively enumerate the set of reachable states. According to Definition 4.4 the protocol (I, R, A) is safe iff any reachable state of $reach(I, R \cup \overline{A})$ contains the attack fact. Filtering the reachable states for containment of the attack fact thus yields an enumeration of the attack states: Once an attack state is found, the algorithm answers with the result *insecure*. This algorithm is guaranteed to terminate with this result iff the protocol has an attack, but does not terminate otherwise.

[1] Note that the infinite branching can be "shifted to the depth" by using for instance single transitions for intruder deduction steps as in Example 4.1.

5.2 Using Lazy Evaluation

An approach that allows for a declarative working with infinite data-structures is the use of lazy evaluation in functional programming languages like Haskell [119].[2] At hand of the concrete example of the NSPK protocol, Basin [22, 23] demonstrated how lazy evaluation can be used to declaratively build a semi-decision algorithm for protocol analysis similar to the one sketched above. This work can be seen as the starting point of the OFMC tool.

The key idea behind the lazy infinite-state approach is to explicitly formalize an infinite tree as an element of a data-type in a lazy programming language. This yields a finite, computable representation of the model that can be used to generate arbitrary prefixes of the tree on-the-fly, i.e. in a demand-driven way.

It separates (both conceptually and structurally) the protocol modeling as a transition system from heuristics and other search reduction algorithms, and from search itself: The transition system generates an infinite tree, and heuristics can be seen as tree transducers that take an infinite tree and return one that is, in some way, smaller or more restricted. The resulting tree is then searched. Although semantics, heuristics, and search are all formulated independently, lazy evaluation serves to co-routine them together in an efficient, demand-driven fashion.

Unfortunately, lazy evaluation is subtle. [22, 23] implement iterative deepening search in Haskell as bounded depth-first search with increasing depth bound (the branching of the tree is always finite in these papers). The advantage of iterative deepening as opposed to breadth-first search is that it uses only a small amount of memory, while being a certain factor slower due to the repetitions. However, due to the lazy evaluation, the definition of [22, 23] leads to the same memory usage as breadth-first search since Haskell "observes" that it shall evaluate lazily the same search tree several times (namely increasingly large prefixes of the tree). Lazy evaluation is however defined to evaluate several copies of the same parameter only once. Therefore the part of the tree explored so far is kept in memory, leading to an even larger memory consumption than breadth-first search (which essentially just needs to keep the current ply in memory).

The fix for this problem is tricky: instead of a single tree, we use a function that takes a depth bound and returns a tree (and the term describing the initial state depends on this bound). This hides from Haskell's evaluation strategy the fact that essentially the same tree is being searched. This work-around somewhat defies the argument of declarativity. However, we demonstrate in Section 8.3 that lazy evaluation can be helpful when implementing the lazy intruder technique described in the next section.

5.3 Feasibility

Even if we bound both the number of sessions and the depth of terms that the intruder can generate, the size of the search tree can be infeasible even for the most simple protocols. There are two main reasons for this:

- There is often a large choice of messages that the intruder can create and that can be received by honest agents; this problem gets even more severe the more messages the intruder has learned (or is given initially).
- A standard problem of concurrent systems results from the number of possible interleavings induced by the parallel processes (which are in our case the different protocol runs).

In this part, we will introduce techniques to address these problems:

- In Chapter 6 we introduce the *lazy intruder technique* that allows for a more efficient representation of the intruder. Note that the lazy intruder technique even works without the bounds on the depth of messages that the intruder can generate, and thus implements the

[2] Note that there is no relation between the lazy intruder and the lazy functional programming languages, except that both are demand-driven ("lazy") techniques.

original Dolev-Yao closure without restriction. We then extend the lazy intruder technique so that also inequalities on terms can be handled. This is necessary to later handle negative facts and conditions in IF transition rules.

- We then show how to integrate the lazy intruder into search algorithms. To that end, we first introduce an abstract data-type for lazy intruder constraint stores in Chapter 7. This ADT hides several details of constraint reduction from other aspects of search. Based on the ADT, we define symbolic transition systems for IF protocol descriptions and show their equivalence with the transition system defined by the IF semantics in Chapter 8. In Section 8.4, we introduce the technique of symbolic sessions. This technique exploits the lazy intruder to lazily search the space of instantiations of protocol roles with agent names.
- In Chapter 9, we describe *constraint differentiation*, a new technique for reducing the number of interleavings. Constraint differentiation is a modification of the lazy intruder technique to incorporate ideas from partial-order reduction, which cannot directly be applied when using the lazy intruder.

Chapter 6

The Lazy Intruder

The *lazy intruder* is an optimization technique that significantly reduces the search tree without excluding any attacks. This technique uses a symbolic representation to avoid explicitly enumerating the possible messages that the Dolev-Yao intruder can generate, by storing and manipulating constraints about what must be generated. The representation is evaluated in a demand-driven way and hence the intruder is called *lazy*.

6.1 Constraints

The Dolev-Yao intruder leads to an enormous branching of the search tree when one naïvely enumerates all messages that the intruder can generate and send. The lazy intruder technique exploits the fact that the actual value of certain parts of a message is often irrelevant for the receiver. Therefore, whenever the receiver will not further analyze the value of a particular message part, we can postpone during the search the decision about which value the intruder actually chooses for that part by replacing it with a variable and recording a constraint on which knowledge the intruder can use to generate the message. We express this information using constraints of the form $\mathit{from}(T; IK)$, which intuitively expresses that T is a set of terms generated by the intruder from his set of known messages IK (for "intruder knowledge").

Definition 6.1. *The* semantics of a constraint $\mathit{from}(T; IK)$ *is the set of ground substitutions σ for the variables in the constraint such that all terms of $T\sigma$ can be deduced from $IK\sigma$ according to the Dolev-Yao model. Formally:*

$$[\![\mathit{from}(T; IK)]\!] = \{\sigma \mid \mathit{ground}(\sigma) \wedge \mathit{ground}(T\sigma \cup IK\sigma) \wedge T\sigma \subseteq \mathcal{DY}(IK\sigma)\}\,.$$

A constraint set *is a finite set of constraints and its semantics is the intersection of the semantics of its elements, i.e., overloading notation, $[\![\{c_1, \ldots, c_n\}]\!] = \sqcap_{i=1}^{n} [\![c_i]\!]$. A constraint set C is* satisfiable *if $[\![C]\!] \neq \emptyset$, and we then write $\mathit{satisfiable}(C)$.*

We say that a constraint $\mathit{from}(T; IK)$ is simple *if $T \subseteq \mathcal{V}$, and we then write $\mathit{simple}(\mathit{from}(T; IK))$. A constraint set C is* simple *if all its constraints are simple, and we then write $\mathit{simple}(C)$.*

Example 6.1. As an example, consider again the trace of the attack on the Yahalom protocol introduced in Section 2.3. While this attack trace is ground, we consider here a *symbolic* trace as it occurs in the context of the lazy intruder. The precise construction of the respective symbolic transition system based on the lazy intruder will be introduced in Chapter 8, but the idea of a symbolic trace is as follows. We consider the interaction of the intruder with honest agents where the exchanged messages have the most general possible form from the point of view of only the honest agents. More precisely, we consider what forms of messages may be sent to them, ignoring for a moment the question whether the intruder can generate the respective messages. Using *from* constraints, we then formalize the requirement that the intruder is able to generate each message he sends from the messages he initially knows and that he received from honest agents so far.

We consider the following symbolic trace:

$$
\begin{array}{lll}
1. & \mathsf{i} \to \mathsf{b}: & \mathsf{A}, \mathsf{NA} \\
2. & \mathsf{b} \to \mathsf{i}: & \{|\mathsf{A}, \mathsf{NA}, \mathsf{nb}|\}_{\mathsf{k(b)}} \\
2'. & \mathsf{i} \to \mathsf{s}: & \{|\mathsf{A}', \mathsf{NA}', \mathsf{NB}'|\}_{\mathsf{k(B')}} \\
3. & \mathsf{s} \to \mathsf{i}: & \{|\mathsf{B}', \mathsf{kab}, \mathsf{NA}', \mathsf{NB}'|\}_{\mathsf{k(A')}}, \{|\mathsf{A}', \mathsf{kab}|\}_{\mathsf{k(B')}} \\
4. & \mathsf{i} \to \mathsf{b}: & \{|\mathsf{A}, \mathsf{KAB}|\}_{\mathsf{k(b)}}, \{|\mathsf{nb}|\}_{\mathsf{KAB}}
\end{array}
$$

Here, the intruder first contacts an agent b in role $\mathcal{B}$, claiming to be some agent A and sending some nonce NA—both a variables, since neither the claimed sender name nor the concrete value for the nonce are determined yet. The agent b answers with the message of step 2, using a nonce nb, which is a constant because this value is not chosen by the intruder. The message also contains the values A and NA chosen by the intruder. It is encrypted with the key $\mathsf{k(b)}$ that b shares with the server. In this attack trace, the message goes to the intruder according to the step compression technique (while the agent b actually wants to send it to the server). Next, the intruder sends some message to the server s, which may be completely different from what b has just sent before. Thus, we have fresh variables $\mathsf{A}', \mathsf{B}', \mathsf{NA}'$, and NB' here. The server answers accordingly, where kab is a constant representing the key generated by the server. Finally, i sends the message to b. Note that in this message, the intruder must again use the values that b expects to find in the respective positions, namely the value A that he received in the first message, and the constant nb he created before. Also the first part must be encrypted with his key $\mathsf{k(b)}$. Only the key-term KAB that b learns in this message is not yet determined and may be up to the choice of the intruder.

Denoting with $IK_0 = \{\, \mathsf{a}, \mathsf{b}, \mathsf{i}, \mathsf{k(i)} \,\}$ the initial knowledge of the intruder and with m_1, m_2, m_2', m_3, and m_4 the exchanged messages, we thus have the following constraints on the variables:

$$
\begin{array}{ll}
\{ & \mathit{from}(\{m_1\}; IK_0) \\
 & \mathit{from}(\{m_2'\}; IK_0 \cup \{m_2\}) \\
 & \mathit{from}(\{m_4\}; IK_0 \cup \{m_2, m_3\}) \quad \}
\end{array}
$$

We will use this constraint set in the following as an example to demonstrate how the algorithms for constraint reduction work. □

6.2 Constraint Reduction

We now introduce a reduction algorithm to determine whether a given constraint set is satisfiable and to obtain a finite representation for the set of substitutions that satisfy the constraint set. The key ideas are the following. A simple constraint set (i.e. one with only variables on the left-hand sides of constraints) can be regarded as a *solved form*: a simple constraint set requires that the intruder is able to generate some arbitrary value from his knowledge (as opposed to a more specific value represented by a non-variable term). It is straightforward that a simple constraint set is always satisfiable:

Lemma 6.1. *A simple set of constraints C is satisfiable.*

Proof. Let σ be a substitution with $dom(\sigma) = vars(C)$ and $x\sigma = \mathsf{i}$ for every $x \in dom(\sigma)$. Since $\mathit{intrudable}(\mathsf{i})$ and thus $\mathsf{i} \in \mathcal{DY}(IK\sigma)$ for every intruder knowledge IK, we have $\sigma \in [\![C]\!]$ and thus C is satisfiable. □

The fact that we consider simple constraints as a solved form is the reason for calling this technique the *lazy intruder*: rather than exploring all (infinitely many) messages that the intruder can generate from a particular knowledge, we perform reductions only until we reach simple constraints. Thus, the algorithm for lazy intruder constraints works in a demand-driven, lazy fashion.

Most reduction steps that we perform on the constraints involve a substitution of variables. Since the substituted variables do no longer occur in the constraint after the substitution, we consider pairs (C, σ) of a constraint set C and a substitution σ to properly represent the set of solutions of the initially given constraint set. Note that the domain of σ and the variables of C are disjoint for all such pairs that we will consider. For simplicity of the notation, we overload the semantics function for constraints to such pairs of a constraint set and a substitution:

Definition 6.2. *For a constraint set C and a substitution σ with $dom(\sigma) \cap vars(C) = \emptyset$, we define the semantics of (C, σ) as*

$$[\![(C, \sigma)]\!] = \{\sigma\sigma' \mid \sigma' \in [\![C]\!]\} \ .$$

The reduction of a pair (C, σ) produces a finite set of such pairs $\{(C_1, \sigma_1), \ldots, (C_n, \sigma_n)\}$, and we will show that this transformation is correct in the sense that $[\![(C, \sigma)]\!] = \bigcup_{i=1}^{n} [\![(C_i, \sigma_i)]\!]$, i.e. the reduction algorithm neither excludes nor introduces solutions with respect to the original constraint set and substitution. Moreover, we show that after finitely many reductions we arrive at a finite set of pairs of a constraint set and substitution where the constraint sets are simple. By Lemma 6.1 and the correctness of the reductions, we thus have that the result of the reduction is the empty set iff the given constraint set is unsatisfiable. Moreover, in case the constraint set is satisfiable, we have a finite representation of its solutions by a set of pairs of a simple constraint and a substitution: for the variables in the domain of the substitution, we have a specific value, and for the variables of the constraint we can insert an arbitrary value that the intruder can generate from the respective knowledge.

Well-formed Constraints When considering decryption of messages in a symbolic intruder knowledge, there is a problem with the demand-driven fashion of the lazy intruder: since a variable can represent any message, there is a potential analysis step whenever a variable is in the intruder knowledge (and the resulting message from this analysis step is a new variable that could again represent a message to be analyzed). As we will formally show below, however, it is possible to exclude such analysis steps entirely, if we restrict ourselves to a certain kind of constraint sets, called well-formed constraint sets:

Definition 6.3. *A constraint set C is* well-formed *if one can index the constraints,*

$$C = \{\mathit{from}(T_1; IK_1), \ldots, \mathit{from}(T_n; IK_n)\} \ ,$$

so that the following conditions hold:

$$IK_i \subseteq IK_j \ \ \textit{for } i \leq j \ , \tag{6.1}$$

$$vars(IK_i) \subseteq \bigcup_{j=1}^{i-1} vars(T_j) \ . \tag{6.2}$$

Intuitively, (6.1) requires that the intruder knowledge increases monotonically, and (6.2) requires that every variable that appears in terms known by the intruder is part of a message that the intruder created earlier. Said in another way, variables only "originate" from the intruder (though they may represent a part of a message that he does not know). As an example, the reader may convince himself that the constraint set in Example 6.1 is well-formed. We will see as part of Chapter 8 that well-formed constraints are sufficient to express all intruder deduction problems that we need to cover in the analysis of IF protocol specifications.

Why not a Calculus? We will describe the lazy intruder technique below by an algorithm which is close to the real implementation in OFMC. In contrast, most other presentations of the lazy intruder technique, including our own [24, 28], are based on a calculus. While such a presentation is standard for many logic-related problems, there is a significant gap between such a calculus and the real implementation in the case of the lazy intruder. More in detail, the calculus admits a

large number of possible sequences of reductions and only a small portion of them actually needs to be considered by the real implementation. (The complete enumeration of all possible reduction sequences can be worse than the naïve enumeration of ground terms.) A completeness proof for a calculus does not tell us anything about the completeness of an implementation that considers only a small portion of all reductions that the calculus admits. We therefore present the lazy intruder here directly as an algorithm and show its correctness (including completeness) and termination. However, we are as abstract as possible in the presentation of the algorithm and deliberately avoid to pin down choices that are irrelevant for completeness (or termination). Also, we describe the algorithm as a function and abstract from minor implementation details such as representing sets by lists.

6.2.1 Reduction without Analysis

As the first part of the reduction algorithm, we will leave out analysis of messages entirely, and focus only on what the intruder can generate from a given knowledge. Recall that the semantics of *from* constraints is defined using the $\mathcal{DY}$ closure and that this is the closure under two kinds of rules, namely generate rules (G_{axiom} and G_{comp}) and analysis rules (A_{pair_i}, A_{scrypt}, A_{crypt}, and A_{sign}). We have also defined the closure $\mathcal{DY}^{gen}$ as the closure under only the generate rules. We will use this as a formal basis to restrict our attention to reductions without analysis steps, namely we define a variant of the semantics of constraints that is based on $\mathcal{DY}^{gen}$:

Definition 6.4. *We define the* generate-only semantics *of from constraints as follows:*

$$[\![\mathit{from}(T; IK)]\!]^{gen} = \{\sigma \mid \mathit{ground}(\sigma) \wedge \mathit{ground}(T\sigma \cup IK\sigma) \wedge T\sigma \subseteq \mathcal{DY}^{gen}(IK\sigma)\}\,.$$

Like in Definition 6.1, we extend this definition to sets of constraints and pairs (C, σ) of a constraint set and a substitution where $dom(\sigma) \cap vars(C) = \emptyset$:

$$\begin{aligned} [\![\{c_1, \ldots, c_n\}]\!]^{gen} &= \sqcap_{i=1}^{n} [\![c_i]\!]^{gen} \\ [\![(C, \sigma)]\!]^{gen} &= \{\sigma\sigma' \mid \sigma' \in [\![C]\!]^{gen}\} \end{aligned}$$

We now define a reduction algorithm for well-formed constraint sets that is correct with respect to the generate-only semantics. We define this algorithm as a function from a pair of a constraint set and a substitution to a finite set of pairs of a simple constraint set and a substitution. We first give a formal definition of the algorithm, then explain it informally and give an example, and finally we formally show its correctness and termination.

Definition 6.5. *Let $Red^{gen}(C, \sigma)$ be the following algorithm:*

$$Red^{gen}(C, \sigma) = \begin{cases} \{(C, \sigma)\} & \text{if } \mathit{simple}(C) \\ \bigcup_{(C', \sigma') \in Red_1^{gen}(C, \sigma)} Red^{gen}(C', \sigma') & \text{otherwise} \end{cases}$$

$$\begin{aligned} Red_1^{gen}(C \cup \{\mathit{from}(\{t\} \cup T; IK)\}, \sigma) = & \bigcup_{(t_1, \ldots, t_n) \in generate(t)} (C \cup \{\mathit{from}(\{t_1, \ldots, t_n\} \cup T; IK)\}, \sigma) \\ & \cup \bigcup_{\tau \in unifyall(t, IK)} ((C \cup \{\mathit{from}(T; IK)\})\tau, \sigma\tau) \\ & \text{where } t \notin \mathcal{V} \end{aligned}$$

$$generate(t) = \begin{cases} \{(t_1, \ldots, t_n)\} & \text{if } t = f(t_1, \ldots, t_n) \text{ and } \mathit{intrudable}(f) \\ \emptyset & \text{otherwise} \end{cases}$$

$$unifyall(t, IK) = \bigcup_{s \in IK} unify(\{(s, t)\})$$

where we assume that the pattern matching on the left-hand side of Red_1^{gen}, i.e. the selection of a term $t \notin \mathcal{V}$ on the left-hand side of some from constraint, is performed by some deterministic selection algorithm which we do not prescribe.

We start our explanation with the algorithm Red_1^{gen}. The assumption of this algorithm is that it is called for a non-simple constraint set, i.e. there is a term $t \notin \mathcal{V}$ on the left-hand side of some constraint (otherwise, Red_1^{gen} is undefined). As we will prove later, correctness and termination of the reduction algorithm do not depend on the choice of this term t, thus we do not prescribe how to choose this term.

The first part of the Red_1^{gen} algorithm, which we also refer to as the *generate part* of Red_1^{gen}, is the symbolic counter-part of the G_{comp} rule. For the term t, the algorithm calls the function *generate* to determine whether the term may have been composed by the intruder from its subterms. This is the case if $t = f(t_1, \ldots, t_n)$ and f is intrudable. Using a set of results for *generate* makes the formulation of Red_1^{gen} simpler: if the term cannot have been composed, then this first union in the body of Red_1^{gen} is simply empty. Thus, if the term t can be composed from $t_1, \ldots, t_n$, we replace t with the set $\{t_1, \ldots, t_n\}$ in the constraint.

The second part of the Red_1^{gen} algorithm, which we also refer to as the *unify part* of Red_1^{gen}, is the symbolic counter-part of the G_{axiom} rule. It considers the possibilities to unify the term t with any term from the respective intruder knowledge IK. To that end, Red_1^{gen} calls the function *unifyall* which returns all unifiers between t and a term $s \in IK$. For each unifier τ, we have the remaining constraint set, without t to generate, under the substitution τ.

Finally, the Red^{gen} algorithm yields a set of pairs of a simple constraint set and a substitution as follows. As long as the given constraint set is not yet simple, it calls the Red_1^{gen} algorithm to eliminate one non-variable term, which yields a set of results, and then Red^{gen} recursively reduces each resulting constraint set.

Example 6.2. We show that the constraint set of Example 6.1 can be reduced with Red^{gen} to yield the attack trace we have described in Section 2.3, if we first perform an analysis step that cannot be performed by Red^{gen}, which in turn requires a substitution. More concretely, we consider the substitution $\sigma = [\mathsf{A}' \mapsto \mathsf{i}, \mathsf{NB}' \mapsto \mathsf{nb}]$, so that the intruder can find out nb and kab (because he can decrypt the first part of $m_3\sigma$). Thus we consider the constraint set:

$$\left\{ \begin{array}{l} \mathit{from}(\{m_1\}; IK_0) \\ \mathit{from}(\{m_2'\sigma\}; IK_0 \cup \{m_2\}\sigma) \\ \mathit{from}(\{m_4\sigma\}; IK_0 \cup \{m_2, m_3, \mathsf{nb}, \mathsf{kab}\}\sigma) \end{array} \right\} .$$

For simplicity, we omit the generating steps for pairs, e.g. we assume that the message $m_1\sigma = \langle \mathsf{A}, \mathsf{NA} \rangle$ is immediately decomposed into the set of terms $\{\mathsf{A}, \mathsf{NA}\}$, so that the first constraint is already simple.

With Red_1^{gen}, we first choose to reduce the message $m_2'\sigma = \{\!|\mathsf{i}, \mathsf{NA}', \mathsf{nb}|\!\}_{\mathsf{k}(\mathsf{B}')}$. There are two possible reductions, namely with the generate part (i.e. the intruder has to construct the terms $\mathsf{k}(\mathsf{B}'), \mathsf{i}, \mathsf{NA}'$ and nb) or unification with the message $m_2\sigma$. Let us follow the latter case, which results in the unifier $\sigma_1 = [\mathsf{A} \mapsto \mathsf{i}, \mathsf{NA}' \mapsto \mathsf{A}, \mathsf{B}' \mapsto \mathsf{b}]$. The constraint set now looks as follows (omitting the second constraint that has now an empty left-hand side):

$$\left\{ \begin{array}{l} \mathit{from}(\{\mathsf{i}, \mathsf{NA}\}; IK_0), \\ \mathit{from}(\{\{\!|\mathsf{i}, \mathsf{KAB}|\!\}_{\mathsf{k}(\mathsf{b})}, \{\!|\mathsf{nb}|\!\}_{\mathsf{KAB}}\}; IK_1) \end{array} \right\} .$$

where $IK_1 = IK_0 \cup \{\{\!|\mathsf{i}, \mathsf{NA}, \mathsf{nb}|\!\}_{\mathsf{k}(\mathsf{b})}, \{\!|\mathsf{b}, \mathsf{kab}, \mathsf{NA}, \mathsf{nb}|\!\}_{\mathsf{k}(\mathsf{i})}, \{\!|\mathsf{i}, \mathsf{kab}|\!\}_{\mathsf{k}(\mathsf{b})}, \mathsf{nb}, \mathsf{kab}\}$.

The name of the intruder i on the left-hand side of the first constraint can easily be eliminated using a generate reduction (as i is an intrudable function symbol). For the next reduction we choose the term $\{\!|\mathsf{i}, \mathsf{KAB}|\!\}_{\mathsf{k}(\mathsf{b})}$. Again there are two possible reductions: that the intruder generates the term from its components or that we unify it with a term in the intruder knowledge. There are two candidates for unification in the intruder knowledge, namely the two messages encrypted with

$\mathsf{k(b)}$. Let us consider the first one, $\{|\mathsf{i}, \mathsf{NA}, \mathsf{nb}|\}_{\mathsf{k(b)}}$, where the unifier is $\sigma_2 = [\mathsf{KAB} \mapsto \langle \mathsf{NA}, \mathsf{nb} \rangle]$. The resulting constraint set is:

$$\{ \quad \mathit{from}(\{\mathsf{NA}\}; IK_0) \\ \quad \mathit{from}(\{\{|\mathsf{nb}|\}_{\langle \mathsf{NA},\mathsf{nb} \rangle}\}; IK_1) \quad \} \ .$$

The remaining non-simple term can only be addressed using a generate reduction (as there is no term that can be unified in the intruder knowledge). Thus we have the constraint set:

$$\{ \quad \mathit{from}(\{\mathsf{NA}\}; IK_0) \\ \quad \mathit{from}(\{\mathsf{NA}, \mathsf{nb}\}; IK_1) \quad \} \ .$$

Finally, the term nb is eliminated by a unification step, since $\mathsf{nb} \in IK_1$, resulting in a simple constraint set. Considering the symbolic attack trace we started from in Example 6.1, we obtain the following instance of this trace from the substitutions performed during constraint reduction:

1. $\mathsf{i} \to \mathsf{b}$: i, NA
2. $\mathsf{b} \to \mathsf{i}$: $\{|\mathsf{i}, \mathsf{NA}, \mathsf{nb}|\}_{\mathsf{k(b)}}$

2′. $\mathsf{i} \to \mathsf{s}$: $\{|\mathsf{i}, \mathsf{NA}, \mathsf{nb}|\}_{\mathsf{k(b)}}$
3. $\mathsf{s} \to \mathsf{i}$: $\{|\mathsf{b}, \mathsf{kab}, \mathsf{NA}, \mathsf{nb}|\}_{\mathsf{k(i)}}, \{|\mathsf{i}, \mathsf{kab}|\}_{\mathsf{k(b)}}$

4. $\mathsf{i} \to \mathsf{b}$: $\{|\mathsf{i}, \mathsf{NA}, \mathsf{nb}|\}_{\mathsf{k(b)}}, \{|\mathsf{nb}|\}_{\mathsf{NA},\mathsf{nb}}$

Note that this attack trace is still symbolic, as we have the variable NA in it, indicating that the value the intruder sends for his own nonce does not matter for the attack and we can still represent all possible instances of the value (that the intruder can generate from his initial knowledge) in this symbolic attack trace. □

We now show that the reduction algorithm terminates and is correct with respect to the generate-only semantics:

Theorem 6.1. *Red^{gen} terminates for all inputs (thus returning a finite set of results) and is correct with respect to the generate-only semantics in the following sense: If C is a well-formed constraint set and σ a substitution with $dom(\sigma) \cap vars(C) = \emptyset$, then*

$$[\![(C, \sigma)]\!]^{gen} = \bigcup_{(C',\sigma') \in Red^{gen}(C,\sigma)} [\![(C', \sigma')]\!]^{gen} \ ,$$

and for every $(C', \sigma') \in Red^{gen}(C, \sigma)$, it holds that C is a well-formed simple constraint set, $dom(\sigma') \cap vars(C') = \emptyset$, and $dom(\sigma') \cup vars(C') = dom(\sigma) \cup vars(C)$.

Proof. We split the proof into four parts: invariants, termination, soundness, and completeness.

Invariants We first show that several properties are preserved throughout Red^{gen}. When Red^{gen} and Red_1^{gen} are called with a pair (C, σ) where $vars(C) \cap dom(\sigma) = \emptyset$ and C is well-formed, then the following properties hold for all pairs (C', σ') that are constructed during Red^{gen} (i.e. those used as the argument in a recursive call of Red^{gen}, in a call of Red_1^{gen}, and that are returned as a result):

1. $dom(\sigma') \cap vars(C') = \emptyset$,
2. $dom(\sigma) \cup vars(C) = dom(\sigma') \cup vars(C')$, and
3. C' is well-formed.

First recall that for $\tau \in \mathit{unify}\{(s,t)\}$ we have $\mathit{dom}(\tau) \cup \mathit{vars}(\mathit{ran}(\tau)) \subseteq \mathit{vars}(s) \cup \mathit{vars}(t)$ and $\mathit{dom}(\tau) \cap \mathit{vars}(\mathit{ran}(\tau)) = \emptyset$. The same thus also hold for all substitutions resulting from $\mathit{unifyall}(t, IK)$.

For the conditions (1) and (2) on the variables of substitutions and constraints, consider first the cases in the unification part of Red_1^{gen}. For the resulting constraint set $C' = C \cup \{\mathit{from}(T; IK)\}\tau$ and the resulting substitution $\sigma' = \sigma\tau$ we have the following properties. Since $\mathit{vars}(t\tau) = \mathit{vars}(s\tau)$ for some term $s \in IK$, it follows that $\mathit{vars}(t) \subseteq \mathit{dom}(\tau) \cup \mathit{vars}(IK\tau)$. Thus $\mathit{vars}(C') = \mathit{vars}(C) \setminus \mathit{dom}(\tau)$, and therefore $\mathit{vars}(C') \cap \mathit{dom}(\sigma') = \emptyset$. Moreover

$$\mathit{vars}(C') \cup \mathit{dom}(\sigma') = (\mathit{vars}(C) \setminus \mathit{dom}(\tau)) \cup \mathit{dom}(\tau) \cup \mathit{dom}(\sigma) = \mathit{vars}(C) \cup \mathit{dom}(\sigma) \ .$$

For the cases in the generate part of Red_1^{gen}, we have that $\mathit{vars}(f(t_1, \ldots, t_n)) = \mathit{vars}(t_1) \cup \ldots \cup \mathit{vars}(t_n)$. Therefore, the set of variables in the constraints does not change. It follows that the conditions (1) and (2) are met over all arguments and results of Red_1^{gen} and Red^{gen}.

For the well-formedness, we first show that $C\sigma$ is well-formed for every well-formed constraint C and every substitution σ. To see that, observe that $IK_i \subseteq IK_j$ implies $IK_i\sigma \subseteq IK_j\sigma$ for every intruder knowledges IK_i and IK_j. Thus the condition (6.1) of well-formedness remains preserved under substitution. Moreover, if $\mathit{vars}(IK_i) \subseteq \bigcup_{j=1}^{i-1} \mathit{vars}(T_j)$ holds for an intruder knowledge IK_i and left-hand sides $T_1, \ldots, T_{i-1}$, then we have

$$\mathit{vars}(IK_i\sigma) = \mathit{vars}(IK_i) \setminus \mathit{dom}(\sigma) \subseteq \left(\bigcup_{j=1}^{i-1} \mathit{vars}(T_j)\right) \setminus \mathit{dom}(\sigma) = \bigcup_{j=1}^{i-1} \mathit{vars}(T_j\sigma) \ .$$

Thus condition (6.2) of well-formed constraint sets remains preserved under substitution.

Next, we show that all (C', σ') that result out of a unify case in Red_1^{gen} are well-formed. The substitution of the original constraint C under the unifier τ is still well-formed. It remains to show that this also holds when removing the unified left-hand side term t. Let $T_1, \ldots, T_{i-1}$ be the left-hand sides of the preceding constraints of C in the order induced by the well-formedness. Because τ is a unifier for t and some term $s \in IK$, we have that $\mathit{vars}(t\tau) \subseteq \mathit{vars}(IK\tau)$, and by the well-formedness of $C\tau$, we have $\mathit{vars}(IK\tau) \subseteq \bigcup_{j=1}^{i-1} \mathit{vars}(T_j)$. Thus all variables of $t\tau$ are already introduced in an earlier constraint, i.e. the second condition of the well-formedness is not destroyed by removing $t\tau$ from the left-hand side. For the generate part, we have that the set of variables on the left-hand side never changes and neither does the intruder knowledge. Thus also invariant (3) is preserved throughout Red_1^{gen} and Red^{gen}.

We conclude the part on the invariants with the remark that every constraint that is returned by Red^{gen} is simple.

Termination It is straightforward to see that *unifyall*, *generate* terminate. Therefore also Red_1^{gen} terminates. It remains to show that Red^{gen} cannot run into an infinite loop. To that end, we define a weight function w for terms, constraints, and constraint sets. Then we show that all results of $\mathit{Red}_1^{gen}(C, \sigma)$ have a strictly smaller weight than C according to a well-founded ordering on the weight. Thus there cannot be an infinite chain of constraint sets with decreasing weight. For a term t we define

$$w(t) = \begin{cases} 1 + w(t_1) + \ldots + w(t_n) & \text{if } t = f(t_1, \ldots, t_n) \\ 1 & \text{otherwise .} \end{cases}$$

The weight of a *from* constraint is the sum of the weight of its left-hand side terms:

$$w(\mathit{from}(\{t_1, \ldots, t_n\}; IK)) = w(t_1) + \ldots + w(t_n)$$

The weight of a set of constraints is defined as a pair of natural numbers, namely the number of variables that occur in the constraints and the sum of the weights of the constraints:

$$w(\{c_1, \ldots, c_n\}) = (\#\mathit{vars}(\{c_1, \ldots, c_n\}), w(c_1) + \ldots, w(c_n))$$

We order these pairs lexicographically, overloading the symbol ">", defining

$$(v_1, g_1) > (v_2, g_2) \text{ iff } v_1 > v_2 \ \vee \ (v_1 = v_2 \ \wedge \ g_1 > g_2)\,.$$

This order is well-founded, as the ordering on both components is well-founded over the natural numbers.

It thus remains to show that $w(C) > w(C')$ for each $(C', \sigma') \in \mathit{Red}_1^{gen}(C, \sigma)$:

- If (C', σ') results from the *generate* part of Red_1^{gen}, then we have $w(\mathit{from}(\{t\} \cup T; IK)) = 1 + w(\mathit{from}(\{t_1, \ldots, t_n\} \cup T; IK))$, since $t = f(t_1, \ldots, t_n)$. As the rest of the constraint set is not touched and C and C' contain the same set of variables, $w(C) > w(C')$.
- If (C', σ') results from the unify part of Red_1^{gen}, we distinguish two cases: First, if $\tau = \mathit{id}$ then we have $w(\mathit{from}(\{t\} \cup T; IK)) = w(t) + w(\mathit{from}(T; IK))$ and since $w(t) > 0$ and nothing else is changed, $w(C) > w(C')$. Second, if $\tau \neq \mathit{id}$, then $\mathit{vars}(C')$ is a proper subset of $\mathit{vars}(C)$, since $\mathit{dom}(\sigma\tau) \cup \mathit{vars}(C') = \mathit{dom}(\sigma) \cup \mathit{vars}(C)$ and $\mathit{vars}(C') = \mathit{vars}(C) \setminus \mathit{dom}(\tau)$ as shown in the invariants part of this proof. So, by the first component of the weight, $w(C) > w(C')$.

Soundness We show

$$[\![(C, \sigma)]\!]^{gen} \supseteq \bigcup_{(C', \sigma') \in \mathit{Red}^{gen}(C, \mathit{id})} [\![(C', \sigma')]\!]^{gen}\,.$$

To that end, it is sufficient to show the soundness of Red_1^{gen}, i.e. that $(C', \sigma') \in \mathit{Red}_1^{gen}(C, \sigma)$ and $\mathit{dom}(\sigma) \cup \mathit{vars}(C) = \emptyset$ implies:

$$[\![(C, \sigma)]\!]^{gen} \supseteq [\![(C', \sigma')]\!]^{gen}\,.$$

Assume that $[\![(C', \sigma')]\!]^{gen} \neq \emptyset$ (otherwise the statement is trivial), and let $\sigma_0 \in [\![(C', \sigma')]\!]^{gen}$, it is to show that $\sigma_0 \in [\![(C, \sigma)]\!]^{gen}$. We distinguish two cases.

1. (C', σ') was induced by the generate part. There are a constraint set C_0, a term t, and sets of terms T and IK such that
   - $t = f(t_1, \ldots, t_n)$ for some terms $t_1, \ldots, t_n$ and an intrudable function f,
   - $C = C_0 \cup \{\mathit{from}(\{t\} \cup T; IK)\}$,
   - $C' = C_0 \cup \{\mathit{from}(\{t_1, \ldots, t_n\} \cup T; IK)\}$, and
   - $\sigma = \sigma'$.

   By the semantics, it holds that $\sigma_0' \in [\![C_0]\!]^{gen}$ where σ_0' is the domain-restriction of σ_0 to $\mathit{vars}(C_0)$. Further it holds that $(\{t_1, \ldots, t_n\} \cup T)\sigma_0 \subseteq \mathcal{DY}(IK\sigma_0)$. Since f is intrudable, we have that $f(t_1, \ldots, t_n)\sigma_0 \in \mathcal{DY}(IK\sigma_0)$. Therefore $\sigma_0 \in [\![(C, \sigma)]\!]^{gen}$.

2. (C', σ') was induced by the unify part. There are a constraint set C_0, a term t, sets of terms T and IK, a term $s \in IK$, and a substitution $\tau \in \mathit{unify}(\{(s, t)\})$ such that
   - $C = C_0 \cup \{\mathit{from}(\{t\} \cup T; IK)\}$,
   - $C' = C_0 \cup \{\mathit{from}(T; IK)\}$, and
   - $\sigma\tau = \sigma'$.

   By the semantics, it holds that $\sigma_0' \in [\![C_0 \cup \{\mathit{from}(T; IK)\}]\!]^{gen}$ where σ_0' is the domain-restriction of σ_0 to $\mathit{vars}(C')$. It holds that $\sigma_0 \succeq \sigma'$, thus $t\sigma_0 = s\sigma_0$ and therefore $\sigma_0 \in [\![C_0 \cup \{\mathit{from}(\{t\} \cup T; IK)\}]\!]$. Since $\sigma_0 \succeq \sigma$, it follows that $\sigma_0 \in [\![(C, \sigma)]\!]^{gen}$.

This concludes the soundness proof for Red^{gen}.

Completeness We show

$$[\![(C,\sigma)]\!]^{gen} \subseteq \bigcup_{(C',\sigma')\in Red^{gen}(C,id)} [\![(C',\sigma')]\!]^{gen} .$$

To that end, we consider an arbitrary fixed solution $\sigma_0 \in [\![(C,\sigma)]\!]^{gen}$ and show that there is a $(C',\sigma') \in Red^{gen}(C,\sigma)$ such that $\sigma_0 \in [\![(C',\sigma')]\!]^{gen}$. For showing this, we use the concept of Dolev-Yao proofs introduced in Definition 3.7. Since for every constraint $from(T;IK) \in C$, there is a proof that $T\sigma_0 \subseteq \mathcal{DY}^{gen}(IK\sigma_0)$, and thus there exists a labeling of each term $t \in T$ of such a constraint with a Dolev-Yao proof for $t\sigma_0 \in \mathcal{DY}^{gen}(IK\sigma_0)$, i.e. a tree with $t\sigma_0$ at the root and terms of $IK\sigma_0$ at the leaves, and every node is labeled with only generate rules.

We use this labeling as a proof concept (not as a technique to solve constraints which would require first finding all substitutions). More precisely, we will show that there is a $(C',\sigma') \in Red_1^{gen}(C,\sigma)$ such that all left-hand side terms in C' can also be labeled with a Dolev-Yao proof for the solution σ_0, and that $\sigma_0 \succeq \sigma'$. In this way we use the Dolev-Yao proofs as an invariant that guarantees that the solution σ_0 is still supported by some constraint set resulting from Red_1^{gen}. As we have not assumed anything about (C,σ), we can show by induction that this holds for arbitrarily many reduction steps.

Consider now an arbitrary term t of a left-hand side of C that is selected by the Red_1^{gen} algorithm, i.e. $C = C_0 \cup \{from(\{t\} \cup T; IK)\}$ for some constraint set C_0 and sets of terms T and IK, and such that $t \notin \mathcal{V}$. Let dt be the Dolev-Yao proof attached to t (proving that $t\sigma_0 \in \mathcal{DY}(IK\sigma_0)$). We perform a case split over the kind of root node of dt:

- If dt is a leaf, i.e. an axiom, then there is a term $s_0 \in IK\sigma_0$ such that $t\sigma_0 \approx s_0$. Thus there is a term $s \in IK$ such that $\tau \in unify\{(s,t)\}$ and $\sigma_0 \succeq \tau$. We can thus see that one of the solutions of the unify part of Red_1^{gen} still supports the substitution σ_0 as follows. It holds that $\tau \in unifyall(t, IK)$, and therefore one of the results of $Red_1^{gen}(C,\sigma)$ is $C' = (C_0 \cup \{from(T; IK)\})\tau$ and $\sigma' = \sigma\tau$. As $\sigma_0 \succeq \sigma$ and $\sigma_0 \succeq \tau$, we have $\sigma_0 \succeq \sigma\tau = \sigma'$. Further, there exists a labeling for all left-hand side terms of C' under solution σ_0, namely the same labeling as for the respective terms in C, since $\sigma_0 \succeq \tau$.

- If dt is an inner node, then $t\sigma_0 = f(t'_1, \ldots, t'_n)$ for some ground terms $t'_1, \ldots, t'_n$. As $t \notin \mathcal{V}$ it holds that $t = f(t_1, \ldots, t_n)$ for some terms $t_1, \ldots, t_n$ such that $t_i\sigma_0 = t'_i$ for each $i \in \{1, \ldots, n\}$. Therefore $(t_1, \ldots, t_n) \in generate(t)$ and thus one of the results of $Red_1^{gen}(C,\sigma)$ is $C' = C_0 \cup \{from(\{t_1, \ldots, t_n\}; IK)\}$ and $\sigma' = \sigma$. It holds that $\sigma_0 \in [\![(C',\sigma')]\!]^{gen}$ since $\sigma' = \sigma$ and there exists a labeling of the left-hand sides of C' for the solution σ' as follows. Every term that was already present in C has the same labeling as in C. For the terms $t_1, \ldots, t_n$, we pick the sub-proofs of the Dolev-Yao proof dt: since the root of dt is an inner node, its children are Dolev-Yao proofs for the terms $t'_1, \ldots, t'_n$. We thus label the terms $t_1, \ldots, t_n$ with the Dolev-Yao proofs of $t'_1, \ldots, t'_n$, respectively, and this labeling is correct as $t_i\sigma_0 = t'_i$.

We have thus shown that for the arbitrary solution $\sigma_0 \in [\![(C,\sigma)]\!]^{gen}$ there is a $(C',\sigma') \in Red_1^{gen}(C,\sigma)$ such that $\sigma_0 \in [\![(C',\sigma')]\!]^{gen}$. Therefore an arbitrary number of reduction steps still supports σ_0 and thus Red^{gen} is complete with respect to the generate-semantics, which concludes the proof. □

We conclude the discussion of Red^{gen} by looking at some improvements of the algorithm. Observe that we have a case split for each non-variable term into decomposition and unification with each term in the respective intruder knowledge. This case split can result in a large number of cases, if the intruder already knows many terms that can be unified with the term to generate. We thus look for ways to reduce the number of cases without loosing solutions. To that end, consider the reduction with Red_1^{gen} of the term t in the constraint set $C \cup \{from(\{t\} \cup T; IK)\}$.

- If $t \in \Sigma_0$ and $intrudable(t)$, i.e. a constant that the intruder knows by definition, we can avoid all attempts to unify this term with any term of the intruder knowledge, and only follow the generate part, by removing the term from the constraint (as there are no subterms of t to be

generated). This modification is correct as the (generate-only) semantics of the constraint is not changed by this modification, and it cannot lead to non-termination as the resulting constraint is simpler according to the weight-function introduced in the proof.

- If $t \in IK$, i.e. the term is directly contained in the intruder knowledge, then we can simply remove t for the same reasons (correctness and termination) as above, and thus spare ourselves the possibilities to construct t from its components or to unify it with other terms in the intruder knowledge.

6.2.2 Analyzing the Intruder Knowledge

We come back to the original problem of constraint solving, without the restriction to generate-only deductions. The idea to integrate analysis steps is the following. After all possible analysis rules are applied, i.e. the intruder knowledge is closed under analysis, we can solve the resulting constraints with the generate-only reduction algorithm. There are, however, several problems which make the separation of generation and analysis difficult:

1. Keys may be composed, so that the intruder may first need to generate a key from different components before he can decrypt a message. Consider, for instance, the ground intruder knowledge $M = \{\mathsf{n}_1, \mathsf{n}_2, \{|\mathsf{m}|\}_{\mathsf{n}_1,\mathsf{n}_2}\}$. It holds that $\mathsf{m} \in \mathcal{DY}(M)$ while no analysis rule of $\mathcal{DY}$ can be applied to M (as the pair $\mathsf{n}_1, \mathsf{n}_2$ has to be composed first in order to apply an analysis rule).

2. Another problem arises from the fact that we are dealing with a symbolic intruder knowledge, and that the question when the intruder knowledge is "completely" analyzed may depend on the substitution. For instance

   $$IK = \{\{\mathsf{m}\}_{\mathsf{pk(A)}}, \mathsf{inv}(\mathsf{pk}(\mathsf{i}))\}$$

   is already completely analyzed, if we exclude the substitution $[\mathsf{A} \mapsto \mathsf{i}]$, but it admits the analysis of m for $[\mathsf{A} \mapsto \mathsf{i}]$. When we commit to said substitution, however, we may exclude solutions, e.g. consider the constraint

   $$\mathit{from}(\{\{\mathsf{m}\}_{\mathsf{pk(a)}}\}; IK) \ ,$$

   which is satisfiable for the solution $[\mathsf{A} \mapsto \mathsf{a}]$ (without any analysis steps).

3. Another problem attached to the variables in the intruder knowledge (and to the split between analysis and generate) is the fact that variables themselves may represent encrypted messages that the intruder may decrypt. Trying to "symbolically" analyze such variables does obviously lead to an infinite chain of analysis steps.

4. Finally, the order in which terms can be decrypted is also not immediately clear. $\mathcal{DY}$ is defined as a closure; in particular, the result of an analysis step may both be decipherable itself and be used to decrypt another message. Again, the symbolic intruder knowledge makes this problem even more difficult, as we cannot see which terms will be substituted for variables during reduction.

We now sketch the idea how to deal with these problems before we give the algorithm.

1. The first problem, the interaction between generate and analysis steps, will be solved using constraints itself, i.e. rather than checking that the key needed for decryption is an element of the intruder knowledge, we will impose the new constraint that the key can be derived from the intruder knowledge.

2. The problem of the analysis under certain substitutions can be solved by case splits.

3. The problem of analyzing variables can be solved using the restriction to well-formed constraints. The idea is the following. Consider a well-formed constraint C with $v \in IK$ for some $\mathit{from}(T; IK) \in C$. By the well-formedness of C, there must be a constraint $\mathit{from}(T; IK_0) \in C$ such that $IK_0 \subsetneq IK$, $v \notin \mathit{vars}(IK_0)$, and $v \in \mathit{vars}(T)$, i.e. an "earlier" constraint on which the variable was introduced. Suppose that this earlier constraint $\mathit{from}(T; IK_0)$ is simple, i.e. $v \in T$. Then, for every solution $\sigma \in [\![C]\!]$, it holds that $v\sigma \in \mathcal{DY}(IK_0\sigma) \subseteq \mathcal{DY}((IK \setminus \{v\})\sigma)$. Intuitively, v represents a term that the intruder has generated himself before. Thus we can safely eliminate v from the intruder knowledge without losing any solution. Note that this however requires that the "earlier constraints" are already simple, i.e. for analysis we have to proceed along the order imposed by the well-formed constraints.

4. The last problem is addressed in the algorithm for analysis by a case split over different sequences of analysis steps.

Definition 6.6. *We define the algorithm ana that takes a set of terms IK and returns a set of pairs (M, C) of a set of terms M and a set of constraints C:*

$$\mathit{ana}(IK) = \bigcup_{\substack{((k_1,M_1),\ldots,(k_m,M_m)) \\ \in \mathit{anaterms}(IK), \\ 0 \le n \le m}} (M_1 \cup \ldots \cup M_n, \{\ \mathit{from}(\{k_1\}; IK), \mathit{from}(\{k_2\}; IK \cup M_1), \ldots, \mathit{from}(\{k_n\}; IK \cup M_1 \cup \ldots M_{n-1})\ \})$$

$$\mathit{anaterms}(IK) = \begin{cases} \{\epsilon\} & \text{if } \mathit{decana}(t) = \emptyset \text{ for all } t \in IK \\ \{((k,M), KCs) \mid\ t \in IK \wedge (k,M) \in \mathit{decana}(t) \wedge KCs \in \mathit{anaterms}((IK \setminus \{t\}) \cup M)\ \} & \text{otherwise} \end{cases}$$

$$\mathit{decana}(t) = \begin{cases} \{(k, \{c\})\} & \text{if } t = \{\!|c|\!\}_k \\ \{(\mathsf{inv}(k), \{c\})\} & \text{if } t = \{c\}_k \\ \{(\mathsf{i}, \{c\})\} & \text{if } t = \mathsf{sign}(k, c) \\ \{(\mathsf{i}, \{c_1, c_2\})\} & \text{if } t = \langle c_1, c_2 \rangle \\ \emptyset & \text{otherwise} \end{cases}$$

where ϵ stands for the empty sequence.

Intuitively, this algorithm computes, for a given (symbolic) intruder knowledge IK, a set of cases for the analysis, where each case (M, C) is pair of a set of messages M that could be derived provided that we can derive the necessary key-terms according to the constraint set C.[1]

To do that, the algorithm first calls *anaterms* to determine a sequence of analysis steps that can be performed (we will describe this algorithm below). These analysis steps have the form (k, M), meaning intuitively that the intruder can obtain the clear-text messages M from some encrypted message in his knowledge, if he can generate the key k. If we have any sequence of m such analysis steps, we choose some prefix (i.e. the first n elements) of this sequence. The reason for choosing a prefix of the sequence is problem 2 mentioned above: we may choose not to perform all analysis steps because they may involve substitutions that later exclude some solutions.

The constraints necessary to derive the clear-texts are that the ith key in the sequence can be generated from the intruder knowledge IK and the first $i - 1$ sets of clear-text messages.

The *anaterms* algorithm chooses a term t from the given IK that can be analyzed according to the *decana* function, i.e. if the intruder knows the key-term k, then he can obtain the set of clear-texts M. After that, *anaterms* analyzes the other terms $(IK \setminus \{t\})$ and the newly learned clear-texts M. This algorithm thus computes every possible sequence of analysis steps on the symbolic intruder knowledge (except analyzing variables in the intruder knowledge).

[1] This constraint set is not necessarily well-formed, as it may contain variables that are not introduced on some left-hand side. The constraint set will later be used together with other constraints in a well-formed way.

Example 6.3. We continue with Example 6.2, showing that the analysis steps (without substitution) are found by the analysis algorithm, i.e. we consider $ana(IK_0 \cup \{m_2, m_3\})$. We consider a similar simplification as in Example 6.2, namely we consider the pair m_3 as two messages. Then there are only three messages for which *decana* does not return the empty set, namely the symmetric encrypted ones. Let us pick the first part of m_3, namely $\{|\mathsf{B}', \mathsf{kab}, \mathsf{NA}', \mathsf{NB}'|\}_{\mathsf{k}(\mathsf{A}')}$. This will yield an analysis sequence that starts with the pair

$$(\mathsf{k}(\mathsf{A}'), \{\mathsf{B}', \mathsf{kab}, \mathsf{NA}', \mathsf{NB}'\}) ,$$

where we have again already decomposed the pairs in the clear-text for simplicity.

As a possible result of *ana*, we thus have that for all solutions that satisfy the constraint $from(\mathsf{k}(\mathsf{A}'); IK_0 \cup \{m_2, m_3\})$ (and the other constraints of this problem), the intruder can add the messages $\mathsf{B}', \mathsf{kab}, \mathsf{NA}', \mathsf{NB}'$ to all constraints that have as knowledge a superset of $IK_0 \cup \{m_2, m_3\}$. We will later continue this example to see that this new constraint is indeed satisfiable for the substitution given in Example 6.2. □

The correctness of *ana* is related to the entire constraint reduction algorithm *Red*, which we define first before we prove any properties of *ana*.

6.2.3 Integration of Generation and Analysis

The idea for the integration of generation and analysis is to follow the order on the given set of constraints that is induced by the well-formedness. We successively add constraints in this order to a set of simple constraints by first analyzing the intruder knowledge of the new constraint (with respect to the simple ones), then adding the new constraint with the analyzed intruder knowledge, and finally reducing this constraint set using Red^{gen}:

Definition 6.7. *Let C_1 and C_2 be constraint sets such that C_1 is well-formed and simple, C_2 is well-formed, and let σ be a substitution. We define the following reduction algorithm:*

$$Red(C_1, C_2, \sigma) = \begin{cases} \{(C_1, \sigma)\} & \text{if } C_2 = \emptyset \\ \{(C''', \sigma''') \mid & (IK', C') \in ana(IK) \wedge \\ & (C'', \sigma'') \in Red^{gen}(C_1 \cup C' \cup \{from(T; IK \cup IK')\}, \sigma) \wedge \\ & (C''', \sigma''') \in Red(C'', (C_2 \setminus \{from(T; IK)\})^{IK'}\sigma'', \sigma'')\} \\ & \text{if } fwo(C_1, C_2) = from(T; IK) \end{cases}$$

where $fwo(C_1, C_2)$ is the first constraint of C_2 in the order induced by the well-formedness of $C_1 \cup C_2$ (if $C_2 \neq \emptyset$), and where

$$\begin{aligned} from(T; IK)^{IK'} &= from(T; IK \cup IK') \\ \{c_1, \ldots, c_n\}^{IK'} &= \{c_1^{IK'}, \ldots, c_n^{IK'}\} \end{aligned}$$

We also write $Red(C)$ as a short-hand for $Red(\emptyset, C, id)$.

The algorithm works on a constraint set C that is separated into two parts C_1 and C_2. C_1 is already simple and no further analysis steps will be performed on C_1. C_2 is yet to be reduced (which may lead to substitutions also in C_1 that require further reductions there). If C_2 is empty, then everything is done, as we have a simple constraint set C_1 and just return it. Otherwise, we pick the first constraint $from(T; IK)$ in C_2 according to the order of the well-formedness of $C_1 \cup C_2$. We then choose a sequence of analysis steps to be performed on IK, i.e. a pair $(IK', C') \in ana(IK)$. Here, IK' is the set of new messages that can be obtained, and the constraint set C' expresses that the intruder can get the necessary keys (without any further analysis steps). Next, we reduce the initial part C_1 and the new constraints C' with Red^{gen}, as both C_1 and C' require no analysis

steps. The resulting constraints C'' form the new simple part, i.e. the first argument, in the next round of *Red*. The second argument are those constraints of C_2 that have not yet been processed, i.e. $C_2 \setminus \{from(T; IK)\}$ with two modifications: we add the newly learned intruder knowledge IK' to each constraint in C_2, as they have all an intruder knowledge that is a superset of IK and therefore the same messages can be derived by analysis, and then we apply the substitution σ'' resulting from the reduction with Red^{gen}.

Note that we cannot proceed in the reduction of constraints in an arbitrary order, but must stick order induced by the well-formedness of $C_1 \cup C_2$. The reason is that the algorithm is only complete if all variables in the intruder knowledge that is being analyzed occur in an earlier constraint that is already simple (and thus every variable represents a term that the intruder has created before from a smaller knowledge).

Example 6.4. We conclude the development of the running example 6.1, 6.2, and 6.3. Let C be the constraint set defined in Example 6.1. We follow one of the reductions computed by $Red(C)$.

The reduction of the first constraint, $from(\{m_1\}; IK_0)$, is trivial: there is nothing to analyze and (considering the pair m_1 as already decomposed) it is also already simple.

For the second constraint, $from(\{m_2'\}; IK_0 \cup \{m_2\})$, there is the possibility to analyze m_2 which is encrypted with $\mathsf{k(b)}$, but as the intruder cannot generate $\mathsf{k(b)}$, the resulting key-generation constraint is unsatisfiable. We must therefore follow the case in which the term is left unanalyzed. To generate the left-hand side term m_2', we have two possibilities returned by Red^{gen}. First, the intruder could generate the term from its components; this would lead to the match $B' = \mathsf{i}$ as there is no other way for the intruder to generate the key $\mathsf{k(B')}$. We do not follow this possibility here. Second, the intruder can unify this term with m_2 in his intruder knowledge, leading to the substitution $\sigma = [\mathsf{B'} \mapsto \mathsf{b}, \mathsf{A'} \mapsto \mathsf{A}, \mathsf{NA'} \mapsto \mathsf{NA}, \mathsf{NB'} \mapsto \mathsf{nb}]$.[2]

For the third constraint, $from(\{m_4\sigma\}; IK_0 \cup \{m_2, m_3\}\sigma)$, we can now perform an analysis step on the first component of the pair $m_3\sigma$, namely $\{|\mathsf{B, kab, NA, nb}|\}_{\mathsf{k(A)}}$. This leads to the new knowledge $IK' = \{\mathsf{B, kab, NA, nb}\}$ (implicitly decomposing all pairs again for simplicity) and the additional constraint $from(\{\mathsf{k(A)}\}; IK_0 \cup \{\mathsf{m_2, m_3}\}\sigma)$ (as already explained in Example 6.3). This constraint is easily reduced by Red^{gen} as $\mathsf{k(i)} \in IK_0$, leading to the unifier $A \mapsto i$. The left-hand side term $m_4\sigma$ is reduced by Red^{gen} as in Example 6.2, leading to the final substitution $[\mathsf{A} \mapsto \mathsf{i}, \mathsf{A'} \mapsto \mathsf{i}, \mathsf{NB} \mapsto \mathsf{nb}, \mathsf{NB'} \mapsto \mathsf{nb}, \mathsf{KAB} \mapsto \mathsf{NA}, \mathsf{nb}]$ and leaves the variable NA unsubstituted. □

We now show the main theorem of the lazy intruder, i.e. that *Red* is correct and terminates:

Theorem 6.2. *Let C be a well-formed constraint set. Then $Red(C)$ terminates and returns a finite set of pairs of a simple, well-formed constraint and a substitution such that*

$$[\![C]\!] = \bigcup_{(C',\sigma') \in Red(C)} [\![(C', \sigma')]\!] .$$

Moreover, for every $(C', \sigma') \in Red(C)$, $vars(C') \cap vars(\sigma') = \emptyset$, $vars(C') \cup vars(\sigma') = vars(C)$.

Proof. We split the proof into four parts, namely invariants, the termination, the soundness, and the completeness.

Invariants We first show that several invariants are preserved throughout *Red*. If *Red* is called with arguments (C_1, C_2, σ) such that $C_1 \cup C_2$ is a well-formed constraint set, C_1 is a well-formed and simple constraint set, and σ is a substitution with $dom(\sigma) \cap vars(C_1 \cup C_2) = \emptyset$, then for every recursive call to $Red(C_1', C_2', \sigma')$ it holds that

- C_1' is well-formed and simple,
- $C_1' \cup C_2'$ is well-formed,

[2] This substitution is incomparable to the substitution σ considered in Example 6.2; there we have started with a substitution that allows for the analysis of nb decrypting message m_3. We will see that the reduction considered here results in the same final substitution as in Example 6.2, with the difference that here the substitution is not given, but found during analysis.

- $vars(C_1' \cup C_2') \cap dom(\sigma') = \emptyset$, and
- $vars(C_1' \cup C_2') \cup dom(\sigma') = vars(C_1 \cup C_2) \cup dom(\sigma)$.

Moreover, all calls to Red^{gen} are with arguments as required, namely with (C', σ') such that C' is well-formed and $vars(C') \cap dom(\sigma') = \emptyset$. Also, observe that the initial call $Red(C)$ for a well-formed constraint set C satisfies the preconditions, since $Red(C)$ is a short-hand for $Red(\emptyset, C, id)$.

Consider a call to $Red(C_1, C_2, \sigma)$ that satisfies the invariants and where $C_2 \neq \emptyset$. Then for every pair $(IK', C') \in ana(IK)$ we have that $vars(IK') \subseteq vars(C') \subseteq vars(C_1)$ since $C_1 \cup \{from(T; IK)\}$ is well-formed and in both IK' and C' only variables can occur that also occur in C_1. Moreover, the intruder knowledge in every constraint of C' is a superset of all intruder knowledges in C_1, and the intruder knowledges in C' are monotonically increasing in the order they were created by ana. Thus $C_1 \cup C'$ is well-formed. Since $IK \cup IK'$ is a superset of all intruder knowledges in $C_1 \cup C'$ and contains only variables from C_1, also $C_1 \cup C' \cup \{from(T; IK \cup IK')\}$ is well-formed. Finally, $dom(\sigma)$ is disjoint from the variables in $C_1 \cup C_2$, thus the conditions for Red^{gen} are met in the call during Red.

In order to show that the conditions for the recursive call to Red are met we use the properties of Red^{gen} (as proven in Theorem 6.1), namely that for every resulting pair $(C'', \sigma'') \in Red^{gen}(\ldots)$, we have that C'' is simple and well-formed, $vars(C'') \cap dom(\sigma'') \neq \emptyset$, and that the set of variables in C' and in the domain of σ' is the same as in the arguments of the call to Red^{gen}. Thus we have

$$\begin{aligned} vars(C'') \cup dom(\sigma'') &= vars(C_1 \cup C' \cup \{from(T; IK \cup IK')\}) \cup dom(\sigma) \\ &= vars(C_1) \cup vars(T) \cup dom(\sigma) \ . \end{aligned} \tag{6.3}$$

Therefore, in the recursive call to Red, we have the same set of variables as in $vars(C_1 \cup C_2) \cup dom(\sigma)$. To see that, note that the set of variables is

$$\begin{aligned} V &= vars(C'' \cup (C_2 \setminus \{from(T; IK)\})^{IK'}\sigma'') \cup dom(\sigma'') \\ &= vars(C'') \cup dom(\sigma'') \cup vars((C_2 \setminus \{from(T; IK)\})^{IK'}\sigma'') \ . \end{aligned}$$

Since $vars(T \cup IK \cup IK') \subseteq vars(C'') \cup dom(\sigma'')$, we have that

$$V = vars(C'') \cup dom(\sigma'') \cup vars(C_2) \ .$$

By (6.3), we have that

$$V = vars(C_1 \cup C_2) \cup dom(\sigma'') \ .$$

Next, we show that the variables of the constraint set $C'' \cup (C_2 \setminus \{from(T; IK)\})^{IK'}\sigma''$ and the domain of σ'' are still disjoint: $vars(C'') \cap dom(\sigma'')$ are already disjoint by the properties of Red^{gen}. The substitution σ'' is applied to the newly added constraints, so the variables of the result are also disjoint from $dom(\sigma'')$.

Finally, we have to show that the constraints in the recursive call are well-formed. By the properties of Red^{gen}, we have that C'' is already well-formed and simple. Also $IK \cup IK'$ is a superset of all intruder knowledges of C'. The same does not necessarily hold for C'' anymore, since Red^{gen} may have substituted variables (as part of σ''), but as Red^{gen} does not perform any changes to intruder knowledges other than substitution, we have that $(IK \cup IK')\sigma$ is a superset of each intruder knowledge in C''. The second argument of the recursive call is obtained by first removing the processed constraint $from(T; IK)$ from C_2, then adding the new intruder knowledge IK' to all remaining constraints of C_2, and finally applying the current substitution σ'' to the constraint set. Hence, we have that all constraints in this set contain at least the intruder knowledge $(IK \cup IK')\sigma''$, thus their intruder knowledge is at least as large as the intruder knowledge of any constraint in C''. Moreover, in these constraints, the only new variables (that do not appear in C'') are those that have been present in $C_2 \setminus \{from(T; IK)\}$. We can thus conclude that $C'' \cup (C_2 \setminus \{from(T; IK)\})^{IK'}\sigma''$ is a well-formed constraint.

Finally, we look at the case $C_2 = \emptyset$ when Red terminates. By the invariant, it follows that C_1 is a simple and well-formed constraint, and that $dom(\sigma) \cap vars(C_1) = \emptyset$. Thus every result returned by Red is a pair of a simple well-formed constraint set and substitution with disjoint domain.

Termination As the first step, we show that *anaterms* terminates for each input. To that end, we show that in every recursive call of *anaterms*, the weight of the intruder knowledge is strictly smaller according to the weight-function w introduced in the proof of Theorem 6.1, i.e. we show that

$$\begin{aligned} w((IK \setminus \{t\}) \cup M) &\leq w(IK) - w(t) + w(M) \\ &< w(IK) \end{aligned}$$

The $\leq$ part is immediate, and it is thus sufficient to show that $w(t) > w(M)$, which follows from the definition of $decana(t)$: for instance, in the case $t = \{\!|m|\!\}_k$ we have $w(t) = 1+w(k)+w(m) > w(m)$. The other cases are similar. The termination of *anaterms* now follows from the fact that IK is finite, and thus there can only be finitely many recursive calls with an intruder knowledge of strictly smaller weight.

The termination of *ana* follows from the termination of *anaterms*. For the termination of *Red*, observe that besides the calls to the terminating algorithms *ana* and Red^{gen}, there is the recursive call to *Red* itself. In each such recursive call, the number of elements in the constraint set given in the second argument is reduced by one, and the algorithm terminates when this argument contains the empty set.

Soundness For the soundness, we have to show that $[\![(C', \sigma')]\!] \subseteq [\![C]\!]$ for every $(C', \sigma') \in Red(C)$.

First, observe that we have already shown the soundness of Red^{gen} with respect to the generate-only semantics $[\![\cdot]\!]^{gen}$. Thus, Red^{gen} is also sound with respect to the unrestricted semantics, since $[\![C]\!]^{gen} \subseteq [\![C]\!]$.

Second, using the algorithm *ana*, we introduce new constraints to an already simple constraint set. This restricts the set of solutions of this simple constraint set to those allowed by the new (key-derivation) constraints, thus this is not a problem for soundness. However, we have to show that it is sound to add to every subsequent constraint the terms that result out of this analysis, which is only the case if the key-derivation constraints are given.

To that end, observe that for any intruder knowledge IK, any term $t \in IK$, and any ground substitution σ with $dom(\sigma) \supseteq vars(IK)$, it holds that $M\sigma \subseteq \mathcal{DY}(IK\sigma)$ for every $(k, M) \in decana(t)$ with $k\sigma \in \mathcal{DY}(IK\sigma)$ (which follows straightforwardly by the respective analysis rules of $\mathcal{DY}$).

For *ana*, we conclude that for every σ_0 with

$$\sigma_0 \in [\![\{from(\{k_1\}; IK), from(\{k_2\}; IK \cup \{M_1\}), \ldots, from(\{k_n\}; IK \cup \{M_1, \ldots, M_{n-1}\})\}]\!] ,$$

it holds that $(M_1 \cup \ldots M_{n-1})\sigma_0 \subseteq \mathcal{DY}(IK\sigma_0)$.

For *Red*, we can thus conclude that, if the intruder knowledge IK' is added to every constraint of C_2, then every ground substitution that satisfies the new constraint also satisfies the old constraint. Formally, $\sigma_0 \in [\![(C_1 \cup C_2, \sigma)]\!]$ implies

$$\sigma_0 \in [\![(C'' \cup (C_2 \setminus \{from(T; IK)\})^{IK'}\sigma'', \sigma'')]\!] ,$$

since $IK'\sigma_0 \subseteq \mathcal{DY}(IK\sigma_0)$ and $NIK \supseteq IK$ for every right-hand side NIK of a constraint in C_2.

Hence, we have that the extension of the intruder knowledges is sound, and thus *Red* is sound.

Completeness Like in the completeness part of the proof of Theorem 6.1, we consider a solution $\sigma_0 \in [\![C]\!]$, and show that this solution remains preserved throughout the algorithm in at least one case of the reduction process of $Red(C)$. Also, we again assume that every left-hand-side term t of a constraint of C is labeled with a Dolev-Yao proof showing that $t\sigma_0$ can be derived from $IK\sigma_0$ where IK is the respective right-hand side.

Following the structure of *Red*, we will consider a simple well-formed constraint C_1 in which all left-hand side terms are labeled with Dolev-Yao proofs (for the respective intruder knowledge and σ_0) that contain only generate rules. Further, we consider a constraint $from(T; IK)$ according to *Red* where the left-hand side terms are also labeled with Dolev-Yao proofs (for IK and σ_0), but

that may also contain analysis rules. We show that there is a pair $(C', IK') \in ana(IK)$ such that the terms of T and all left-hand sides of C' can be labeled with generate-only Dolev-Yao proofs for σ_0 and $IK \cup IK'$. Then completeness follows from the fact that Red^{gen} is complete with respect to the generate-only semantics.

To show that such a $(C', IK') \in ana(IK)$ exists, we determine a prefix $(0 \leq m \leq n)$ of a sequence of analysis steps $((k_1, M_1), \ldots, (k_n, M_n)) \in anaterms(IK)$ such that there is a labeling of all constraints with respect to σ_0: each k_i is labeled with a generate-only Dolev-Yao proof for $k_i\sigma_0$ with leaves in $(IK \cup M_1 \cup \ldots M_{i-1})\sigma_0$. This gives us a labeling for the left-hand side terms in C' with respect to σ_0. Moreover, the constraint $from(T; IK \cup IK')$ can be labeled with generate-only Dolev-Yao proofs for σ_0, since $IK' = M_1 \cup \ldots \cup M_m$.

To determine an appropriate sequence $((k_1, M_1), \ldots, (k_m, M_m))$ of analysis steps, we will step-by-step consider the analysis rules in the Dolev-Yao proofs of T, namely identifying a possible further analysis step for the sequence and replacing the respective analysis rule in the proof with a leaf node (that refers to a message in $M_i\sigma_0$). We start at any of the "inner-most" analysis steps, i.e. steps such that no sub-tree contains analysis steps.

We now show that there is an analysis step in a Dolev-Yao of proof of T such that the analyzed term is a leaf node and that it matches a non-variable term in IK.

First, consider the case that we have a sub-proof of the form

$$\dfrac{\dfrac{\dfrac{P_1}{k} \quad \dfrac{P_2}{c}}{\{\!|c|\!\}_k} \quad \dfrac{P_2}{k}}{c}$$

for some Dolev-Yao proofs P_1 and P_2. We can simplify this sub-proof into

$$\dfrac{P_2}{c}$$

since it contains the redundancy that the intruder first encrypts a term and then decrypts it again. Similar simplifications can be performed for encryption and decryption with asymmetric cryptography and pairing. We will assume in the following that all Dolev-Yao proofs are simplified in this way. This has the consequence that there cannot be a decryption of a term that has been composed by the intruder (only the key-term may be composed by the intruder) but an analyzed term can only be at a leaf of the proof tree or at a node of an analysis step.

Thus consider any analysis rule used in the Dolev-Yao proofs of T, such that the subproofs do not contain further analysis rules. Let s be the analyzed term in the proof. Thanks to the simplification of the Dolev-Yao proofs, s cannot result out of a composition and, since we consider a node with no analysis nodes in the subtrees, s can only be a leaf node, i.e. we have the situation

$$\dfrac{\overline{s} \quad \dfrac{P}{k}}{c} \; A_{\text{scrypt}}, A_{\text{crypt}}, A_{\text{sign}}, \text{ or } A_{\text{pair}_i}$$

(without the subtree $\dfrac{P}{k}$ in the cases A_{pair_i} or A_{sign}), where P is a generate-only Dolev-Yao proof for the key-term k.

Since s is a leaf node, there is a term $t \in IK$ such that $s = t\sigma_0$. Let $IK_t = \{t' \in IK \mid t'\sigma_0 = s\}$. (And note that $IK_t \neq \emptyset$.)

We now consider how to eliminate the case that $IK_t \subseteq \mathcal{V}$. Suppose $IK_t \subseteq \mathcal{V}$. We choose a $t \in IK_t$ that is minimal in the sense that no other $t' \in IK_t$ appears in an earlier constraint (according to the well-formedness order) in $C_1 \cup C_2$. We can use the following transformation of the Dolev-Yao proof to either arrive at a contradiction to $IK_t \subseteq \mathcal{V}$ or to show that the analysis step is redundant. Since the constraint set $C_1 \cup C_2$ is well-formed and we have considered the "first" constraint in C_2, there is a constraint $from(T_0; IK_0) \in C_1$ with $t \in T_0$ (since C_1 is simple) and $IK_0 \subseteq IK$. Intuitively speaking, the variable t is a term that the intruder created "earlier" (from a smaller knowledge). Moreover, since t is minimal in the set IK_t with respect to occurrence

in earlier constraints, $IK_0 \cap IK_t = \emptyset$. In $\mathit{from}(T_0; IK_0)$, the term t is labeled with a Dolev-Yao proof dt for the term $t\sigma_0$ that contains only generate-rules and whose leaves are from $IK_0\sigma_0$. We can thus replace the leaf node for s with the Dolev-Yao proof dt without destroying the correctness of the labeling. We distinguish two cases for the form of dt. First, if dt is a leaf, there is a term $t' \in IK_0$ with $s = t'\sigma_0$, thus $t' \in IK_t$, contradicting $IK_0 \cap IK_t = \emptyset$. Second, since dt contains only generate rules, we can simplify the resulting Dolev-Yao proof again, removing the analysis node in question, since the intruder analyzed a term he composed himself earlier.

So we can now assume that there is a term $t \in IK_t$ with $t \notin \mathcal{V}$, i.e. it must be of the form $\{|t_2|\}_{t_1}$, $\{t_2\}_{t_1}$, $\mathsf{sign}(t_1, t_2)$, or $\langle t_1, t_2\rangle$, respectively. We show that using $\mathit{decana}(t)$, the *ana* algorithm will identify this possible decryption of a term in the symbolic intruder knowledge IK.

We consider only the case that the analysis rule is A_{scrypt}, the other cases are similar, mutatis mutandis. We have $t = \{|t_2|\}_{t_1}$ and $\mathit{decana}(t) = \{(t_1, \{t_2\})\}$. Thus, for IK, we can obtain a sequence that starts with $(t_1, \{t_2\})$ from $\mathit{anaterms}(IK)$. In the Dolev-Yao proof, we can thus replace the analysis rule with a leaf for $IK \cup \{t_2\}$ and σ_0, and label t_1 in the new constraint $\mathit{from}(\{t_1\}; IK)$ with the proof P that labels the key-term in the analysis rule (which contains only generate rules and leafs in $IK\sigma_0$).

In all Dolev-Yao proofs, we replace all other occurrences of the same analysis step, i.e. analyzing the same term s within any proof of T, since the result of the analysis step is a term of $M_i\sigma_0$, and thus also present in the new intruder knowledge. This is necessary since we will not consider analysis of the same term t anymore: the rest of the sequence of steps will be taken from $\mathit{anaterms}(IK \setminus \{t\} \cup M)$.

We can now repeat the same steps for the constraint $\mathit{from}(T; IK \cup \{t_2\})$ and $\mathit{anaterms}(IK \setminus \{t\} \cup M)$ until there are no more analysis rules in the proofs of T. We finally arrive at the required constraints labeled with generate-only Dolev-Yao proofs under σ_0. Thus $\mathit{Red}^{gen}(C_1 \cup C' \cup \{\mathit{from}(T; IK \cup IK')\})$ will obtain a solution that still supports σ_0. The completeness of the entire algorithm is obtained by induction over the recursive structure of *Red*, namely that we step-by-step add constraints (in the order of the well-formedness) and never exclude the substitution σ_0. This concludes the completeness proof of *Red*, and thus the proof for the correctness and termination of *Red*. □

6.2.4 Improvements to the Analysis

As already mentioned, the analysis can lead to a large number of cases, consisting of all prefixes of all sequences of analysis steps. We will now consider several improvements that can be made to reduce the number of cases.

- The "analysis" of both t_1, t_2 and $\mathsf{sign}(t_1, t_2)$ is possible without knowing any keys. Therefore, in $\mathit{decana}(\cdot)$, we have used as derivation key the trivial message i (that the intruder can always construct) for uniformity in the definition of analysis. There is a straightforward optimization, namely we can avoid to regard these decompositions as analysis steps entirely, by ensuring that the intruder knowledge is always closed under decomposition of pairs and signatures. This addition of the components of pairs and the content of signatures in the intruder knowledge does not change the semantics of a constraint. The well-formedness of constraints is also not destroyed as long as the addition is performed on all intruder knowledges in a constraint set. For termination, observe that all such decompositions are always allowed by the analysis anyway, and we have just made them mandatory, excluding some cases of $\mathit{anaterms}(IK)$ (while we do not need to consider such terms in $\mathit{decana}(\cdot)$ anymore).

- For pairs, we can even go a step further and remove the analyzed pair from the intruder knowledge, as it can also be composed by the intruder whenever necessary. Again this does not change the semantics of a constraint set or destroy its well-formedness when performed on all intruder knowledges. For termination observe that removing constraints from the intruder knowledge can lead only to fewer and shorter sequences in $\mathit{ana}(IK)$. Similarly,

we can remove symmetrically encrypted messages, whenever they have been successfully analyzed, since the intruder knows the key and can thus re-encrypt the messages.

- *Red* "chooses" a sequence of analysis steps and then later checks that all the necessary keys can be generated. Consider the case of a sequence $((k_1, M_1), \ldots, (k_n, M_n))$ where the key k_i for some $i \in \{1, \ldots, n\}$ cannot be generated, i.e. the resulting constraint (together with the simple constraint set) is unsatisfiable. In this case, all sequences that start with $((k_1, M_1), \ldots, (k_i, M_i))$ lead to an unsatisfiable constraint set. Thus, it is not necessary to explore the search space of sequences that are an extension of an already unsatisfiable sequence. Rather, we can interleave analysis steps with the Red^{gen} algorithm to check that every generated constraint is satisfiable, and abort the exploration of possible extensions of the analysis sequence as soon as we find an unsatisfiable constraint.[3]

- Before determining a sequence of analysis steps, we can perform a filtering process on the intruder knowledge to be analyzed as follows.[4] First, we can exclude from the analysis every message $t \in IK$ such that for every $(k, M) \in decana(t)$ it holds that $M \subseteq IK$, i.e. where the result would not give the intruder any message that he did not already know. Second, we can check if the respective key can be generated without substitutions, i.e. whether $(C'', id) \in Red^{gen}(C_1 \cup \{from(\{k\}; IK)\})$ (where C_1 are the simple constraints as in body of *Red*). Then, the derivation of M is without loss of generality in the sense that we do not commit to any substitution by this analysis step. Formally, $[\![(C_1 \cup C_2, \sigma)]\!] = [\![(C_1 \cup C_2^M, \sigma)]\!]$ in this case (where C_2 are as in the body of *Red*), since for every $\sigma_0 \in [\![C_1 \cup C_2]\!]$, $M\sigma_0 \subseteq \mathcal{DY}(IK\sigma_0)$. By this filtering before $anaterms(\cdot)$, we reduce the number of prefixes of sequences of analysis steps in $anaterms(IK)$.

6.3 Inequality Constraints

We now augment the lazy intruder constraint formalism with inequalities, which are necessary for the handling of negative facts and conditions in the transition rules of IF protocol descriptions. These inequalities are part of the symbolic states that we will consider and act as additional constraints on variables. We first define the syntax and semantics of such inequalities, and then show how they can be integrated into the constraint reduction.

Definition 6.8. *We allow the following form of inequality constraints, where we again use the bold-font symbols* $\mathbf{\wedge}$ *and* $\mathbf{\vee}$ *for distinction from the meta-connectives:*

$$Ineq ::= \mathcal{T}_\Sigma(\mathcal{V}) \neq \mathcal{T}_\Sigma(\mathcal{V}) \mid Ineq \mathbf{\wedge} Ineq \mid Ineq \mathbf{\vee} Ineq$$

The semantics is defined in the standard way: given an interpretation σ, i.e. a ground substitution for all variables of an inequality, we define the models-relation, indexed by the algebra $\mathcal{A}$ in which the terms are interpreted:

$$\begin{array}{lll} \sigma \models_{\mathcal{A}} s \neq t & \textit{iff} & s \not\approx_{\mathcal{A}} t \\ \sigma \models_{\mathcal{A}} \phi \mathbf{\wedge} \psi & \textit{iff} & \sigma \models_{\mathcal{A}} \phi \ \wedge \ \sigma \models_{\mathcal{A}} \psi \\ \sigma \models_{\mathcal{A}} \phi \mathbf{\vee} \psi & \textit{iff} & \sigma \models_{\mathcal{A}} \phi \ \vee \ \sigma \models_{\mathcal{A}} \psi \end{array}$$

In the case of the free algebra, we omit the index $\mathcal{A}$.

We say an inequality ϕ is satisfiable in $\mathcal{A}$*, and write $satisfiable_{\mathcal{A}}(\phi)$ if there is an interpretation σ that $\sigma \models_{\mathcal{A}} \phi$.*

[3] One can exploit the lazy evaluation of Haskell here, by representing the set of sequences as a lazy tree (where each node represents a common prefix of sequences) and exploring this tree lazily in the constraint reduction, avoiding to generate extensions whenever a prefix has turned out to be unsatisfiable.

[4] We do not remove messages from the intruder knowledge, but only remove them from $anaterms(\cdot)$.

It is straightforward to check whether inequalities are satisfiable, as long as the word problem in $\mathcal{A}$ is decidable. We will consider only the case of the free algebra here, for which the word problem is simply the syntactic equality.

The idea for integrating lazy intruder constraints with inequalities is the following. Suppose we have a simple (and thus satisfiable) constraint set C and an inequality ϕ with $vars(\phi) \subseteq vars(C)$ such that ϕ is satisfiable. Then there is substitution for $vars(C)$ that satisfies both C and ϕ: the intruder can generate an arbitrary number of different fresh constants, substituting the variables in C, and the inequalities are satisfied under this substitution:

Lemma 6.2. *Let ϕ be an inequality and $vars(\phi) = \{v_1, \ldots, v_n\}$. Let $c_1, \ldots, c_n$ be fresh constants (that do not occur in ϕ) with $c_i \neq c_j$ for $i \neq j$. Let $\tau = [v_1 \mapsto c_1, \ldots, v_n \mapsto c_n]$. Then satisfiable($\phi$) iff satisfiable($\phi\tau$).*

Proof. Assume ϕ is satisfiable, but $\phi\tau$ is not. (The other direction is obvious.) Then there is at least one inequality $s \not\doteq t$ in ϕ such that $s \neq t$, while $s\tau = t\tau$. Due to the free algebra, there is a position $p \in pos(s) \cap pos(t)$ such that $s_0 = root(s|p) \neq t_0 = root(t|p)$. We distinguish the following cases:

- $s_0 \in \mathcal{V}$ and $t_0 \in \mathcal{V}$. It follows that $s\tau|p = c_i$ and $t\tau|p = c_j$ for two indices $i \neq j$. In particular, $c_i \neq c_j$ and therefore $s\tau \neq t\tau$, contradicting the assumption.
- $s_0 \notin \mathcal{V}$ and $t_0 \notin \mathcal{V}$. Then $root(t\tau|p) = t_0$ and $root(s\tau|p) = s_0$, thus $s\tau \neq t\tau$, yielding the same contradiction as in the previous case.
- $s_0 \in \mathcal{V}$ and $t_0 \notin \mathcal{V}$. In this case, $s\tau|p = c_i$ for some index i. Suppose $t\tau|p = c_i$, then $t|p = c_i$, thus c_i is a subterm of ϕ contradicting the requirements on c_i. So $s \neq t$.
- $s_0 \notin \mathcal{V}$ and $t_0 \in \mathcal{V}$. this case is analogous to the previous one.

Thus, in all cases we obtain the contradiction, and therefore if $\phi\tau$ is unsatisfiable, then so is ϕ. □

We can thus conclude:

Theorem 6.3. *Let C be a simple set of lazy intruder constraints, and ϕ an inequality that is satisfiable in the free algebra with $vars(\phi) \subseteq vars(C)$. Then there is a substitution $\sigma \in [\![C]\!]$ with $\sigma \models \phi$.*

Proof. Let $vars(C) = \{v_1, \ldots, v_n\}$. Recall that $\mathbb{N} \subseteq \Sigma_0$ and that for $n \in \mathbb{N}$, *intrudable*(n). Let k be the maximum of all numbers that occur in C and ϕ. We consider the substitution $\sigma = [v_1 \mapsto k+1, \ldots, v_n \mapsto k+n]$.

This means that the intruder uses for each variable a natural number that has never occurred before. By Lemma 6.2, $\phi\sigma$ is satisfiable. Since all natural numbers are intrudable, $\sigma \in [\![C]\!]$. □

Therefore, to check the satisfiability of a combination of a set C of well-formed lazy intruder constraints and an inequality ϕ, we first apply the reduction algorithm of the previous section to C, yielding a set of simple constraints and substitutions. For each of these (C_i, σ_i), we check that $\phi\sigma_i$ is satisfiable. Formally, there is a σ with $\sigma \in [\![C]\!]$ and $\sigma \models \phi$ iff there is a $(C_i, \sigma_i) \in Red(C)$ and $\phi\sigma_i$ is satisfiable.

Chapter 7

An ADT for the Lazy Intruder

The previous chapter has introduced a form of constraints, along with a reduction algorithm to express Dolev-Yao intruder deduction symbolically. Before we show how to integrate these constraints into the algorithms for checking protocol security, we perform an abstraction step and consider sets of constraints as an abstract data-type (ADT), called *constraint stores*. This allows us to hide the more technical aspects like performing substitutions and keeping track of the intruder knowledge. Instead, we have just a set of functions like "learn message" for manipulating the constraint store on a more abstract level.

Besides making the presentation of the subsequent chapters less technical, the ADT also highlights the generality of the idea behind the lazy intruder, as it is not tailored towards the usage in a particular formalism like the intermediate format.

Definition 7.1. *We define an abstract data-type CS (for Constraint Store) with the following operations:*

$$\begin{aligned}
\mathit{initial} &: CS \\
\mathit{addknow} &: CS \times [\mathcal{T}_\Sigma(\mathcal{V})] \to CS \\
\mathit{genmsgs} &: CS \times [\mathcal{T}_\Sigma(\mathcal{V})] \to CS \\
\mathit{sub} &: CS \times \mathit{Subst} \to CS \\
\mathit{addineq} &: CS \times \mathit{Ineq} \to CS \\
\mathit{reduce} &: CS \to [CS] \\
\mathit{satisfiable} &: CS \to \mathbb{B} \\
\mathit{getvars} &: CS \to [\mathcal{V}]
\end{aligned}$$

where $[t]$ denotes a finite set of elements of type t, and Subst the set of all substitutions. (Ineq is defined in Definition 6.8)

This defines the syntax of the data-type. Before we formally define a meaning, we briefly give an intuition.

A constraint store represents the state of the intruder (i.e. what messages he knows) and the constraints like what messages he had to generate so far. Therefore the meaning of a constraint store can be expressed as a set of interpretations (ground substitutions) for the variables that occurred in the terms from which the constraint store was constructed, as well as a set of messages that the intruder currently knows. To define this independent of the implementation of the data-type CS, we require that there is a semantics function for whatever formalism is used here that maps a constraint store to an infinite set of pairs of an interpretation and an intruder knowledge (which are ground terms):

$$[\![\cdot]\!] : CS \to \mathbb{P}(\mathit{Subst}, \mathbb{T}(\mathcal{T}_\Sigma)) \quad .$$

The *initial CS* represents the initial constraint store, where the intruder knows nothing but his own name, and no variables are to be interpreted. The *addknow* operation means that the

intruder learns a new set of messages that he can use for future construction of messages. The *genmsgs* operation adds the constraint that the intruder must be able create some messages of the given format from his current knowledge. The *sub* operation is used to apply a substitution to the constraint store. The *addineq* operation adds inequality constraints (as defined in Section 6.3) to the constraint store.

The *reduce* operation performs the constraint reduction, yielding a finite set of simple constraint stores. The *satisfiable* operation yields true iff there is at least one model for all constraints in the store, i.e. if the semantics is not the empty set. In fact, the *reduce* operation is not ultimately necessary, as we can always use the *satisfiable* operation to check if the constraint is satisfiable, however the *reduce* operation gives us control over when to apply reduction, which will be helpful for the strategy of search of the transition system in Section 8.3. Finally, the *getvars* operation is a simple query function that returns the set of variables which have been introduced to the constraint (via *genmsgs*) so far and have not been substituted yet (by a *sub* operation or due to reductions). We will formally require that new variables can only be introduced into the constraint store by the *genmsgs* and the *sub* operation.

Definition 7.2. *The operations on the data-type CS must satisfy the following requirements:*

$$
\begin{array}{rcl}
[\![initial]\!] & = & \{(id, \mathcal{DY}(\emptyset))\} \\
[\![addknow(c, NIK)]\!] & = & \{(\sigma, \mathcal{DY}(IK \cup NIK\sigma)) \mid (\sigma, IK) \in [\![c]\!]\} \\
 & & \quad \textit{if } vars(NIK) \subseteq getvars(c) \\
[\![genmsgs(c, T)]\!] & = & \{(\sigma\tau, IK) \mid (\sigma, IK) \in [\![c]\!] \wedge T\sigma\tau \subseteq \mathcal{DY}(IK) \\
 & & \quad \wedge dom(\tau) = vars(T\sigma) \setminus getvars(c) \wedge ground(\tau)\} \\
[\![sub(c, \tau)]\!] & = & \{(\sigma, IK) \mid (\sigma, IK) \in [\![c]\!] \wedge \sigma \succeq \tau\} \\
 & & \quad \textit{if } dom(\tau) \subseteq getvars(c) \\
[\![addineq(c, \phi)]\!] & = & \{(\sigma, IK) \mid (\sigma, IK) \in [\![c]\!] \wedge \sigma \models \phi\} \\
 & & \quad \textit{if } vars(\phi) \subseteq getvars(c) \\
\bigcup_{c' \in reduce(c)} [\![c']\!] & = & [\![c]\!] \\
satisfiable(c) & \textit{iff} & [\![c]\!] = \emptyset \\
getvars(genmsgs(c, T)) & = & getvars(c) \cup vars(T) \\
getvars(addknow(c, T)) & = & getvars(c) \\
 & & \quad \textit{if } vars(T) = getvars(T) \\
getvars(sub(c, \sigma)) & = & (getvars(c) \setminus dom(\sigma)) \cup vars(ran(\sigma)) \\
 & & \quad \textit{if } dom(\sigma) \subseteq getvars(c) \\
getvars(addineq(c, \phi)) & = & getvars(c) \\
 & & \quad \textit{if } vars(\phi) \subseteq getvars(c) \\
getvars(c') & = & getvars(c) \\
 & & \quad \textit{if } c' \in reduce(c)
\end{array}
$$

Observe that some requirements have additional conditions; if the operations on the data-type are used in a way violating the conditions, it is undefined what shall happen.

7.1 Implementation of the Constraint Store

We now describe how the data-type can be implemented using the reduction algorithm and the satisfiability check for inequality constraints introduced in the previous chapters. We will then show that the requirements of the abstract data-type are met.

Definition 7.3. *We implement a CS as a quadruple (C, ϕ, IK, σ), where C is a well-formed set of constraints, ϕ is an inequality constraint, IK is a set of terms (the current intruder knowledge), and σ records all substitutions of variables that have been performed by constraint reduction so far. We define the semantics function for this implementation of constraint stores as*

$$[\![(C, \phi, IK, \sigma)]\!] = \{(\sigma\tau, \mathcal{DY}(IK\tau)) \mid \tau \in [\![C]\!] \wedge \tau \models \phi\} .$$

The implementation of the operations is as follows:

$$\begin{array}{rcl}
addknow((C,\phi,IK,\sigma),T) & = & (C,\phi,IK \cup T\sigma,\sigma) \\
 & & \quad \textit{if } vars(T) \subseteq vars(C) \\
genmsgs((C,\phi,IK,\sigma),T) & = & (C \wedge from(T\sigma; IK),\phi,IK,\sigma) \\
sub((C,\phi,IK,\sigma),\tau) & = & (C\tau,\phi\tau,IK\tau,\sigma\tau) \\
 & & \quad \textit{if } dom(\tau) \subseteq getvars(c) \\
addineq((C,\phi,IK,\sigma),\psi) & = & (C,\phi \wedge \psi,IK,\sigma) \\
 & & \quad \textit{if } vars(\phi) \subseteq getvars(c) \\
reduce((C,\phi,IK,\sigma),\psi) & = & \{(C',\phi\sigma',IK\sigma' \cup NIK,\sigma\sigma') \mid (C',\sigma') \in Red(C) \\
 & & \quad \wedge\, satisfiable(\phi\sigma') \wedge NIK = \cup_{from(T;IK') \in C'}\, IK'\} \\
satisfiable(c) & \textit{iff} & reduce(c) = \emptyset \\
getvars((C,\phi,IK,\sigma)) & = & vars(C)
\end{array}$$

We now show that this is a valid implementation of the abstract data-type *CS*:

Theorem 7.1. *The implementation of Definition 7.3 satisfies the requirements of Definition 7.2.*

Proof. First note that all operations on *CS* preserve the following invariants:

- $vars(\phi) \cup vars(IK) \cup vars(ran(\sigma)) \subseteq vars(C)$,
- *from* constraints C of the constraint store are well-formed, and for each $from(T; IK') \in C$ it holds that $IK' \subseteq IK$.

We need to prove both invariants simultaneously, because they depend on each other. We consider each operation individually:

- *addknow*: the only change is to the intruder knowledge, and since the new knowledge may not contain new variables, the first invariant is preserved. As the *from* constraints are not touched, they remain well-formed. As the intruder knowledge only increases, the condition that it is a superset of all intruder knowledges in the *from* constraints is preserved.
- *genmsgs*: the first invariant is obvious as only C is augmented with a new constraint. The new *from* constraint has the intruder knowledge IK, so it is a superset of the previous intruder knowledges in C and contains only variables that occur in C. As C is well-formed, the augmented set of *from* constraints is thus also well-formed, and the second invariant is preserved, too.
- *sub*: as the first invariant already holds on the constraints, all variables substituted in ϕ, IK, and $ran(\sigma)$ are present and substituted in C, too. The same relationship holds for the newly introduced variables, i.e. those in $ran(\sigma)$. Thus the first invariant is preserved. The second one remains also true as well-formedness is closed under substitution, as we have already shown in Theorem 6.1.
- *addineq*: only augments the inequalities, which may not contain new variables. Thus the first invariant is preserved. The second one is also as neither intruder knowledge nor *from* constraints are touched.

- *reduce*: is implemented using *sub* which preserves the invariants, provided that the side-condition of *sub* is preserved: the substitution σ' with which *sub* is called results out of the call $Red(C)$. This is guaranteed to substitute only variables of C, thus $dom(\sigma') \subseteq vars(C) = getvars(c)$. Note that this argument also requires that the second invariant holds, namely that C is already well-formed, as otherwise $Red(C)$ is not guaranteed to produce meaningful results. The resulting constraint stores are again well-formed. The augmented intruder knowledge with everything that was derived as new intruder knowledge during constraint reduction.

We can now show that all conditions on the meaning of the constraint store are preserved by the operations. Let $S = [\![c]\!]$ be the meaning of a constraint store c and $c' = op(c', arg)$ the result of some operation on c. We have to show that S and $S' = [\![c']\!]$ are in the relationship required by the semantics of the ADT.

- *addknow*: Here the requirement is that the knowledge of the solutions in S' is $\mathcal{DY}(IK \cup NIK\sigma)$, where IK is the old ground knowledge, NIK the new terms added, and σ a ground substitution. In the implementation, this is performed on the symbolic level (i.e. without ground substitutions) and without Dolev-Yao closure; the semantics maps this correctly to the ground level, taking first the ground substitution for the constraints and using the $\mathcal{DY}$ closure of the ground intruder knowledge.

- *genmsgs*: The requirement is that additionally $T\sigma\tau \subseteq \mathcal{DY}(IK)$ and where τ substitutes the new variables in T. This condition is the meaning, expressed symbolically, of the *from* constraint that is added by this operation.

- *sub*: The requirement says that all substitutions should be excluded that are not instances of τ. As applying τ to the entire constraint C removes all variables of $dom(\tau)$ from C, a different choice cannot be done for these variables anymore.

- *addineq*: The requirement is that only substitutions are allowed that satisfy the inequality. A constraint expressing exactly this is added to the constraint store.

- *reduce*: As the constraint set it well-formed, the correctness of this operation follows from the correctness of *Red*, i.e. by Theorem 6.2: the union of the results is equivalent to the input. Also, we perform every resulting substitution on the entire constraint store. Checking the satisfiability of $\phi\sigma'$ is also not a restriction as otherwise the respective case would have as semantics the empty set and could thus be omitted. Finally, the new intruder knowledge NIK can only contain messages derivable in the respective case using $\mathcal{DY}$.

- *satisfiable*: this follows directly from Theorem 6.3.

- *getvars*: the conditions on the variable handling here reflects how variables are introduced and removed by the respective operations.

□

By this theorem, we can from now on use this ADT and do not need to care about technical details of the implementation of constraint sets such as well-formedness.

Chapter 8

The Symbolic Transition System

We now show how we can integrate the lazy intruder into the transition system that is defined by the intermediate format, using the ADT defined in the previous chapter. The integration into other approaches based on strand spaces or process calculi can be done similarly. However we do not considered this here.

So far, we have considered protocols as infinite-state transition systems where each state is a ground term. With the lazy intruder, we now move to a symbolic approach. The idea behind the symbolic approaches is to use symbolic states (i.e. terms with variables) to represent sets of ground states (i.e. sets of terms without variables). The permissible substitutions for variables are described by a constraint store like the one defined in the previous chapter. It is thus straightforward to extend the semantics of the constraint stores to a semantics of states:

Definition 8.1. *A symbolic state is a tuple (S, c) where S is an IF state (which may contain variables), c is a constraint store, and $vars(S) \subseteq getvars(c)$. We define the semantics of a symbolic state as follows:*

$$[\![(S, c)]\!] = \{S\sigma \mid (\sigma, IK) \in [\![c]\!]\} .$$

For readability, we will use the following convention for variable names in the following: $s, t, \ldots$ denote symbolic states, $S, T, \ldots$ denote sets of facts (i.e. ground states or the first components of symbolic states), and $c, c', \ldots$ for constraint stores (i.e. members of the abstract data-type we have previously defined). We now define a symbolic transition system, i.e. one based on symbolic states, so that the semantics of the reachable symbolic states is equivalent to the set of reachable ground states according to the definition of IF.

Definition 8.2. *Given an IF protocol description (I, R, A) with a (ground) initial state I, we define the* symbolic initial state *(i.e. the initial state of the symbolic transition system) to be*

$$I^{sym} = (I, addknow(initial, \{m \mid \mathsf{iknows}(m) \in I\})) ,$$

i.e. we initialize the constraint store by adding the initial knowledge of the intruder in the initial state of the IF protocol description.

Observe that $[\![I^{sym}]\!] = \{I\}$. To define a transition relation for lazy states, we need an algorithm for subset unification in order to unify the left-hand side of a rule with a symbolic state:

Definition 8.3. *We define the following algorithms for finding a complete set of unifiers for the problem $P\sigma \subseteq S\sigma$, as well as for the the problem $P\sigma \cap S\sigma \neq \emptyset$, where P and S are finite sets of terms. These algorithms are based on the algorithm gunify that extends unification to conjunctions*

and disjunctions of equalities:

$$\begin{aligned}
\mathit{subsetunify}(P,Q) &= \mathit{gunify}(\bigwedge_{p\in P}\bigvee_{q\in Q} p \doteq q)\\
\mathit{intersecunify}(P,Q) &= \mathit{gunify}(\bigvee_{p\in P}\bigvee_{q\in Q} p \doteq q)\\
\mathit{gunify}(\phi) &= \begin{cases} \mathit{unify}(\{(t_1,s_1),\ldots,(t_n,s_n)\}) & \text{if } \phi = t_1 \doteq s_1 \wedge \ldots t_n \doteq s_n\\ \mathit{gunify}(\phi_1)\cup \mathit{gunify}(\phi_2) & \text{if } \phi = \phi_1 \vee \phi_2\\ \mathit{gunify}((\phi_1\wedge\psi_1)\vee(\phi_1\wedge\psi_2)) & \text{if } \phi = \phi_1\wedge(\psi_1\vee\psi_2)\\ \mathit{gunify}((\phi_1\wedge\psi)\vee(\phi_2\wedge\psi)) & \text{if } \phi = (\phi_1\wedge\phi_2)\vee\psi \end{cases}
\end{aligned}$$

Note that all these algorithms are guaranteed to return a finite set of results after finitely many steps, since this is already the case for *unify*. The following lemma tells us that these algorithms are correct:

Lemma 8.1. *Consider a conjunction or disjunction of equalities ϕ, and let $\{\sigma_1,\ldots,\sigma_n\} = \mathit{gunify}(\phi)$. Then it holds that*

$$\{\sigma \mid \sigma \models \phi\} = \{\sigma \mid \exists i \in \{1,\ldots,n\}.\, \sigma \succeq \sigma_i\}\ .$$

Let P and Q be finite sets of terms, and $\{\sigma_1,\ldots,\sigma_n\} = \mathit{subsetunify}(P,Q)$. Then it holds that

$$\{\sigma \mid P\sigma \subseteq Q\sigma\} = \{\sigma \mid \exists i \in \{1,\ldots,n\}.\, \sigma \succeq \sigma_i\}\ .$$

Let $\{\sigma_1,\ldots,\sigma_n\} = \mathit{intersecunify}(P,Q)$. Then it holds that

$$\{\sigma \mid P\sigma \cap Q\sigma \neq \emptyset\} = \{\sigma \mid \exists i \in \{1,\ldots,n\}.\, \sigma \succeq \sigma_i\}\ .$$

Proof. For $\mathit{gunify}(\phi)$, in the base case where ϕ is a conjunction of equalities, the correctness is obvious, as this is the standard unification problem for which *unify* is already a correct algorithm. The case of a disjunction is also obvious. The other two cases when we have a conjunction of disjunctions, we have to use the fact that $\phi\wedge(\psi_1\vee\psi_2) = (\phi\wedge\psi_1)\vee(\phi\wedge\psi_2)$ (and similarly, for the other case). In other words, by "multiplying out" we reduce this to the problem of disjunctions of conjunctions of inequality, which we can already solve.

For *subsetunify* we simply use the fact that $P\sigma \subseteq Q\sigma$ iff $\bigwedge_{p\in P}\bigvee_{q\in Q} p\sigma = q\sigma$. For *intersecunify* we use the fact that $\bigvee_{p\in P}\bigvee_{q\in Q} p\sigma = q\sigma$. □

8.1 The Symbolic Successor Relation

We now define the transition relation for symbolic states in analogy to Definition 4.2:

Definition 8.4. *Let $r = L =[V]\Rightarrow R$ be an IF rule. Let further P be the set of positive facts of L without the* iknows *facts; let M be the set of message terms m such that* iknows$(m) \in L$*; let N be the set of negative facts of L; and let C be the set of conditions of L.*

We also assume that the variables in the symbolic state (S,c) that we consider is disjoint from those in r (which can be achieved by simple renaming in the rule). Then we define that $(S,c) \Rrightarrow_r (T,c_4)$ holds iff there is a substitution σ such that

- $\sigma \in \mathit{subsetunify}(P,S)$
- $E = \mathit{intersecunify}(N,S)$
- $T = (S\sigma \setminus P\sigma) \cup R\sigma\tau$,
- $c_1 = \mathit{sub}(c,\sigma)$,

- $c_2 = genmsgs(c_1, M\sigma)$,
- $c_3 = addineq(c_2, C\sigma \wedge \bigwedge_{\tau \in E} negsub(\tau))$,
- $c_4 = addknow(c_3, \{m \mid \mathsf{iknows}(m) \in R\}\sigma)$.

where τ is an injective substitution with $dom(\tau) = V$ and $ran(\tau)$ is a set of fresh unintrudable constants.[1] *Moreover*

$$negsub([x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]) = x_1 \neq t_1 \vee \ldots \vee x_n \neq t_n \ .$$

For a set of rules R, let $S \Rrightarrow_R T$ be $\bigcup_{r \in R} S \Rrightarrow_r T$. The set of reachable symbolic states *is defined as follows for an initial state I and a set of rules R:*

$$sreach(I, R) = \{(S, c) \mid I^{sym} \Rrightarrow (S, c)\} \ .$$

Observe that we do everything analogously to the ground version except for the following important differences:

- There are variables in states, so we do no longer have matching, but unification, for which we use the algorithms defined in Definition 8.3.
- The negative conditions are not directly checked, but expressed as additional constraints, and the check for satisfiability of the generated constraints is not mandatory. This is because we do not want to prescribe, as part of the formalism, whether or not constraints are directly reduced after every transition. Instead, we leave this decision to the search strategy, discussed in Section 8.3.
- Note that there are finitely many successors: the set of substitutions σ that can result from *subsetunify* is finite, the intruder generation is handled by the constraints.

The intruder knowledge (not closed under $\mathcal{DY}$) is stored both as iknows facts in the states and in the constraint store. It is important for the following that these two accounts are equal:

Lemma 8.2. *For every reachable symbolic state (S, c) and for each $(\sigma, IK) \in [\![c]\!]$, it holds that*

$$IK = \mathcal{DY}(\{m\sigma \mid \mathsf{iknows}(m) \in S\}) \ .$$

Proof. This follows directly from the fact that for every transition, we directly add all new iknows facts to the constraints using *addknow*. □

The next lemma is the central step towards proving that the symbolic transition system and the ground transition system are equivalent. Namely, it tells us that the symbolic transition relation is correct in the following sense: if we first apply the symbolic transition relation to a symbolic state and then the semantics function, then we obtain the same set of ground states as when we first apply the semantics function and then use the ground transition relation:

Lemma 8.3. *For every reachable symbolic state $s = (S, c)$ and transition rule r, it holds that*

$$\bigcup_{t \mid s \Rrightarrow_r t} [\![t]\!] = \bigcup_{S_0 \in [\![s]\!]} \{T_0 \mid S_0 \Rightarrow_r T_0\} \ .$$

[1] Again, we assume that there is a deterministic mechanism to create fresh constants. We will silently assume that in corresponding transitions of the ground and the symbolic transition system, the same fresh constants are created.

Proof. Let again r be a rule transition rule and P, M, N, C, V, and R the positive left-hand side facts (without iknows facts), left-hand side iknows facts, negative facts, inequalities, fresh variables, and the right-hand side facts, respectively. Let τ be a substitution for the fresh variables. Then we have:

$$\bigcup_{t|s\Rightarrow_r t} [\![t]\!] \overset{\text{Def. 8.4}}{=} \bigcup_{t\in A} [\![t]\!]$$
$$\text{where } A = \{(T, c_4) \mid \exists \sigma, E, c_1, \ldots c_3.\ \sigma \in \mathit{subsetunify}(P, S) \wedge$$
$$E = \mathit{intersecunify}(N, S) \ \wedge\ T = (S\sigma \setminus P\sigma) \cup R\sigma\tau \wedge$$
$$c_1 = \mathit{sub}(c, \sigma) \ \wedge\ c_2 = \mathit{genmsgs}(c_1, M\sigma) \wedge$$
$$c_3 = \mathit{addineq}(c_2, C\sigma \wedge \textstyle\bigwedge_{\tau\in E} \mathit{negsub}(\tau)) \wedge$$
$$c_4 = \mathit{addknow}(c_3, \{m \mid \mathsf{iknows}(m) \in R\}\sigma)\}$$

$$\overset{\text{Lem. 8.1}}{=} \bigcup_{t\in A} [\![t]\!]$$
$$\text{where } A = \{(T, c_4) \mid \exists \sigma, c_1, \ldots c_3.\ P\sigma \subseteq S\sigma \wedge$$
$$N\sigma \cap S\sigma \neq \emptyset \ \wedge\ T = (S\sigma \setminus P\sigma) \cup R\sigma\tau \wedge$$
$$c_1 = \mathit{sub}(c, \sigma) \ \wedge\ c_2 = \mathit{genmsgs}(c_1, M\sigma) \wedge$$
$$c_3 = \mathit{addineq}(c_2, C\sigma) \wedge$$
$$c_4 = \mathit{addknow}(c_3, \{m \mid \mathsf{iknows}(m) \in R\}\sigma)\}$$

$$\overset{\text{Def. 7.2}}{=} \{T \mid \exists \sigma.\ \mathit{ground}(\sigma) \wedge \mathit{dom}(\sigma) = \mathit{vars}(S) \cup \mathit{getvars}(c) \cup \mathit{vars}(P) \wedge$$
$$P\sigma \subseteq S\sigma \wedge N\sigma \cap S\sigma \neq \emptyset \ \wedge\ T = (S\sigma \setminus P\sigma) \cup R\sigma\tau \wedge$$
$$M\sigma \subseteq \mathcal{DY}(\{m\sigma \mid \mathsf{iknows}(m) \in S\}) \wedge \sigma \models C\}$$

$$\overset{\substack{\mathit{vars}(s)\cap \\ \mathit{vars}(r)=\emptyset}}{=} \{T \mid \exists \chi\sigma.\ \mathit{ground}(\sigma\chi) \wedge \mathit{dom}(\chi) = \mathit{vars}(S) \cup \mathit{getvars}(c) \wedge \mathit{dom}(\sigma) = \mathit{vars}(P) \wedge$$
$$P\sigma \subseteq S\chi \wedge N\sigma \cap S\chi \neq \emptyset \ \wedge\ T = (S\chi \setminus P\sigma) \cup R\sigma\tau \wedge$$
$$M\sigma \subseteq \mathcal{DY}(\{m\chi \mid \mathsf{iknows}(m) \in S\}) \wedge \sigma\chi \models C\}$$

$$\overset{\text{Def. 8.4}}{=} \bigcup_{S_0\in[\![s]\!]} \{T \mid \exists \sigma.\ \mathit{ground}(\sigma) \wedge \mathit{dom}(\sigma) = \mathit{vars}(P) \wedge$$
$$P\sigma \subseteq S_0 \wedge N\sigma \cap S_0 \neq \emptyset \ \wedge\ T = (S_0 \setminus P\sigma) \cup R\sigma\tau \wedge$$
$$M\sigma \subseteq \mathcal{DY}(\{m \mid \mathsf{iknows}(m) \in S_0\}) \wedge \sigma \models C\}$$

$$\overset{\text{Def. 4.2}}{=} \bigcup_{S_0\in[\![s]\!]} \{T_0 \mid S_0 \Rightarrow_r T_0\}$$

Observe that the transformation labeled "Def. 7.2" refers to the properties of the abstract data-type CS, and its correct implementation is based on the correctness of the lazy intruder technique (Theorem 6.2) and its correct extension with inequalities (Theorem 6.3). □

We can now conclude that the set of reachable symbolic states represents exactly the set of reachable states of the ground model:

Lemma 8.4. *Let I be an IF initial state and R a set of IF transition rules. It holds that*

$$\bigcup_{s|I^{sym}\Rightarrow_R^* s} [\![s]\!] = \{S \mid I \Rightarrow_R^* S\}\ .$$

Proof. This is shown by induction over the number of applications of the two successor relations. For the induction basis, observe that $[\![S_0^{sym}]\!] = \{S_0\}$. For the induction step, consider a set of symbolic states S' reached after n applications of $\Rightarrow_R$, and the set of ground states S reached after n applications of $\Rightarrow_R$. The induction hypothesis is $\bigcup_{s\in S'} [\![S']\!] = S$. By Lemma 8.3, the set of symbolic states T' reached in $n+1$ transitions of any rule have the same semantics as the

ground successors of the states in S (the semantics of the S' states). So the set T of ground states reached in $n+1$ steps is equal to the semantics of T', which proves the statement. □

8.2 Symbolic Attack States

We now come to the attack states for the symbolic transition system. As in the ground transition system, we consider the IF attack rules as rules without a right-hand side, and that we extend to IF rules that produce the attack symbol on the transition. The symbolic attack states are thus those states that contain the attack fact, but we additionally require that the constraints in an symbolic attack state must be satisfiable:

Definition 8.5. *The set of* symbolic attack states*, denoted SAttackState, is the set of all symbolic states* (S, c) *such that* $\mathsf{attack} \in S$ *and satisfiable*(c)*. An IF protocol description* (I, R, A) *is called* safe in the symbolic transition system *iff*

$$\mathit{sreach}(I, R \cup \overline{A}) \cap \mathit{SAttackState} = \emptyset \ ,$$

where again $\overline{A}$ *is the set of rules* $a \Rightarrow a.\mathsf{attack}$ *for each* $a \in A$*.*

We can now show that the ground and symbolic models coincide in the sense that the same protocols are safe:

Theorem 8.1. *A protocol description* (I, R, A) *is safe iff it is safe in the symbolic transition system.*

Proof. First observe that for every symbolic state (S, c) it holds that

$$\exists S_0 \in [\![(S, c)]\!].\ \mathsf{attack} \in S_0 \text{ iff } \mathsf{attack} \in S \wedge [\![(S, C)]\!] = \emptyset \ .$$

This is because facts cannot be variables themselves (i.e. they must have the form $f(t_1, \ldots, t_n)$ for some $f \in \mathcal{F}$ and terms $t_i \in \mathcal{T}_\Sigma(\mathcal{V})$, and thus if $\mathsf{attack} \in S_0$ for some $S_0 \in [\![(S, c)]\!]$, then follows $\mathsf{attack} \in S$. Vice-versa, if $\mathsf{attack} \in S$, then also $\mathsf{attack} \in S_0$ for all $S_0 \in [\![(S, c)]\!]$.

We conclude that a symbolic state is a symbolic attack state iff its semantics contains attack states. By Lemma 8.4, the semantics of all symbolic reachable states is the set of reachable states of the ground transition system. For the set of rules $R \cup \overline{A}$ where R are the transition rules and A the attack rules, we thus have that there is a reachable symbolic attack state iff the ground reachable states contain an attack state. □

8.3 Organizing Symbolic State Exploration

To summarize, the previous sections provide us with a symbolic method for checking security protocols. We start with a symbolic initial state and use the constraint store data-type to define a transition relation on symbolic states. Attack states in this symbolic transition system are identified by the fact symbol attack and calling the satisfiability check of the constraint store data-type. As this data-type is an abstract interface to the constraint reduction, the "higher layers" of the search algorithm (i.e. construction and exploration of the transition system) do not have to deal with any low-level technicalities (like well-formedness of constraints).

A central feature of this approach is that the successor relation of the symbolic transition system is finite, while the successor relation of the ground transition system is almost always infinite as a result of the Dolev-Yao model. This finiteness is achieved without performing any restrictions. The infinite branching of the ground transition system was caused by the intruder deduction relation that in most cases allows the intruder to say infinitely many different messages that an agent could receive. The symbolic technique of the lazy intruder thus allows us to tackle one of the aspects of infinity in the protocol analysis problem without performing any restrictions. To illustrate this, consider a protocol description (I, R, A) where $\Rightarrow_R$ terminates, i.e. where we

have a bounded number of sessions. Then, the set of symbolic reachable states is finite, while the ground transition system may still have infinitely many different states. A related result of [121] is that the insafety problem for a bounded number of sessions (but without a bound on the intruder) is NP-complete.

Besides having a terminating algorithm for dealing with the infinitely many possible messages the intruder can construct, there is another aspect of the lazy intruder technique that is of much greater practical relevance: even if for some reason we can bound the set of messages that the intruder can say at any point, the branching is often enormous. This is the reason why many forward exploration approaches fail even on fairly simple protocols. The symbolic approach, in contrast, summarizes many transitions into one, often resulting in a state-space that is not much larger than the ground transition system without the intruder (which would not be very meaningful to explore, of course).

Observe, however, that sometimes the symbolic transition system permits transitions that would have no counter-part in the ground transition system, namely when the semantics of a reached symbolic state is the empty set, because the constraints are unsatisfiable. We can avoid this by checking the satisfiability of the constraints after every transition; if the semantics of a symbolic state is the empty set, then this state and all its successors can be dropped from the state-space without changing the semantics: obviously the semantics of all successor states is the empty set, too, according to Lemma 8.4.

We can regard this as two possible strategies: checking satisfiability of the constraints of every reachable state and checking satisfiability only for attack states. We call them the *eager* and the *lazy strategy to check satisfiability*, respectively. The lazy one has the disadvantage that we often explore large parts of the search space unnecessarily, because the constraints are unsatisfiable. The disadvantage of the eager strategy is that we often consider the same reductions several times, since the successors of a symbolic state have as constraints a superset of the constraints of their predecessor.

It is thus desirable that the satisfiability check does not enumerate all of the many possible solutions to a constraint store (in terms of simple constraints and substitutions), but rather stops the reduction as soon as the first solution is found. While it is quite "technical" to formulate this in many programming languages, Haskell's lazy evaluation allows us to formulate the reduction algorithm such that it returns a lazy list of solutions. When checking this list for emptiness, we may stop the reduction process as soon as the first solution is found. Thus, further solutions are "automatically not computed".

Yet a further improvement results out of the following observation. If C is a well-formed constraint set and $C \cup C'$ is an also well-formed extension of C (as it is the case in a successor states of a state with constraint store C), then we can exploit the following property:

$$[\![C \cup C']\!] = \bigcup_{(C_i,\sigma_i)\in Red(C)} \{\sigma_i\sigma' \mid \sigma' \in [\![C_i \cup C'\sigma_i]\!]\} \ ,$$

which follows from the fact that *Red* is correct. This tells us that, if we reduce C first, then we can add further constraints to the solutions of $Red(C)$, and continue the computation on them, instead of reducing $C_1 \cup C_2$. Thus, if we have obtained a lazy list $Red(C)$ in the satisfiability check, then we can re-use this result straightforwardly to reduce additional constraints in a successor state. The lazy evaluation of Haskell thus allows us to declaratively formulate an optimal compromise between lazy and eager strategy to check satisfiability—never missing unsatisfiable constraints, and never re-computing solutions that have already been computed.

8.4 Symbolic Sessions

In practice it is often advantageous to organize and control search by considering (and searching) different scenarios under which a protocol should be checked, corresponding to different concrete sessions.

A straightforward way to specify a bounded number of sessions is shown in Example 4.2 for the Yahalom protocol (although we have left unbounded the server in this example): We simply specify as the initial state a set of state facts, each representing one agent who is prepared to communicate. While this is a common restriction of protocol specification languages, e.g. in HLPSL [48] and CAPSL/CIL [70], this manual specification of sessions by the user is unsatisfactory. In practice, to avoid the state-space explosion due to parallelism, it is desirable to iterate through many scenarios, with differing role instances. Support for searching among instances is thus called for here.

[39] proposes an alternative to manual session specification by automatically generating all session instantiations for a given number of sessions (up to isomorphism, i.e. sessions that are identical up to a renaming of the constants used for honest agent names). Note that the number of honest agents can be bounded by the number of roles multiplied with the number of sessions. This produces thus a number of initial states and therefore analysis problems that can be checked individually. However, even for a small number of sessions, the number of generated initial states is enormous.

One can interpret automatic session generation as a kind of parameter search: the protocol model is parameterized over a scenario and we can explore the values of this parameter (for a given upper bound). The idea is that we can use the lazy intruder technique to solve this parameter search problem, rather than separating it from the protocol analysis. This is possible as there is no essential difference between choosing an agent from a limited set of possibilities when generating sessions, and using the lazy intruder to choose a message from his knowledge during the normal search.

More precisely, we take advantage of the symbolic representation of the lazy intruder to avoid enumerating all possible session instances for a given bound on the number of sessions. To do this, we instantiate the roles with variables rather than with constant agent names. The variables are then instantiated *by unification during search* (rather than before search), either to constant agent names or to other variables.

For the example of the Yahalom protocol and 3 sessions, we can specify the following *symbolic* facts in the initial state:

$$\begin{array}{l}\mathsf{state}_{\mathcal{A}}(0,\mathsf{sess1},\mathsf{A}_1,\mathsf{B}_1).\mathsf{state}_{\mathcal{B}}(0,\mathsf{sess1},\mathsf{B}'_1).\\ \mathsf{state}_{\mathcal{A}}(0,\mathsf{sess2},\mathsf{A}_2,\mathsf{B}_2).\mathsf{state}_{\mathcal{B}}(0,\mathsf{sess2},\mathsf{B}'_2).\\ \mathsf{state}_{\mathcal{B}}(0,\mathsf{sess3},\mathsf{A}_3,\mathsf{B}_3).\mathsf{state}_{\mathcal{B}}(0,\mathsf{sess2},\mathsf{B}'_3)\end{array}$$

along with the inequality constraint

$$\mathsf{A}_1 \neq \mathsf{i} \wedge \mathsf{A}_2 \neq \mathsf{i} \wedge \mathsf{A}_3 \neq \mathsf{i} \wedge \mathsf{B}'_1 \neq \mathsf{i} \wedge \mathsf{B}'_2 \neq \mathsf{i} \wedge \mathsf{B}'_3 \neq \mathsf{i}\,.$$

Note that we have used different variables like B_1 and B'_1 even in the different state facts that belong to the same session. The reason is that actually the state facts of one session have no real relationship in the semantics, i.e. $\mathcal{A}$ might be thinking to talk to somebody else in role $\mathcal{B}$ than it actually does. The inequality constraint expresses that the agent who is represented by a state fact is not the intruder; this is not a restriction, as the intruder does not necessarily stick to the protocol and is thus not represented by state facts and transition rules, but by the $\mathcal{DY}$-closure of his knowledge.

We have so far not defined what an initial state with symbolic facts is supposed to mean, and the IF initial state is ground by definition. (Note that the symbolic initial state by definition is also a ground term with an empty constraint store.) We therefore now extend the definitions of syntax and semantics of IF:

Definition 8.6. *We extend the IF syntax to allow* symbolic session initial states *by removing the side-condition that the initial state may not contain variables. Moreover, the initial state may contain an inequality constraint (in the syntax of Ineq) with the condition that the variables of the inequality constraint are a subset of the state term.*

Given a finite set of agents $\mathsf{Agent} \subset \Sigma_0$ *with* $\mathsf{i} \in \mathsf{Agent}$*, we define the semantics of a symbolic session initial state* (s,ϕ)*, where* s *is the set of facts and* ϕ *is the inequality constraint, as follows:*

$$[\![(s,\phi)]\!]_{\mathsf{Agent}} = \{s\sigma \cup \{\mathsf{iknows}(a)\} \mid a \in \mathsf{Agent}\} \mid dom(\sigma) = vars(s) \wedge ran(\sigma) \subseteq \mathsf{Agent} \wedge \sigma \models \phi\}$$

The set of reachable states for a symbolic session initial state (s, ϕ) *is defined as the union of the reachable states for each ground initial state in* $[\![(s, \phi)]\!]$. *Other notions like* safety of a protocol description *are extended accordingly.*

This definition actually mirrors what automated session generation as discussed above may do: instantiate all variables in the initial state with any value of the agent names in the set Agent, and respecting the restrictions imposed by the inequality constraints.[2] Moreover, we have required that all agent names are added to the initial intruder knowledge, which we discuss below.

We now show how this extended problem can be solved by the symbolic lazy intruder approach without enumerating all ground initial states. The only thing that prevents us from directly using the symbolic session initial state as the symbolic initial state of our symbolic transition system is the following. In order for the lazy intruder technique to be complete, all constraint sets need to be well-formed. In particular, each variable that appears in a constraint set must be introduced on the left-hand side of some constraint first. To speak in terms of the constraint store data-type, we cannot add knowledge (in this case, messages containing agent names) without requiring the intruder to generate a message containing these variables first.

Intuitively speaking, combining session instantiation with the lazy intruder means that the intruder chooses the concrete names of the agents in the initial state, but leaves variables for these choices and lazily instantiates them during the search. The initial state must therefore be associated with a constraint store that requires the lazy intruder to "generate" all agent names from his initial intruder knowledge IK_0. We do exactly this in the following modification of the definition of the symbolic initial state:

Definition 8.7. *For a symbolic session initial state* (s, ϕ)*, we redefine the initial state of the symbolic transition system to be*

$$(s, \phi)^{sym} = (s \cup \{\mathsf{iknows}(a)\} \mid a \in \mathsf{Agent}\}, \mathit{addineq}(\mathit{genmsgs}(c_0, \mathit{vars}(s)), \phi)) \ ,$$

where $c_0 = \mathit{addknow}(\mathit{initial}, \{m \mid \mathsf{iknows}(m) \in s\} \cup \mathsf{Agent})$. *All depending concepts like* reachable states *are modified accordingly.*

We have thus added the set of all agent names to the initial intruder knowledge and moreover required that all variables of the initial state are generated by the intruder. The reason that we give the intruder the name of all agents is that we want him to choose these values and he cannot do that unless he knows them. This means a kind of restriction: we cannot reasonably check for the secrecy of agent names, or perform a meaningful analysis of protocols in which the safety depends on the assumption that the intruder cannot find out the agent names.

There are obviously more solutions for the constraints than we want: for the agent names the intruder may now pick any term that he can generate from his initial knowledge (not only constants from Agent), so there is a chance of false positives (i.e. unrealistic attack detected), e.g. the intruder "generates" an agent name like $\mathsf{i}, \mathsf{i}, \mathsf{i}$. In experiments we never had such false positives; a way to systematically exclude them is using a (partially) typed model.

We now formally show that this new symbolic initial states represents a superset of the symbolic session initial state:

Theorem 8.2. *For each symbolic session initial state it holds that*

$$[\![(s, \phi)^{sym}]\!] \supseteq [\![(s, \phi)]\!] \ .$$

Proof. The semantics of the constraint

$$c = \mathit{addineq}(\mathit{genmsgs}(\mathit{addknow}(\mathit{initial}, \{m \mid \mathsf{iknows}(m) \in s\} \cup \mathsf{Agent}), \mathit{vars}(s)), \phi)$$

[2] As we have required the set Agent to be finite (which is not a restriction if we have finitely many sessions), we can explicitly perform this "unfolding" step for the symbolic session initial state and check the protocol for each ground initial state generated this way.

is by definition the set $\{(\sigma, IK_0) \mid dom(\sigma) = vars(s) \wedge vars(s)\sigma \subseteq \mathcal{DY}(IK_0) \wedge \sigma \models \phi\}$ for $IK_0 = \{m \mid \mathsf{iknows}(m) \in s\} \cup \mathsf{Agent}$. As $\mathsf{Agent} \subseteq \mathcal{DY}(IK_0)$, we have

$$[\![(s \cup \{\mathsf{iknows}(a) \mid a \in \mathsf{Agent}, c\})]\!] \subseteq [\![(s, \phi)]\!] .$$

□

With Theorem 8.2, it is straightforward to integrate the symbolic session approach with the lazy intruder technique, as we have a valid symbolic initial state for the search.

Example 8.1. To illustrate session instantiation at work, we now consider again the example NSPK (see Figure 4.4), this time for the following symbolic session initial state:

$$\mathsf{state}_{\mathcal{A}}(0, \mathsf{sess1}, \mathsf{A}, \mathsf{B}).\mathsf{state}_{\mathcal{B}}(0, \mathsf{sess1}, \mathsf{B}', \mathsf{A}') \wedge \mathsf{A} \neq \mathsf{i} \wedge \mathsf{B}' \neq \mathsf{i} .$$

The attack-trace starts with the agent A sending a message for B, encrypted with B's public key. The intruder can analyze this message iff $\mathsf{B} = \mathsf{i}$; hence, we have a case split, i.e. one state where we perform the substitution $\mathsf{B} = \mathsf{i}$ and one where we have the additional inequality $\mathsf{B} \neq \mathsf{i}$. The attack corresponds to the first case, where the substitution is performed. In this case, the intruder learns the nonce n_1 that A has created for him. In the next step, the intruder sends a message to B', posing as A'. B' responds with a message encrypted with the public key of A', and again the intruder can decrypt that message iff $\mathsf{A}' = \mathsf{i}$. Consider the case $\mathsf{A}' \neq \mathsf{i}$. In this case, the intruder chooses to send (under his real name) an answer to A's first message. A expects this message to be encrypted with her public key and to contain the nonce that she sent earlier to i, i.e. we have the constraint set

$$\begin{array}{ll} \{ & \mathit{from}(\{\mathsf{A}, \mathsf{A}', \mathsf{B}'\}; IK_0) , \\ & \mathit{from}(\{\mathsf{NA}\}; IK_0 \cup \{\mathsf{n}_1\}) , \\ & \mathit{from}(\{\{\mathsf{n}_1, \mathsf{NB}\}_{\mathsf{pk(A)}}\}; IK_0 \cup \{\mathsf{n}_1, \{\mathsf{NA}, \mathsf{n}_2\}_{\mathsf{pk(A')}}\}) \ \} \end{array}$$

and the inequalities

$$\mathsf{A} \neq \mathsf{i} \wedge \mathsf{B}' \neq \mathsf{i} .$$

Here NA is a variable representing the nonce that the intruder sent earlier to B', n_1 is the nonce that A has created for i, n_2 is the nonce that B' has created for A', and NB is a variable for the value that the intruder sends to A as his nonce.

The constraint reduction will identify two possible solutions for this constraint set: either the intruder generated the last message $\{\mathsf{n}_1, \mathsf{NB}\}_{\mathsf{pk(A)}}$ from its components or he replays the message he just received from B', since it is unifiable using the substitution

$$[\, \mathsf{A}' \mapsto \mathsf{A} \,,\ \mathsf{NA} \mapsto \mathsf{n}_1 \,,\ \mathsf{NB} \mapsto \mathsf{n}_2 \,] .$$

Hence, the constraint reduction has inferred that the considered trace works if B' thinks he is talking to $\mathsf{A}' = \mathsf{A}$ and A thinks she is talking to i. The rest of the attack is straightforward: A sends the final message of the protocol $\{\mathsf{n}_2\}_{\mathsf{pk(i)}}$, so the intruder knows the nonce that B' has generated for $\mathsf{A}' \neq \mathsf{i}$, which is a violation of secrecy. □

This example demonstrates how, during search, the space of possible instances is narrowed down in a demand-driven fashion. Observe that during the example derivation, variables of type Agent were instantiated with other variables or with the constant i. In general, the A_j's can be instantiated only with those constant agent names that appear in the initial state or in some transition rule, and this is usually just the name of the intruder i. This bears some similarity to the phenomenon observed in [57] that "two agents are sufficient"—an honest agent and a dishonest one. This result requires, however, that there are no negative facts or conditions[3] like in IF protocol descriptions. Another difference is that, while [57] implies that all honest agents have the same name, the symbolic session technique instantiates agent names lazily; therefore, in an attack trace, all agents who may be different are represented by different variables or constants.

This concludes our exposition of the symbolic transition system that integrates the lazy intruder into the protocol analysis for IF protocol descriptions.

[3]There is one exception: one may require that all agents within the same session have different names; then the method requires $k + 1$ agents.

Chapter 9

Constraint Differentiation

As pointed out before, even when bounding both the depth of message terms that the intruder can create and the number of sessions, the search spaces associated to security protocols are still large (though finite) due to two kinds of *state explosions*. The first kind of explosion is caused by the standard Dolev-Yao intruder model. The second kind of explosion stems from the large number of possible interleavings resulting from parallel executions of a protocol by the honest principals and the intruder. The lazy intruder technique is an effective way to tackle the first kind of state explosion, and one that does not even require any bounds on the capabilities of the intruder. This technique is therefore used in a large variety of tools [6, 28, 37, 38, 52, 53, 60, 77, 91, 111].

What can be said about the second kind of explosion? To better understand this problem, it is helpful to distinguish between two different kinds of search spaces. The first, depicted on the right-hand side of Figure 9.1, represents search in the space of constraints to find solutions to a given constraint problem. The different realizations of the lazy intruder technique each provide their own notion of constrained state and constraint reduction rules, which define the transitions in this state space. The second state space is the space of symbolic states, which is searched to determine if a state representing an attack (i.e., a violation of a security property) is reachable. Transitions here typically correspond to actions such as principals communicating with each other. Note that these state spaces are layered: the second search space, depicted on the left-hand side of the Figure 9.1 builds upon the first as each symbolic state has an associated constraint set, and determining which variable substitutions are possible requires solving the associated constraint. As the picture suggests, the first kind of search is performed in a backward fashion ("Can the terms to be generated be reduced to terms the intruder knows?"), while the second kind is performed in a forward fashion ("Which states are reachable from the initial state?").

In standard model-checking approaches for concurrent systems, the interleaving problem is often handled using *partial-order reduction* (POR), which reduces the number of interleavings that need to be considered by exploiting independencies between the possible transitions [117]. One might expect that the direct combination of the lazy intruder with partial-order reduction as part of the second layer of the search (i.e. while searching the symbolic state space) would allow us to simultaneously reduce both kinds of combinatorial explosion. However, as we will explain below, this combination is not effective: the different transitions of the lazy intruder rarely lead to the same successor state, and therefore there is practically no independence of transitions that can be exploited by partial-order reduction.

We show now how to effectively integrate the lazy intruder and ideas from partial-order reduction by using independence information from the second layer, i.e. the symbolic transition system, when searching the first layer, i.e. the constraint reduction. We call the resulting technique *constraint differentiation*.

Figure 9.2 illustrates the intuition behind constraint differentiation. Consider two symbolic states s' and s'' that can be reached from some state s by different interleavings of the same actions, e.g. message exchanges, as represented by the wavy arrows. In practice, there is often a substantial overlap between the sets of ground states represented by s' and s''. Constraint

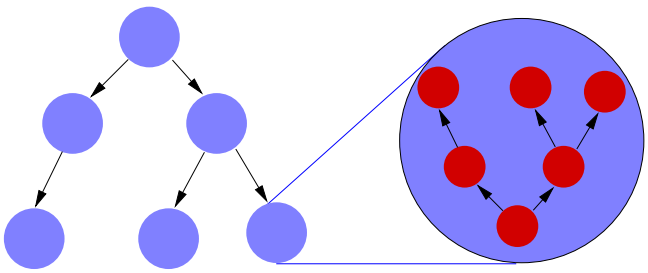


Figure 9.1: Layered search spaces: symbolic states (left) and constraint sets (right)

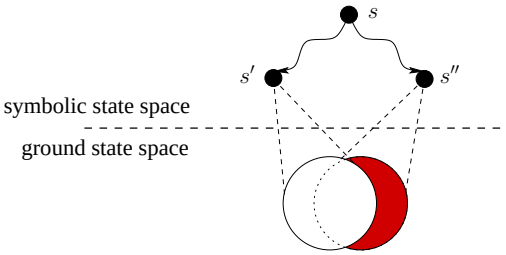


Figure 9.2: The intuition behind constraint differentiation.

differentiation exploits these overlaps by restricting the constraints of s'' to those ground states that are not already covered by s', i.e. the shaded part in Figure 9.2. (Symmetrically, we could restrict s' instead of s''.)

Like for the lazy intruder technique, we formally prove that constraint differentiation is correct in the sense that it does neither exclude any attacks from the transition system (so it is complete) nor introduces new ones (so it is sound). To illustrate the practical relevance of constraint differentiation, we present experimental results showing the improvement on the performance of our tool OFMC that is achieved by constraint differentiation. In particular, this improvement increases the number of sessions for which the verification of complex protocols like IKE is feasible. Thus extends the scope of problems that can effectively be tackled.

9.1 A Common POR Situation

The most common form of IF rules is the following:

$$\mathsf{state}.(\cdot).\mathsf{iknows}(\cdot) \Rightarrow \mathsf{state}.(\cdot).\mathsf{iknows}(\cdot).\ldots\ ,$$

i.e. a transition rule of an honest agent who waits for a particular form of message, sends a reply and updates his local state (and the dots on the right-hand side indicate that there may be additional facts like $\mathsf{secret}(\cdot,\cdot)$ may be created). In particular, note that there are no negative conditions, additional requirements and the like. We will focus on the situation that, in a symbolic state s, at least two transitions are possible. Let us consider the case that there are different state facts matched by the left-hand sides of the transition rules. Then in each of the two successor states of the transitions, the other transition is still possible. This is depicted in Figure 9.3, and we now discuss it in detail.

This is very similar to the situation in model-checking where partial-order reduction (POR) is usually applied, namely when two—in some sense *independent*—transitions can be applied in different orders, leading to—in some sense equivalent—successor states (s_2 and s_4 in the Figure). We shall now see why this is not directly possible in the context of the lazy intruder.

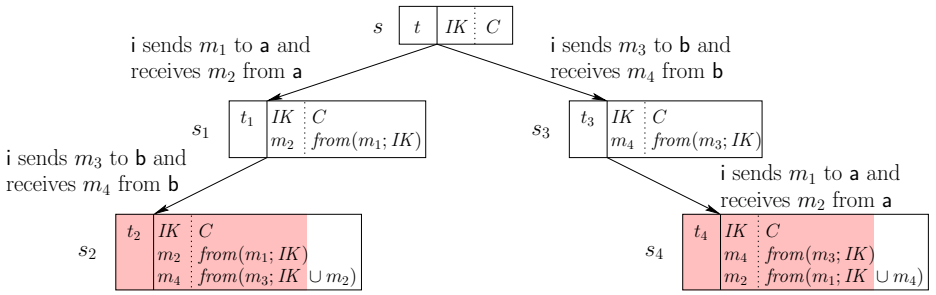


Figure 9.3: An illustration of constraint differentiation for $t_2 = t_4$ (for each symbolic state s we display its term t as well as the intruder knowledge and the *from* constraints as generated by the constraint store data-type).

In the left sequence of transitions, the intruder i first sends a message m_1 to an agent a, receiving the answer m_2, and afterwards he sends a message m_3 to an agent b, receiving the answer m_4. In the right sequence, the intruder first talks to b and then to a. The transitions result in the states s_2 and s_4. Note that $t_2 = t_4$, i.e. the symbolic states s_2 and s_4 contain the same set of symbolic facts. For clarity, we have depicted the actual *from* constraints and symbolic intruder knowledge (without $\mathcal{DY}$-closure) that arise from using the constraint store interface. In this case, for every substitution σ, the represented ground states $t_2\sigma$ and $t_4\sigma$ are the same; however, the constraints, determining the set of permissible substitutions, are different due to the fact that the intruder generated the respective messages at different states of his knowledge. Thus, the symbolic states s_2 and s_4 represent different sets of ground states, and we cannot apply partial-order reduction to eliminate one of the two interleavings.[1]

The observation that leads to a reduction is that in this situation there is an overlapping of the set of ground states, which is represented by the two symbolic states s_2 and s_4, as shown by the shaded part in Figure 9.3: all those ground states in the semantics of the symbolic states that do not exploit the new knowledge m_2 or m_4 are covered by the other symbolic state. The idea is that we can use independence of transitions by exploiting precisely this overlap. If, for example, we "prefer" the left sequence, then for the state s_4 reached by the other sequence we will only be interested in solutions that are not subsumed by s_2 already, i.e. those where the intruder actually uses the message m_4 that he learned in the first transition to generate the message m_1 in the second transition.

In this way, we can propagate information about independencies obtained in the second search layer, the symbolic transition system, to the first layer, the constraint reduction. We exploit the fact that we only need to consider solutions for a given constraint set that are obtained by using new knowledge. In the example, we could express the fact that the message m_4 needs to be used when creating m_1 by using constraints of the form $D\text{-}from(\{m_1\}; IK; \{m_4\})$, which intuitively has the same meaning as the constraint $from(\{m_1\}; IK \cup \{m_4\})$, except that we exclude all solutions satisfying $from(\{m_1\}; IK)$.

Mirroring the development of the lazy intruder, we proceed as follows: we introduce constraint differentiation by defining this new kind of constraint and the associated reduction algorithm. We then show that this algorithm has properties analogous to those of the *Red* algorithm, namely it terminates and is correct. We conclude the chapter by integrating the new constraint reduction into the second layer of the search as a transformation of the search tree induced by the symbolic transition system.

[1] One may wonder whether without step compression, i.e. without this property of the constraints, POR could be more effective. One can show that directly applying POR to a symbolic transition system without step compression can only result in reductions that are also achieved using step compression.

9.2 Constraint Reduction with Constraint Differentiation

Definition 9.1. *A D-from* constraint c *has the form*

$$\textit{D-from}(T; IK; NIK)\,,$$

where T, IK, and NIK are sets of messages. Its semantics and its generate-only semantics are

$$\begin{aligned} [\![c]\!] &= [\![\lceil c \rceil]\!] \setminus [\![\lfloor c \rfloor]\!] \\ [\![c]\!]^{gen} &= [\![\lceil c \rceil]\!]^{gen} \setminus [\![\lfloor c \rfloor]\!]^{gen}\,, \end{aligned}$$

where

$$\begin{aligned} \lceil \textit{D-from}(T; IK; NIK) \rceil &= \textit{from}(T; IK \cup NIK) \\ \lfloor \textit{D-from}(T; IK; NIK) \rfloor &= \textit{from}(T; IK) \end{aligned}$$

are functions mapping from constraints to D-from constraints. We say that D-from$(T; IK; NIK)$ *is* simple *if $T \subseteq \mathcal{V}$ and $T \neq \emptyset$.*

Extending Definition 6.1, we now write simply constraint *to refer to a from constraint or a D-from constraint. As before, all terms are extended straightforwardly to sets of constraints. As shorthand, we use the terms T-*part*, IK-*part*, and NIK-*part *to refer to the respective parts of a D-from constraint* D-from$(T; IK; NIK)$.

Note that all extensions of definitions and algorithms in this chapter are conservative in the sense that, for sets of *from* constraints (without the new *D-from* constraints), the extended definitions and algorithms are equivalent to the definitions and algorithms given in Chapter 6.

Intuitively, *NIK* represents new messages that are not in *IK* (it is an acronym for *new intruder knowledge*). As we will use them, *NIK* and *IK* will be disjoint, so that the constraint *D-from*$(T; IK; NIK)$ states that the set of terms T must be generated by the intruder using the knowledge in the set $IK \cup NIK$, but we are only interested in solutions that employ new information in *NIK*; hence, we exclude all solutions of *from*$(T; IK)$. The function $\lceil \cdot \rceil$ yields a *from* constraint set by removing the requirement on new knowledge; therefore $[\![C]\!] \subseteq [\![\lceil C \rceil]\!]$. Similarly, $[\![\lfloor C \rfloor]\!] \cup [\![C]\!] = [\![\lceil C \rceil]\!]$, as $\lfloor \cdot \rfloor$ returns the solutions removed from $\lceil C \rceil$. Note that for a simple constraint set C both $\lceil C \rceil$ and $\lfloor C \rfloor$ are simple; thus for a simple constraint set C, $\lceil C \rceil$ is satisfiable. However, unlike for *from* constraints, it does not always hold that every simple *D-from* constraint set C is satisfiable (although this is usually the case). Since constraint differentiation is used only for optimization, solutions that satisfy $[\![\lceil C \rceil]\!]$ but not $[\![C]\!]$ are not a problem of soundness, but only of efficiency. Also note that *D-from*$(\emptyset; IK; NIK)$ is unsatisfiable (while *from*$(\emptyset; IK \cup NIK)$ is satisfiable), therefore, in the definition of simple *D-from* constraints, we have excluded the case that the set of terms T to be generated is empty.

The well-formedness is extended as follows:

Definition 9.2. *A set C of from and D-from constraints is called* well-formed*, if $\lceil C \rceil$ is well-formed and for every* D-from$(T; IK; NIK) \in C$*, there is a set T' with* from$(T'; IK) \in \lceil C \rceil$*. The order on the constraints of C induced by the well-formedness is the one induced by the well-formedness of $\lceil C \rceil$.*

This definition ensures that the differentiation of the intruder knowledge into old knowledge *IK* and new knowledge *NIK* is always "based" on an earlier *from* constraint with knowledge *IK* (or a *D-from* constraint where the union of both knowledge parts is *IK*). This means that in *D-from* constraint exactly those messages are new that the intruder has learned since the "last" constraint.

We now give extensions to the algorithms defined in Section 6.2. First we extend to *D-from* constraints the definition for the algorithm Red^{gen} that reduces with respect to the generate-only semantics defined in Definition 6.5. While the main Red^{gen} algorithm itself does not need a modification, only the sub-algorithm Red_1^{gen} is extended by two cases for *D-from* constraints. More specifically, recall that Red_1^{gen} chooses a constraint with a non-variable term t on the left-hand side and we have to add the case that said constraint is a *D-from* constraint:

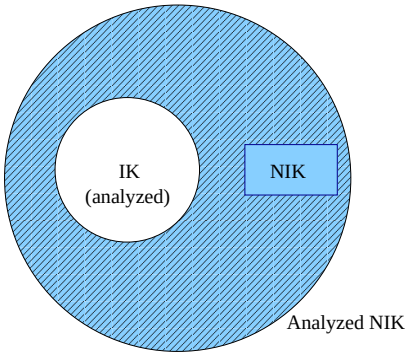


Figure 9.4: Analyzing the intruder knowledge in the context of constraint differentiation.

Definition 9.3. *We extend the definition of* Red_1^{gen} *for the case of D-from constraints as follows:*

$$
\begin{aligned}
&Red_1^{gen}(C \cup \{\textit{D-from}(\emptyset; IK; NIK)\}, \sigma) = \emptyset \\
&Red_1^{gen}(C \cup \{\textit{D-from}(\{t\} \cup T; IK; NIK)\}, \sigma) = \\
&\qquad \bigcup_{(t_1,\ldots,t_n)\in generate(t),(n>0 \vee T\neq\emptyset)} (C \cup \{\textit{D-from}(\{t_1,\ldots,t_n\} \cup T; IK; NIK)\}, \sigma) \\
&\qquad \cup \bigcup_{\tau\in unifyall(t,IK),\ T\neq\emptyset} ((C \cup \{\textit{D-from}(T; IK; NIK)\})\tau, \sigma\tau) \\
&\qquad \cup \bigcup_{\tau\in unifyall(t,NIK)} ((C \cup \{\textit{from}(T; IK \cup NIK)\})\tau, \sigma\tau) \\
&\textit{where } t \notin \mathcal{V}
\end{aligned}
$$

where we again assume that the pattern matching on the left-hand side of $Red_1^{gen}(\cdot,\cdot)$*, i.e. the selection of a non-simple constraint and a non-variable term on the left-hand side of this constraint, is performed by some (deterministic) selection algorithm which we do not prescribe.*

The intuition behind this algorithm is as follows. If there is a constraint $\textit{D-from}(\emptyset; IK; NIK)$ in the constraint set (recall that this constraint is not simple), then there is no solution for the constraint set (since nothing can be generated using knowledge of NIK). If we have to generate the terms $\{t\} \cup T$ from $IK \cup NIK$ and use something from NIK, then we have now three principal choices:

- First, if the term t has the form $t = f(t_1, \ldots, t_n)$ for some intrudable function f, then we have a solution if $\{t_1, \ldots, t_n\} \cup T$ can be generated from $IK \cup NIK$, using something from NIK. Note that we must here exclude the case that both $n = 0$ (e.g. $f = \mathrm{i}$) and $T = \emptyset$: otherwise the left-hand side of the new *D-from* constraint is empty and thus the constraint is unsatisfiable (since nothing new from NIK can be used).

- Second, if the term t can be unified with some term $s \in IK$, then we can remove it and apply the unifier τ to the constraint set. Again, we must exclude the case that $T = \emptyset$ since then the left-hand side of the resulting *D-from* constraint is again empty. Intuitively, this case means that we generate the term t from IK and thus still the obligation remains to use the new knowledge NIK in the generation of one of the other terms T.

- Finally, if t can be unified with a term $s \in NIK$, we indeed have met the obligation to use something from the new knowledge NIK; thus for the generation of the other terms T we do no longer have the obligation to use the new knowledge NIK, so the result is in this case a *from* constraint with $IK \cup NIK$ as knowledge.

This algorithm is not sound with respect to the generate-only semantics. Consider, for instance, a constraint where the *NIK* part can be generated from the *IK* part, e.g.

$$\textit{D-from}(\mathsf{h}(\mathsf{M}); \mathsf{h}(\mathsf{c}); \mathsf{h}(\mathsf{h}(\mathsf{c}))) \ .$$

Actually, the semantics of such a constraint is empty (because nothing can be generated from $IK \cup NIK$ that could not have been generated from *IK* alone. Similar problems exist if a term of T can be matched both with a term in *IK* and in *NIK*. In general, note that the introduction of *D-from* constraints is purely for efficiency reasons. Therefore, in general it is sufficient that the algorithms are sound with respect to $[\![\lceil C \rceil]\!]$ and $[\![\lceil C \rceil]\!]^{gen}$, i.e. without considering the obligation to use the new knowledge. Focusing on efficiency, we may thus avoid some of the checks and exclusion of substitutions that would be necessary for a precise solution (i.e. respecting all obligations).

As the next step, we extend the definition of the *Red* algorithm of Definition 6.7 to integrate *D-from* constraints into the main reduction algorithm:

Definition 9.4. *Let C_1 and C_2 be constraint sets such that C_1 is well-formed and simple, C_2 is well-formed, and σ is a substitution. Then we define the following reduction algorithm:*

$$\textit{Red}(C_1, C_2, \sigma) =$$

$$\begin{cases} \{(C_1, \sigma)\} & \textit{if } C_2 = \emptyset \\ \{(C''', \sigma''') \mid & (IK', C') \in \textit{ana}(IK) \wedge \\ & (C'', \sigma'') \in \textit{Red}^{gen}(C_1 \cup C' \cup \{\textit{from}(T; IK \cup IK')\}, \sigma) \wedge \\ & (C''', \sigma''') \in \textit{Red}(C'', (C_2 \setminus \{\textit{from}(T; IK)\})^{IK'} \sigma'', \sigma'')\} \\ & \textit{if } \textit{fwo}(C_1, C_2) = \textit{from}(T; IK) \\ \{(C''', \sigma''') \mid & (IK', C') \in \textit{ana}(IK \cup NIK) \wedge \\ & (C'', \sigma'') \in \textit{Red}^{gen}(C_1 \cup C' \cup \{\textit{D-from}(T; IK; (NIK \cup IK') \setminus IK)\}, \sigma) \wedge \\ & (C''', \sigma''') \in \textit{Red}(C'', (C_2 \setminus \{\textit{D-from}(T; IK; NIK)\})^{IK'} \sigma'', \sigma'')\} \\ & \textit{if } \textit{fwo}(C_1, C_2) = \textit{D-from}(T; IK; NIK) \end{cases}$$

where $\textit{fwo}(C_1, C_2)$ is the first constraint of C_2 in the order induced by the well-formedness of $C_1 \cup C_2$ (if $C_2 \neq \emptyset$), and where

$$\textit{D-from}(T; IK; NIK)^{IK'} = \textit{D-from}(T; IK \cup IK'; NIK) \ .$$

With respect to Definition 6.7, the algorithm is only extended by the case for a *D-from* constraint, on which the algorithm works as follows.

- First, we determine a sequence of analysis steps with the same analysis algorithm as before on the intruder knowledge $IK \cup NIK$. Note that, even if all possible analysis steps on *IK* have already been made, we need to include it in the analysis of the new intruder knowledge as *IK* may contain a message that was not decipherable using keys in *IK*, but for which *NIK* contains the necessary keys.
- We then add the messages IK' that are obtained by the analysis to the new intruder knowledge part of the *D-from* constraint (and remove from this part every message that is already contained in *IK*) as they have been obtained using new intruder knowledge. The idea behind this is that when later generating some term of T using $IK' \setminus IK$, then also new intruder knowledge is used, because it could not have been derived without *NIK*. This is illustrated in Figure 9.4.[2]
- As the next step, we reduce the constraints of the simple part C_1, the key-derivation constraints C' and the *D-from* constraint with the new intruder knowledge using the $\textit{Red}^{gen}$ algorithm, as only generate steps have to be performed now. (Also, we will show in the correctness proof below that this constraint set is well-formed.)

[2] As said before, this algorithm may miss some possible redundancies, i.e. be sound with respect to only $[\![\lceil C \rceil]\!]$, e.g. when symbolic terms of *IK* and *NIK* are unifiable, but again this is only a problem of efficiency.

- Finally, we recursively call *Red*, using the results of Red^{gen} as the simple constraint part, and the constraints from C_2 without the constraint *D-from*$(T; IK; NIK)$. Note that we add to all constraints of C_2 the newly analyzed terms IK' to guarantee well-formedness. In particular, for every *D-from* constraint of C_2, we add IK' to the old intruder knowledge part, since according to the well-formedness, their old intruder knowledge part is a superset of $IK \cup NIK$ from which all messages in IK' can be derived. Finally, we also apply the substitution σ'' that resulted from the previous reduction with Red^{gen}.

As said above, the derivation is not always precise: while the algorithm is complete, the algorithm may return solutions for a constraint set C that are not sound with respect to $[\![C]\!]$. However, we will show that the algorithm is sound with respect to $[\![\lceil C \rceil]\!]$. Since all solutions of $\lceil C \rceil$ are still valid solutions, the additional solutions are not a problem of correctness of the approach. However, the algorithm may thus miss possible reductions and return a redundant solution, i.e. one of $[\![\lceil C \rceil]\!] \setminus [\![C]\!]$:

Theorem 9.1. *For a well-formed set C of from and D-from constraints, $Red(C)$ terminates with a set of pairs of a simple, well-formed constraint set and a substitution, and C is complete with respect to $[\![C]\!]$ and sound with respect to $\lceil C \rceil$, i.e.*

$$[\![C]\!] \subseteq \bigcup_{(C',\sigma') \in Red(C)} [\![(C', \sigma')]\!] \subseteq [\![\lceil C \rceil]\!] .$$

Moreover, for every $(C', \sigma') \in Red(C)$, $vars(C') \cap dom(\sigma') = \emptyset$ and $vars(C') \cup dom(\sigma') = vars(C)$.

Proof. Again, we split the proof into four parts: invariants, termination, soundness, and completeness.

Invariants Like in the proof of Theorem 6.2, we first show the following invariants. At the input and output of every algorithm, the constraint sets are well-formed, the domain of the substitutions and constraint sets are disjoint, and the set of variables in the constraint set and the substitution is the same as on the input. Additionally, note that Red_1^{gen} is not defined for all inputs, but only for an input (C, σ) where C is not simple, i.e. when there is some non-variable term on the left-hand side of any constraint in C, or a *D-from* constraint with an empty left-hand side. By definition, Red^{gen} calls Red_1^{gen} only for non-simple constraints.

We begin with the invariants for Red_1^{gen}. Recall that we have three cases, namely one for the *from* constraints (Definition 6.5) and two for the *D-from* constraints (Definition 9.3) and we have shown the invariants only for sets of *from* constraints on the first case so far as in the proof of Theorem 6.1. Except for the well-formedness, however, the proofs straightforwardly carry over to the *D-from* constraints and the new case, since $vars(\textit{D-from}(T; IK; NIK)) = vars(\textit{from}(T; IK \cup NIK))$. For the well-formedness, recall that there is an additional condition for constraint sets C that contain *D-from* constraints: whenever *D-from*$(T; IK; NIK) \in C$, then there is also a constraint *from*$(T'; IK) \in \lceil C \rceil$. It is straightforward that this additional condition is also preserved under substitution of a constraint set, i.e. when C is well-formed, then so is $C\sigma$ for any substitution σ. Since Red_1^{gen} does not perform changes to the intruder knowledge of the constraints other than applying substitutions, it is straightforward that the well-formedness is never destroyed by Red_1^{gen}. By induction, we can conclude that the invariants are also preserved by Red^{gen}.

Turning to the *Red* algorithm, all invariants except well-formedness immediately follow from the proof of Theorem 6.2 using $vars(c) = vars(\lceil c \rceil)$. We thus come to the well-formedness in *Red*. If $C_1 \cup C_2$ is well-formed, then IK_i is a superset of all intruder knowledges in $\lceil C_1 \rceil$ whenever *from*$(T_i; IK_i) \in C_2$ or *D-from*$(T_i; IK_i; NIK_i) \in C_2$. Therefore, in both the cases $fwo(C_1, C_2) =$ *from*$(T; IK)$ and $fwo(C_1, C_2) =$ *D-from*$(T; IK; NIK)$ we have that the constraint set that Red^{gen} is called with is well-formed. Therefore also the resulting C'' is well-formed. Moreover since Red^{gen} does not perform any changes to the IK and NIK parts of constraints other than substitution, we have that IK_i is a superset of all intruder knowledges in $\lceil C'' \rceil$ whenever *from*$(T; IK_i) \in C_2^{IK'}\sigma''$ or *D-from*$(T; IK_i; NIK_i) \in C_2^{IK'}\sigma''$. Thus also the recursive call to *Red* is in both cases with a well-formed constraint set.

We conclude the proof of the invariants by showing that all constraint sets in the results Red^{gen} and Red are simple. By definition, Red^{gen} can only return simple constraint sets (as long as the constraint is not simple further reductions with Red_1^{gen} are performed). The C_1 part of the constraint set in Red remains simple throughout reduction. Finally, the initial call $Red(C)$ is a short-hand for $Red(\emptyset, C, id)$, which satisfies all invariants including the one about simplicity, we thus have that the invariants hold throughout the reduction process.

Termination The two new cases in Red_1^{gen} for *D-from* constraints result in finite case splits as before, and the weight of the modified constraint set in all cases is strictly smaller than the input constraint set. Thus the extended Red^{gen} still terminates.

The analysis algorithm *ana* is not modified and we have shown that it terminates already. The new case in Red thus calls only terminating algorithms and the number of constraints in the second argument of Red is reduced by one in every recursive call. Thus the extended Red also terminates.

Soundness The soundness of $Red(C)$ has to be shown only with respect to $[\![\lceil C \rceil]\!]$. Since $Red(\lceil C \rceil)$ is already sound with respect to $[\![\lceil C \rceil]\!]$ by Theorem 6.2, it is therefore sufficient to show that $(C', \sigma') \in Red(C)$ implies $(\lceil C' \rceil, \sigma') \in Red(\lceil C \rceil)$.

This follows immediately when considering the new cases in Red_1^{gen}, Red^{gen}, and Red: applying the function $\lceil \cdot \rceil$ to the resulting constraint sets of the new cases gives only constraint sets that also result when first applying $\lceil \cdot \rceil$. We show this only for the generate case of Red_1^{gen}, the other cases are similar. Consider the constraint $C' = C \cup \{\mathit{D\text{-}from}(\{t\} \cup T; IK; NIK)\}$ in Red_1^{gen} and any case of the generate part $(t_1, \ldots, t_n) \in \mathit{generate}(t)$. When applying $\lceil \cdot \rceil$ to the resulting constraint, we have

$$\lceil C \cup \{\mathit{D\text{-}from}(\{t_1, \ldots, t_n\} \cup T; IK; NIK)\} \rceil = \lceil C \rceil \cup \{\mathit{from}(\{t_1, \ldots, t_n\} \cup T; IK \cup NIK)\}$$

This solution identically results from reducing $\lceil C' \rceil$ with Red_1^{gen}. The argumentation for the other cases of Red_1^{gen}, as well as for Red^{gen} and Red is similar. Thus we can conclude that $Red(C)$ is sound with respect to $[\![\lceil C \rceil]\!]$.

Completeness For the completeness of Red^{gen}, we perform a similar proof as before, namely showing that all transformations performed on the constraint preserve the existence of a labeling with Dolev-Yao proofs for a given ground solution σ_0 (if one exists). Here we extend the concept of a correct Dolev-Yao labeling to $\mathit{D\text{-}from}(T; IK; NIK)$ with respect to a solution σ_0 as follows: the Dolev-Yao proofs for the terms of $T\sigma_0$ may contain as leaves any terms from $(IK \cup NIK)\sigma_0$, but with the additional restriction that at least one leaf of all proofs for T is a term that is contained in $NIK\sigma_0$. Within Red^{gen}, we asumme that these Dolev-Yao proofs contain no analysis steps, as the completeness of Red^{gen} is with respect to the $[\![\cdot]\!]^{gen}$ semantics.

Consider now a call to Red_1^{gen}. In the reduction of a term t in a *from* constraint, the completeness proof of Theorem 6.1 carries over straightforwardly. When reducing a *D-from* constraint, we have two cases, namely that the left-hand side is empty and the case of a non-variable term on the left-hand side. In the case of an empty left-hand side, we have that the semantics of the constraint (and thus of the entire constraint set) is empty, thus there cannot be a substitution σ_0 that satisfies the constraints and no correct Dolev-Yao labelings for the terms of the left-hand side (in particular, where at least one leaf is an instance of a term in NIK). It is therefore correct to return an empty set of solutions of the reduction step.

Consider now the second case that we reduce a non-variable term t of on the left-hand side of a *D-from* constraint, and we show that every solution σ_0 is preserved by the reduction, using the Dolev-Yao proofs for all left-hand side terms under σ_0. We make a case split with respect to the root node of the Dolev-Yao proof labeling the term t. We consider three cases, namely that the root node is not a leaf, that the leaf is a term of $IK\sigma_0 \setminus NIK\sigma_0$, and that the leaf is a term of $NIK\sigma_0$. (Note that the leaf node may be contained in both $IK\sigma_0$ and $NIK\sigma_0$, which is subsumed by the third case.)

Recall that according to the requirements for a correct labeling of *D-from* constraints, at least one leaf of the Dolev-Yao proofs of $\{t\} \cup T$ is a term of $NIK\sigma_0$.

- The root node is an inner node: As the Dolev-Yao proofs are generate-only proofs, it is a composition from the subterms using an intrudable function symbol f. Since $t \notin \mathcal{V}$, we have that $t = f(t_1, \ldots, t_n)$ for some intrudable f and terms $t_1, \ldots, t_n$ such that the subproofs of the Dolev-Yao proof are σ_0 instances of the t_i. Also, $(t_1, \ldots, t_n) \in \mathit{generate}(t)$ and thus the generate part of Red_1^{gen} produces a new constraint that still supports σ_0: namely we can label the constraint $\mathit{D\text{-}from}(\{t_1, \ldots, t_n\} \cup T; IK; NIK)$ with the subproofs of the proof of $t\sigma_0$. Finally, the condition is satisfied that one of the terms on the left-hand side of the constraint is labeled with a Dolev-Yao proof with a leaf from $NIK\sigma_0$.

- The root node is a leaf from $IK\sigma_0 \setminus NIK\sigma_0$: In this case, there must be a term $t' \in T$ that is labeled with a Dolev-Yao proof with a leaf in $NIK\sigma_0$, or otherwise the precondition of a correct labeling would be violated. Thus $T \neq \emptyset$ and there must be a substitution $\tau \in \mathit{unifyall}(t, IK)$ such that $\sigma_0 \succeq \tau$. This case thus supports the substitution σ_0 and can be correctly labeled namely using the same labels as before for all terms of T where at least one node is from $NIK\sigma_0$.

- The root node is a leaf from $NIK\sigma_0$: In this case, there is a substitution $\tau \in \mathit{unifyall}(t, NIK)$ with $\sigma_0 \succeq \tau$, so the new constraint still respects σ_0 and can be correctly labeled, namely all terms of T are labeled with the same proofs as before, where we do no longer have the restriction that at least one leaf of the proofs is from $NIK\sigma_0$, since the new constraint is a *from* constraint.

This shows the completeness of Red_1^{gen} and by induction completeness follows for also for Red^{gen}.

We now show the completeness of $\mathit{Red}(C)$ with respect to $[\![C]\!]$. As before, we will show that if there is a solution σ_0, there is a constraint that still supports σ_0. To that end, we have again all left-hand sides of the constraints labeled with Dolev-Yao proofs for the left-hand side under σ_0. As an invariant the constraints in C_1 part of the constraint set are still labeled with generate-only Dolev-Yao proofs, and we have to show that there is a sequence of analysis steps in $\mathit{ana}(IK)$ and $\mathit{ana}(IK \cup NIK)$, respectively, that allows for a labeling with generate-only proofs for the constraints in the first arguments of the recursive calls to *Red*.

For the case $\mathit{fwo}(C_1, C_2) = \mathit{from}(T; IK)$ the proof of Theorem 6.2 can still be applied. In the case $\mathit{fwo}(C_1, C_2) = \mathit{D\text{-}from}(T; IK; NIK)$, we can proceed similarly. Namely, we consider again an analysis node in a Dolev-Yao proofs of T with respect to σ_0, such that the derivation of the key-term does not contain analysis steps and that the analyzed term s in the proof tree is a leaf (this is possible by the same argumentation as before), i.e. we have the form

$$\dfrac{\overline{s} \quad \dfrac{P}{k}}{c}\ A_{\mathrm{scrypt}}, A_{\mathrm{crypt}}, A_{\mathrm{sign}}, \text{ or } A_{\mathrm{pair}_i}\ .$$

(without the subtree $\dfrac{P}{k}$ in the case A_{pair_i} or A_{sign}), where P is a generate-only Dolev-Yao proof for the key-term k.

Thus there is a term $t \in IK \cup NIK$ with $s = t\sigma_0$. Again, we can eliminate the case that $t \in \mathcal{V}$: if that is the case, then $t \in T$ for some left-hand side T of the simple constraint set C_1 and thus there is a non-variable term $t' \in IK$ such that $s = t'\sigma_0$.

Considering now the analysis of a non-variable term $t \in IK \cup NIK$. Since $t\sigma_0 = s$ and $t \notin \mathcal{V}$, it must have the form $\{\!|t_2|\!\}_{t_1}$, $\{t_2\}_{t_1}$, $\mathsf{sign}(t_1, t_2)$, or $\langle t_1, t_2\rangle$, and thus $(k', M) \in \mathit{decana}(t)$ with either $k = k'\sigma_0$ or $k' = \mathsf{i}$ and where M is the set of subterms of t that are obtained by the analysis, i.e. for the clear-text c in the Dolev-Yao proof, we have that $c \in M\sigma_0$.

We thus have a sequence of analysis steps in $\mathit{anaterms}(IK \cup NIK)$ that begins with (k', M). The resulting constraint $\mathit{from}(\{k'\}; IK \cup NIK)$ that *ana* returns can be correctly labeled with the subproof for k and contains only generate-steps. Moreover, it holds that $M \subseteq IK'$ for the intruder knowledge IK' that is returned by *ana*. Therefore, we can replace with a leaf node

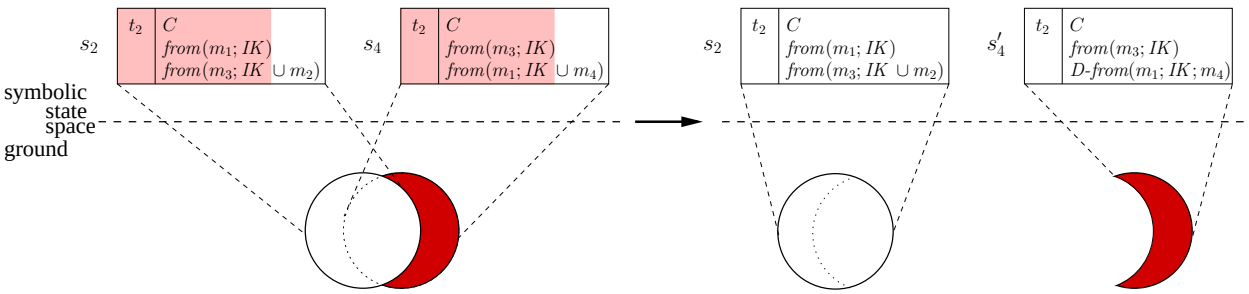


Figure 9.5: Constraint differentiation at work.

every analysis node in the Dolev-Yao proofs of the analyzed constraint $D\text{-}from(T; IK; NIK)$ that have the clear-text c as a result. This gives a correct labeling for the augmented constraint $D\text{-}from(T; IK; NIK \cup IK')$, as $c \in IK'\sigma_0$, and also with c we have at least one leaf in the Dolev-Yao proofs that is a σ_0 instance of a term in the new intruder knowledge part of the augmented constraint.

As before, we can repeat this argumentation to replace all analysis nodes with leaf nodes using $ana(IK \cup NIK)$, where the newly added constraints can be correctly labeled with generate-only Dolev-Yao proofs and the new leaf nodes are σ_0 instances of terms in the new intruder knowledge. Thus the new constraints of C' still support σ_0, as well as $D\text{-}from(T; IK; NIK \cup IK')$, and all of them have generate-only Dolev-Yao proofs. Therefore by the completeness of Red^{gen} with respect to $[\![\cdot]\!]^{gen}$, also the resulting constraint C'' still supports σ_0 with respect to $[\![\cdot]\!]^{gen}$, and thus the condition for the recursive call to *Red* is met, so finally also C''' still supports σ_0, and we have shown the completeness of *Red*. This concludes the proof of termination and correctness of *Red*. □

By Theorem 9.1, we can now apply *D-Red* to reduce the search space without excluding attacks or introducing new ones. To formalize this reduction of the search space, we now integrate constraint differentiation into the symbolic transition system we have introduced in Chapter 8.

9.3 Integrating Constraint Differentiation

Consider again the tree of Figure 9.3. This situation (which occurs whenever two independent actions are performed in two different orders, as explained in Section 9.1) characterizes when we apply constraint differentiation: we exploit the fact that the two symbolic states s_2 and s_4 of Figure 9.3 represent overlapping sets of ground states as shown by the shaded parts in s_2 and s_4.

Figure 9.5 merges parts of Figure 9.2 and Figure 9.3 to illustrate how constraint differentiation works: we pick one, say s_4, of the overlapping states s_2 and s_4 in Figure 9.3 (where $t_2 = t_4$) and replace the *from* constraint that does not appear in the other constraint set with a *D-from* constraint; this yields the transformed state s'_4. That is, we use differentiation of constraints to restrict the extension of one of the two symbolic states to those ground states that are not covered by the other (as illustrated by the shaded part in the set of ground states). The following theorem shows that s_2 and s'_4 represent the same ground states as s_2 and s_4.

Note that, we directly work on the constraint representation here, rather than using the constraint set data-type which would require extensions and be of little value in terms of clarity. We thus see a symbolic state in this chapter simply as a pair (t, C) where t is a set of symbolic facts and C is a constraint set (which may contain inequalities).

Theorem 9.2. *Consider symbolic states $s_2 = (t_2, C_2)$ and $s_4 = (t_2, C_4)$ with constraint sets of the form $C_2 = C \cup \{from(m_1; IK), from(m_3; IK \cup m_2)\}$ and $C_4 = C \cup \{from(m_3; IK), from(m_1; IK \cup m_4)\}$ for some messages m_1, m_2, m_3, m_4 and some constraint set C. Then $[\![s_2]\!] \cup [\![s_4]\!] = [\![s_2]\!] \cup [\![s'_4]\!]$ for $C'_4 = C \cup \{from(m_3; IK), D\text{-}from(m_1; IK; m_4)\}$ and $s'_4 = (t_2, C'_4)$.*

IKE Aggressive Mode Pre-Shared Key													
Mode:	without CD						with CD						
Scenario:	$[a,b],[a,i]$		$[a,b],[a,i],[i,a]$		$[a,b],[a,i],[i,a],[b,i]$		$[a,b],[a,i]$		$[a,b],[a,i],[i,a]$		$[a,b],[a,i],[i,a],[b,i]$	
Ply	s1	s2	s1	s2	s1	s2	s1	s2	s1	s2	s1	s2
1	3	3	4	4	5	5	3	3	4	4	5	5
2	7	7	14	14	23	23	5	5	10	10	16	16
3	13	14	43	45	97	100	7	8	19	21	40	43
4	17	27	112	139	368	420	6	12	30	44	86	111
5	15	53	238	422	1228	1727	5	17	35	81	150	261
6	15	101	393	1262	3501	6989	3	18	31	139	218	578
7		191	483	3699	8232	27835		20	22	215	241	1174
8		410	420	10637	15288	108927		23	8	319	203	2290
9		720		29783	21168	417862		22		436	136	4112
10		960		79939	18900	1565354		12		527	48	7025
11		990		201861		5695140		9		602		11062
12		990		467533		TO		5		576		16390
13				929500		TO				428		22544
14				1583582		TO				233		27443
15				2132130		TO				177		31024
16				1801800		TO				53		29595
17						TO						10531
18						TO						10531
19						TO						7857
20						TO						2371
Nodes	71	4467	1708	7242353	68811	TO	30	155	160	3866	1144	197426
Time	0.16s	13.66s	4.64s	40655.50s	3m41s	TO	0.08s	0.49s	0.49s	21.60s	4.17s	26m30s

Table 9.1: Comparison of OFMC without and with constraint differentiation (CD) for IKE Aggressive Mode Pre-Shared Key: the nodes for each ply of the search tree and search time.

Proof. It suffices to show that $[\![C_2]\!] \cup [\![C_4]\!] = [\![C_2]\!] \cup [\![C'_4]\!]$. Since $\lceil C'_4 \rceil = C_4$ and $[\![\lceil C'_4 \rceil]\!] \supseteq [\![C'_4]\!]$, the direction $\supseteq$ is trivial. To show $[\![C_2]\!] \cup [\![C_4]\!] \subseteq [\![C_2]\!] \cup [\![C'_4]\!]$, we show that for every solution $\sigma \in [\![C_4]\!]$ with $\sigma \notin [\![C_2]\!]$ it holds that $\sigma \in [\![C'_4]\!]$. So assume $\sigma \notin [\![C'_4]\!]$. Then, since $\sigma \in [\![C_4]\!]$, it must hold that $\sigma \in [\![C_4]\!] \setminus [\![C'_4]\!]$, i.e. $\sigma \in [\![\lfloor C_4 \rfloor]\!]$. Hence, $\sigma \in [\![\mathit{from}(m_1; IK)]\!]$ and, since $\sigma \in [\![\mathit{from}(m_3; IK)]\!]$, we have $\sigma \in [\![C_2]\!]$, which contradicts the assumption. □

This theorem allows us to perform a transformation on the search tree that replaces *from* constraints with more restrictive *D-from* constraints without changing the set of represented ground states. If under the more restrictive constraint C'_4 the intruder could not use any new message from his knowledge, then even if C_4 is satisfiable, C'_4 is unsatisfiable (so that the shaded part of the set of ground states in Figure 9.5 is also empty), which we can check using *D-Red*. This is the maximal reduction that can be achieved by constraint differentiation: the node of the state s_4 and its subtree can be completely removed from the search tree as the intruder could not generate anything "interesting", i.e. nothing that he could not have generated before. Note that we can always consider the symmetric situation: if performing the restriction on s_2 rather than s_4 leads to an unsatisfiable constraint set, then we can remove the respective subtree.

When we apply *D-Red* to a state that results by replacing *from* constraints with *D-from* constraints, in the best case the constraint set turns out to be unsatisfiable, so the state (and the respective subtree) can be removed. However, it is also possible that after applying *D-Red* there still remain simple *D-from* constraints (i.e. with variables in the T-part). This means that it is not yet determined what the intruder will use here, so it is possible that it is some message from NIK. Such a *D-from* constraint is nonetheless useful for the reduction, as it constrains the child nodes by excluding certain solutions: the *D-from* constraint prevents all later instantiations of the variable in the T-part if these instantiations do not use some message of the NIK-part.

9.4 Experimental Results

To test the practical relevance of constraint differentiation, we present the difference on the performance of our tool that results from using constraint differentiation. (Experiments where performed on a 2,4 GHz Pentium-4 PC with 512 MB RAM.)

		IKE Main Mode Pre-Shared Key					IKE Aggressive Mode Pre-Shared Key				
Scenario	sn	Ply	without CD		with CD		Ply	without CD		with CD	
			Time	Nodes	Time	Nodes		Time	Nodes	Time	Nodes
$[a,b]$	1	7	0.11	36	0.08	24	4	0.02	7	0.01	6
	2	14	0.30	91	0.20	61	8	0.04	18	0.02	12
$[a,b],[a,i]$	1	11	45.41	8969	9.18	1902	6	0.15	71	0.08	30
	2	22	TO	TO	11m25	98271	12	13.82	4467	0.48	155
$[a,b],[i,b]$	1	10	10.99	2034	3.93	704	6	0.18	75	0.08	28
	2	20	185m52	1014127	3m45	21155	12	18.51	4897	0.57	159
$[a,b],[a,b]$	1	14	22m43	169531	2m38	19642	8	1.81	689	0.35	123
	2	28	TO	TO	429m09	2020491	16	68m56	906789	6.65	1508

Table 9.2: Comparison of search time and tree size without and with constraint differentiation (CD) for IKE Main Mode and Aggressive Mode Pre-Shared Key.

As a relevant example of an industrial-strength protocol we consider the protocol-suite IKE [86]. We have used OFMC to analyze the full specification of each of the individual subprotocols of IKE and several combinations of them. By "full" we mean that we did not simplify the structure of the messages, which contain for instance complex key-terms. OFMC detected a number of minor weaknesses of IKE, which have previously been reported in [109, 133]. To discuss concrete performance results, we consider two subprotocols of IKE, the *Main Mode* and the *Aggressive Mode* of Phase 1 in the pre-shared key variants.

Table 9.1 compares the size of plies of the search tree for Aggressive Mode without and with constraint differentiation. We consider six scenarios varying in the number of parallel and consecutive sessions. We consider two ($[a,b],[a,i]$), three (also $[i,a]$) or four (also $[b,i]$) sessions, where $[a,b]$ means that agent a plays the protocol initiator role and agent b the responder role, and i is the intruder. Moreover, sn denotes n consecutive sessions, where s1 (i.e. only one session) is the standard case and s2 means that an honest agent who has finished his part of the protocol session is prepared to engage in a further session with the same partners. Consecutive sessions are valuable for detecting replay attacks as they create a smaller search space than the same sessions in parallel. We display the number of nodes on each ply of the search tree; note that for the "smaller" scenarios (i.e. fewer parallel and consecutive sessions) the depth of the search tree is smaller, hence the empty cells. We also display the total number of nodes in the tree and the CPU time for searching the entire tree (TO denotes *time out* after one day CPU time).

Table 9.1 shows that constraint differentiation is most effective, as measured by the number of nodes that must be searched, when the original search space contains many interleavings of parallel sessions. The savings are most dramatic on the deeper plies of the search tree as the number of interleavings grows exponentially in the original model; since many interleavings are redundant and constraint differentiation can exploit this redundancy, the number of nodes does not necessarily grow exponentially with the depth of the tree. The difference between an exponential growth without constraint differentiation and an often sub-exponential growth with constraint differentiation leads to more dramatic savings the deeper the tree is searched. Note also that without constraint differentiation the number of nodes on each ply typically grows monotonically with the depth of the ply; with constraint differentiation the number grows on the first few plies and then starts to shrink again. This phenomenon is explained by the fact that with constraint differentiation the deeper we are in the tree, the more successor nodes are completely excluded. The intuition behind this is that many transitions possible in the original model do not permit the use of newly learned messages and are thus pruned from the tree by constraint differentiation.

In summary, by employing constraint differentiation, the OFMC tool scales significantly better with the size of the considered scenario. This extends the scope of OFMC and allows its use in verification for a bounded number of sessions. Table 9.2 shows the broader picture, assessing the savings due to constraint differentiation for both main and aggressive mode for different parallel and consecutive sessions. Constraint differentiation reduces the search space significantly in all cases, and in some cases it even enables the analysis of problems that were out of the scope of OFMC (without constraint differentiation) and other tools.

Part III

Algebraic Properties

In the previous part, we have designed an efficient, symbolic, constraint-based method for the analysis of security protocols: the lazy intruder with constraint differentiation. Throughout Part II, we have used the free algebra assumption, namely that two terms are interpreted differently when they are syntactically different.

As explained in Section 3.5, many security protocols fundamentally depend on the algebraic properties of cryptographic operators (see also [62] for a survey). For example, protocols based on the Diffie-Hellman key-exchange, such as the Station-to-Station, IKE, and JFK protocols, exploit the property of modular exponentiation that $(g^x)^y \; mod \, p = (g^y)^x \; mod \, p$. Without this property, these protocols could not even be executed.

It is however non-trivial to analyze security protocols in presence of algebraic properties. Basic problems like the equality of terms (the ground word problem) and the unifiability problem are in general only semi-decidable, even for relatively simple algebraic theories [17, 19, 95].

The core problem in the context of a formal analysis of security protocols is the intruder deduction problem, introduced in Definition 3.9, namely whether $m \in \mathcal{DY}_E(M)$ holds: can the intruder derive the message m if he knows the set of terms M in the algebra defined by the set of equations E? (Note that both m and M are ground here.) As shown in [1, 2], the intruder deduction problem can be undecidable even for theories where E-unifiability is decidable.

Solutions for the intruder deduction problem have been given for individual algebraic theories of cryptographic operators, such as those formalizing different properties of modular exponentiation or bitwise exclusive or [49, 50, 112]. However, even though these approaches are specialized to particular algebraic properties, the algorithms and correctness proofs are quite complex and usually must be revised or completely re-designed when new properties are added. More general approaches have been recently proposed [51, 97, 110] and we compare our work with them below.

In this part, we present a framework integrating algebraic properties into the analysis of security protocols. This framework is general and can handle algebraic properties of cryptographic operators in a uniform and modular way. By implementing this framework in our analysis tool OFMC we allow users to declare new operators and properties as part of the protocol specifications. Moreover, our framework can be integrated into different methods for protocol analysis. Of course, given the undecidability of the relevant problems, this goal cannot be achieved in full, without any restrictions. For instance, recall that in Theorem 5.1 we showed that, modulo E, neither protocol safety nor insafety is semi-decidable, whenever the ground word problem for the considered algebraic theory E is undecidable. We now briefly describe the main ideas and restrictions of our approach.

Our framework is based on two ideas. The first idea is to use *modular rewriting* to formalize a generalized equational deduction problem for the Dolev-Yao intruder. In doing so, we exploit the fact that we can distinguish two kinds of equational theories associated with security protocols: *cancelation theories* (where equations express that certain operations cancel each other out, such as encryption and decryption with the same symmetric key) and *finite theories* (which are theories that induce finite equivalence classes for all terms, such as the property (3.3) needed for Diffie-Hellman). We show how our use of modular rewriting provides a general solution to the algebraic intruder deduction problem that is not specialized to a particular algebraic theory.

The second idea is to introduce two "depth parameters" that bound the depth of message terms and the operations that the intruder can use to analyze messages (i.e. decompose messages based on his current knowledge). These bounds control the complexity of the equational unification problems that arise, transforming undecidable problems into decidable ones. Along the way, we also show that the considered subproblems (such as unification) become undecidable when any of the restrictions made in our framework are removed. Note also that many protocol analysis methods require bounds on messages in the first place as we have mentioned in Subsection 4.4.4.

Our algorithms are less efficient than those algorithms, when they exist, that are specialized to particular algebraic theories, e.g. [49, 50, 112], and usually work without bounds. Our framework is open to the integration of such specialized algorithms, albeit under the restriction of bounded message depth. In this way, we can benefit from research advances for specialized theories, while being able to fall back on general algorithms when specialized ones are not available.

Finally, we note that our framework is not biased towards a particular protocol analysis method.

It can be used as a basis for handling algebraic equations when employing different types of formalisms (such as strand spaces, process calculi, or rewriting) or techniques (such as abstractions or the symbolic *lazy intruder* technique).

Chapter 10

Finite Theories

10.1 Two Classes of Theories

Our framework is based on *modular rewriting* and exploits the fact that we can distinguish two kinds of equational theories associated with security protocols: *cancelation theories* and *modulo theories*. The idea is that the equations of a cancelation theory express that certain operations (such as encryption followed by decryption with the same key) cancel each other out. Such equations can usually be described by a convergent term-rewriting system and we can thus apply these equations to rewrite all terms into normal-form. The advantage of separating out a convergent subtheory is that we can then neglect its equations during subsequent equality reasoning when all terms are normalized. On the other hand, a modulo theory comprises equations that cannot be oriented into terminating rewrite rules; the standard examples from rewriting are the equations for properties like associativity and/or commutativity. It is common for these equations to form a "background theory" used when applying other rewrite rules (such as the cancelation equations); that is, one performs rewriting modulo the equations of a modulo theory. However, we will not restrict ourselves to a particular modulo theory, like AC, but rather work with a class of theories, namely *finite theories*.

Definition 10.1. *An equational theory E is finite if the equivalence class $[t]_{\approx_E} = \{t' \mid t' \approx_E t\}$ is finite for all terms $t \in \mathcal{T}_\Sigma(\mathcal{V})$.*

A cancelation rule *has the form*

$$f(t_1, \ldots, t_n) \to s \ ,$$

where s is a constant or a subterm of the t_i (and thus the variables of the right-hand side are a subset of the variables on the left-hand side). A cancelation theory *is a set of equations $s \approx t$ such that $s \to t$ or $s \leftarrow t$ is a cancelation rule.*

In the following, we will use F and C to denote finite and cancelation theories, respectively.

Considering our example theory E_{ex} from Figure 3.2, we can decompose the set of equations into two theories F_{ex} and C_{ex} such that F_{ex} is a finite theory and C_{ex} is a cancelation theory: Let F_{ex} be the subset of equations (3.1)–(3.3), and let C_{ex} be the subset of equations (3.4)–(3.13). Note that these theories are not *disjoint*, i.e. the set of symbols from Σ that occurs in them is not disjoint. However finite and cancelation theories are disjoint classes of theories as for a cancelation theory, there are always terms with an infinite equivalence class.

From the definition of finite theories, we can derive that matchability modulo a finite theory is decidable and that there are finitely many matches:[1]

Theorem 10.1. *Given a ground term t and a symbolic term s, the problem whether there is a substitution σ with $t \approx_F s\sigma$ is decidable for every finite theory F. Moreover, there exists a finite algorithm that returns a complete set of substitutions for this property.*

[1] Recall that we require that the matching term is ground.

Proof. $t \approx_F s\sigma$ iff there is a term $t_0 \in [t]_{\approx_F}$ such that $t_0 \approx s\sigma$ for some substitution σ. Since F is a finite theory, the equivalence class $[t]_{\approx_F}$ is finite by definition, and thus we have reduced the problem to finitely many instances of syntactic matching in the free algebra. □

It is straightforward to extend this result to matching for a set of pairs of a ground term and a symbolic term. Also, a special case of equational matching is the ground word problem (when s is also ground), and hence this problem is also decidable for finite theories.

As we will see below, our framework relies on the decidability of matching for finite theories. In contrast, the unification problem (where both terms may contain variables) for finite theories is undecidable. Consider, for example, the theory of distributivity and associativity D_A which is defined by the following equations (there are two equations for distributivity since $\cdot$ is not commutative):

$$\begin{aligned} x \cdot (y + z) &\approx (x \cdot y) + (x \cdot z) \\ (x + y) \cdot z &\approx (x \cdot z) + (y \cdot z) \\ x + (y + z) &\approx (x + y) + z \} \, . \end{aligned}$$

Unifiability in this theory is undecidable as shown in [125]. As equivalence classes in D_A are finite, we thus have that unifiability modulo a finite theory is in general undecidable.

In Chapter 11, we will use the following important property of finite theories, namely that they cannot contain equations that introduce new variables:

Lemma 10.1. *If $l \approx r$ is an equation of a finite theory, then $vars(l) = vars(r)$.*

Proof. Suppose, without loss of generality, that a finite theory F contains an equation $(l \approx r)$ with a variable $v \in vars(l) \setminus vars(r)$, and let σ be a ground substitution for r. Then the ground term $r\sigma$ matches (by σ) the right-hand side r, and thus is equal, under F, to $l\sigma$ for any substitution of the remaining variables. As there is at least one variable v, and we can instantiate it with any term, the equivalence class of $r\sigma$ is infinite and hence F cannot be a finite theory. □

Hence, $l \in \mathcal{V}$ implies $l = r$, so that such trivial equations can be safely omitted.

We conclude this section by observing the relevance of these two kinds of theories to security protocol analysis. As we will see, cancelation rules are closely related to the analysis (e.g. decryption) of terms by the intruder and honest agents, and therefore have a distinguished role in deductions. We will namely define a normal-form of the intruder knowledge as a state where the applications of cancelation rules do not give him any "new" terms (in a sense to be precisely defined later).

10.2 Restriction to a Bounded Variable Depth Model

As unifiability modulo a finite theory is undecidable, we introduce a restriction under which F-unifiability becomes decidable. We introduce bounds on messages, namely on the terms that can be substituted for variables. We discuss in Section 11.3, at which point in the protocol verification these bounds can occur and what they imply. Roughly speaking, we will use them to bound the depth of terms that the intruder can compose (or that honest agents ever accept).

In contrast to the previous two parts, we will assume here that the signature Σ is finite, such that there is a finite set of ground terms of a given maximal depth.[2] We first define a subset of the variable symbols with an associated depth bound, and we then define which substitutions are permissible for these variables.

Definition 10.2. *We call a* bounded variable *a variable for which only terms with bounded depth can be substituted. Let $\mathcal{VB} \subseteq \mathcal{V}$ be the* set of bounded variables *such that every variable v has*

[2] As long as we consider a bounded number of sessions, we can compute in advance how many fresh constants will be needed in any trace, thus there is no loss of generality from this restriction.

an associated depth bound $depth(v) \in \mathbb{N}$. *We extend the function* $depth(\cdot)$ *to arbitrary terms as follows:*

$$\begin{array}{rcll} depth(v) & = & \infty & \text{for } v \in \mathcal{V} \setminus \mathcal{VB}\,, \\ depth(c) & = & 1 & \text{for } c \in \Sigma_0, \\ depth(f(t_1,\ldots,t_n)) & = & 1 + max_{i=1}^{n}\ depth(t_i) & \text{for } f \in \Sigma_n,\ \text{with } n > 0\,. \end{array}$$

We say that a substitution σ respects the depth restrictions*, and write* $respect\_depth(\sigma)$*, iff* $depth(x\sigma) \leq depth(x)$ *for every* $x \in dom(\sigma)$.

For example, we can formulate the problem whether, given terms s and t with $vars(s) \cup vars(t) \subseteq \mathcal{VB}$, it holds that

$$\exists\sigma.\, respect\_depth(\sigma) \wedge s\sigma \approx_E t\sigma\ .$$

We thus bound the maximum depth of terms that may be substituted for variables e.g. in a unification. We will also simply speak of *bounded variables* and *bounded terms* when their depth is not ∞.

Recall that in Subsection 4.4.4, we have sketched a typed model for security protocols that associates atomic and composed types to every variable (or to a subset of the variables for a partially typed model). Every type can be regarded as a term itself (if we consider atomic types as constants) and thus has a depth. Observe that by the typing restriction, we may substitute for a variable of type τ only terms of maximal depth $depth(\tau)$. Thus, for a typed IF protocol specification, it is not a restriction to require that every variable of type τ shall have depth bound $depth(\tau)$. Therefore, the restriction introduced by typing the variables of an IF specification is a special case of bounding variables.

The following result shows why a bound on variables makes many problems easier. Roughly speaking, we show that every algorithm that requires some of its arguments to be ground (like F-matching before) can be lifted to an algorithm where the respective arguments may contain bounded variables (e.g. F-unification where the terms on one side are bounded):[3]

Theorem 10.2. *Let* f *be a computable function that takes as input* n *terms that may contain variables and* m *ground terms, and which returns a finite set of terms. Then the following function* f' *is also computable.* f' *takes as input* n *terms that may contain (arbitrary) variables and* m *terms that may contain bounded variables, and returns a finite set of terms and substitutions such that:*

$$\begin{array}{l} \forall s_1,\ldots,s_n \in \mathcal{T}_\Sigma(\mathcal{V}).\ \forall t_1,\ldots,t_m \in \mathcal{T}(\Sigma,\mathcal{VB}).\ \forall\sigma. \\ \ [ground(t_1\sigma)\ \wedge \ldots \wedge\ ground(t_m\sigma)\ \wedge\ dom(\sigma) \subseteq \mathcal{VB}\ \wedge respect\_depth(\sigma)] \\ \ \Longrightarrow\ \ [(r,\sigma) \in f'(s_1,\ldots,s_n,t_1,\ldots,t_m)\ \iff\ r \in f(s_1\sigma,\ldots,s_n\sigma,t_1\sigma,\ldots,t_m\sigma)]\,. \end{array}$$

Proof. Since Σ is finite, there are only finitely many ground terms of a given maximal depth. Thus, the set of admissible substitutions for a term with only bounded variables is finite. Now, it is possible to construct f' given f as required:

- Given $t_1,\ldots,t_m$ with only bounded variables, compute the finite set Θ of admissible substitutions for the t_i.
- For every $\sigma \in \Theta$, we compute $R_\sigma = f(s_1\sigma,\ldots,s_n\sigma,t_1\sigma,\ldots,t_m\sigma)$. Note that each R_σ is a finite set by the requirements of the theorem. From the results R_σ, we compute the result for f' as follows:

$$f'(s_1,\ldots,s_n,t_1,\ldots,t_m) = \{(r,\sigma) \mid \sigma \in \Theta \wedge r \in R_\sigma\}\ .$$ □

Theorem 10.2 allows us, for instance, to easily lift the F-matching algorithm for finite theories F to an F-unification algorithm where one of the two input terms contains only bounded variables.

[3]Recall that in this part, we assume that terms are built over a *finite* signature Σ.

10.3 Unification modulo Finite Theories

The algorithms for F-matching and F-unification (with variables on one side bounded) that result out of the constructive proofs of Theorem 10.1 and Theorem 10.2 are based on complete enumerations, namely of the F-equivalence class for matching and of substitutions for F-unification. In this section, we consider an improvement that avoids the entire enumeration and that allows for the integration of specialized algorithms for *disjoint subtheories* of F such as [49, 50, 112]. With disjoint subtheories of F we mean that the equations of F can be partitioned into subtheories $F = F_1 \cup \ldots \cup F_n$ such that F_i and F_j have no symbols from Σ in common for $i \neq j$. For instance, our example theory F_{ex} can be partitioned into subtheories $F_\oplus$, containing the equations (3.1) and (3.2), and the subtheory F_{exp}, containing the equation (3.3). Every symbol from Σ that does not occur in any equation of F is called a *free* symbol, and concerning the partitioning into disjoint subtheories, we regard free symbols as belonging to an empty subtheory.[4]

The basic idea to improve F-matching and F-unification is as follows. In the free algebra, there is only one way to decompose the term $f(t_1, \ldots, t_n)$ into a function symbol and subterms, namely f and subterms $t_1, \ldots, t_n$. For algebraic theories, there can be several ways to decompose a term, e.g. in F_{ex}, a Diffie-Hellman key $\mathsf{exp}(\mathsf{exp}(g, x), y)$ can be decomposed either into exp and subterms $\mathsf{exp}(g, x)$ and y or into exp and subterms $\mathsf{exp}(g, y)$ and x. Decomposition can be a basis for a unification algorithm: in the free algebra, two non-variable terms are unified by decomposing each of them into a function symbol and subterms and checking that the function symbol is identical and that corresponding subterms can be unified. We can generalize this algorithm for F-unification by taking the multiple decompositions into account. Note that for a finite theory, there can only be finitely many decompositions of ground terms, and thus also finitely many decompositions for terms with bounded variables. Also, we will show that the decomposition of term $f(t_1, \ldots, t_n)$ can only result in a decomposition f' and subterms $t_1, \ldots, t_n$ where f' belongs to the same subtheory as f. This is the basis for integrating specialized algorithms for subtheories. Finally, there are applications of decomposition other than for F-matching and F-unification, namely for intruder deduction modulo F, where we want to know how we can compose a term from its root symbol and subterms—modulo F.

Formally, we define decomposition modulo F as a subproblem of F-unification:

Definition 10.3. *For a term t, an n-ary symbol f and fresh variables $x_1, \ldots, x_n$, we say S is a* complete set of toplevel decompositions*,* CSTD *for short, iff the following holds:*

- *(Soundness) $t\sigma \approx_E f(x_1, \ldots, x_n)\sigma$ for each $\sigma \in S$.*
- *(Completeness) For each σ such that $t\sigma \approx_E f(x_1, \ldots, x_n)\sigma$, there is a $\tau \in S$ such that $\sigma \succeq \tau$.*

Example 10.1. Consider the term $t = \mathsf{X} \oplus \mathsf{c}$, for a variable X of depth 2 and a constant c, in our theory F_{ex}. Then the following set is a CSTD for t, $\oplus$ and the fresh variables X_1 and X_2:

$$\begin{array}{ll}\{ & [\mathsf{X}_1 \mapsto \mathsf{X}, \mathsf{X}_2 \mapsto \mathsf{c}], \\ & [\mathsf{X} \mapsto \mathsf{X}_3 \oplus \mathsf{X}_4, \mathsf{X}_1 \mapsto \mathsf{X}_3, \mathsf{X}_2 \mapsto \mathsf{X}_4 \oplus \mathsf{c}], \\ & [\mathsf{X}_1 \mapsto \mathsf{c}, \mathsf{X}_2 \mapsto \mathsf{X}], \\ & [\mathsf{X} \mapsto \mathsf{X}_3 \oplus \mathsf{X}_4, \mathsf{X}_1 \mapsto \mathsf{X}_4 \oplus \mathsf{c}, \mathsf{X}_2 \mapsto \mathsf{X}_3] \quad \} \ ,\end{array}$$

[4] As a side remark, we mention why we cannot apply a standard result for combining unification algorithms of disjoint theories [18]. The idea of [18] is based on transforming a given unification problem into a *pure* one, where each equation to unify contains only symbols from *one* of the disjoint subtheories. For instance, in our example theory E_{ex}, we may transform the unification problem

$$\mathsf{exp}(\mathsf{g}, \mathsf{X}_1 \oplus \mathsf{X}_2)\sigma \approx \mathsf{exp}(\mathsf{X}_3, \mathsf{X}_4)\sigma$$

into the problems

$$\mathsf{exp}(\mathsf{g}, \mathsf{X}_5)\sigma \approx \mathsf{exp}(\mathsf{X}_3, \mathsf{X}_4)\sigma \quad \text{and} \quad \mathsf{X}_5\sigma \approx \mathsf{X}_1 \oplus \mathsf{X}_2\sigma \ .$$

Then, each equation can be unified using the unification algorithm for the respective subtheory and the results are combined. The problem in our setting is that we cannot support unification without bounds on all variables of at least one side. Suppose that in this example the variables X_1 and X_2 are unbounded, while X_3 and X_4 are bounded. Then the newly introduced variable X_5 cannot be bounded, and thus we have a unification problem with unbounded variables. A similar problem would occur if we consider only matching, e.g. when replacing X_3 and X_4 in the above example with ground terms. Thus the combination is not possible within our framework.

where X_3 and X_4 are fresh variables. The first solution is the obvious syntactic solution. The second one is based on the possibility that under the substitution $[\mathsf{X} \mapsto \mathsf{X}_3 \oplus \mathsf{X}_4]$, we can decompose the term t also into $\mathsf{X}_3 \oplus (\mathsf{X}_4 \oplus \mathsf{c})$. Note that the decomposition $(\mathsf{X}_3 \oplus \mathsf{X}_4) \oplus \mathsf{c}$ is already covered by the first solution. Further decompositions of X_3 and X_4 are not possible, since X has depth 2 and thus X_3 and X_4 can only be substituted for constants. The third and forth solution result from the first two by the commutativity of $\oplus$: for every decomposition that assigns t_1 to X_1 and t_2 to X_2, we also have the decomposition that assigns t_2 to X_1 and t_1 to X_2. □

This example also illustrates the potential for improvement in F-unification with respect to the entire enumeration of all ground substitutions σ for the variables and the subsequent enumeration of the entire equivalence class of the terms under the ground substitution. First, we may restrict ourselves to substitutions that are relevant for the decomposition of terms, and second, we may omit decompositions that are instances of other decompositions, e.g. when the subterms of the two decompositions are F-equivalent. Thus, if we have an efficient CSTD algorithm for a theory F, we can avoid a lot of cases in the F-unification by remaining "as symbolic as possible".

Determining the F-CSTD of a term t with respect to an n-ary symbol f and fresh variables $x_1, \ldots, x_n$ is a special case of F-unification, namely finding a complete set of substitutions σ such that $t\sigma \approx_F f(x_1, \ldots, x_n)\sigma$. This is the key for integrating existing unification algorithms that are specialized to a disjoint subtheory of F. To that end, we first need to show a property of F-unification, namely that we can divide the task of F-unification into computing a CSTD for a subtheory F_i (and with respect to only symbols of F_i) and F-unification of the subterms. We observe that for disjoint finite theories, the root symbol of equivalent terms must belong to the same subtheory:

Lemma 10.2. *Let F_1 and F_2 be two disjoint finite theories over signatures Σ^1 and Σ^2. If*

$$f(t_1, \ldots, t_n) \approx_{F_1 \cup F_2} f'(s_1, \ldots, s_m)$$

holds, then either $f, f' \in \Sigma^1$ or $f, f' \in \Sigma^2$.

Proof. Suppose that $f \in \Sigma^1$ (the proof for the case $f \in \Sigma^2$ is analogous). By Lemma 10.1, both sides of equations in finite theories must have a function symbol (not a variable symbol) as root; in particular the root symbols on both sides of the equation must be from the same signature. Therefore, the "application"[5] of an equation to $f(t_1, \ldots, t_n)$ at position ϵ is only possible if one side of the equation has f as the root symbol, i.e. the equation must be from F_1 and the root symbol on the other side must also be from Σ^1. Therefore the result of the application is a term with a root symbol from Σ^1. By induction, every sequence of applications of equations at position ϵ results in a term with root symbol in Σ^1. Moreover, applications in the subterms does not change the root symbol. Thus every term in the $F_1 \cup F_2$-equivalence class of $f(t_1, \ldots, t_n)$ has a root symbol in Σ^1. □

From this, we can conclude a central property for equivalence in disjoint finite theories, namely that we can split an F-equivalence proof into two parts: one part on the toplevel that is concerned only with the subtheory of the toplevel, and one part for the subterms.

Theorem 10.3. *Let F_1 and F_2 be two disjoint finite theories over signatures Σ^1 and Σ^2, and let $f \in \Sigma^1$. Then*

$$f(t_1, \ldots, t_n) \approx_{F_1 \cup F_2} f'(s_1, \ldots, s_m)$$

iff there are terms $s'_1, \ldots, s'_m$ such that

$$\begin{array}{rcl} f(t_1, \ldots, t_n) & \approx_{F_1} & f'(s'_1, \ldots, s'_m) \\ s_1 & \approx_{F_1 \cup F_2} & s'_1 \\ & \ldots & \\ s_m & \approx_{F_1 \cup F_2} & s'_m \ . \end{array}$$

[5] Recall that we have defined equational equivalence by rewriting with the equations interpreted as rules in both directions.

Proof. The direction "left to right" of the "iff" is straightforward.

For the converse direction, let us define the notion of the Σ^1-*threshold* of a term t, $\mathrm{thr}_{\Sigma^1}(t)$, which is the set of positions in the term such that up to that position only symbols of Σ^1 (and variable symbols) occur:

$$\mathrm{thr}_{\Sigma^1}(t) = \begin{cases} \{\epsilon\} \cup \bigcup_{i=1}^{n} \bigcup_{p \in \mathrm{thr}_{\Sigma^1}(t_i)} i \cdot p & \text{if } t = f(t_1, \ldots, t_n) \wedge f \in \Sigma^1 \ , \\ \emptyset & \text{if } t = f(t_1, \ldots, t_n) \wedge f \notin \Sigma^1 \ , \\ \{\epsilon\} & \text{if } t \in \mathcal{V} \ . \end{cases}$$

Moreover let

$$\mathrm{below}_{\Sigma^1}(t) = \{t_{|p} \mid p \in pos(t) \wedge p \notin \mathrm{thr}_{\Sigma^1}(t)\}$$

be the set of all subterms of t at positions below the Σ^1-threshold.

Now let $P = \mathrm{thr}_{\Sigma^1}(f(t_1, \ldots, t_n))$ and let a proof for $f(t_1, \ldots, t_n) \approx_{F_1 \cup F_2} f'(s_1, \ldots, s_n)$ be given by a sequence of terms $u_1, \ldots, u_k$, where each term can be obtained from the previous term by one application of a rule in F_1 or in F_2, and $u_1 = f(t_1, \ldots, t_n)$ and $u_k = f(s_1, \ldots, s_n)$. Also, let p_i $(1 \leq i < k)$ be the position in the term u_i at which the rule is applied to produce u_{i+1}.

The idea is now to split the proof into two parts: in the first part, rules are applied only above the Σ^1-threshold of the respective terms (and we will show that these rules must be from F_1), and in the second part, rules are applied only below the threshold (but may be from both F_1 and F_2).

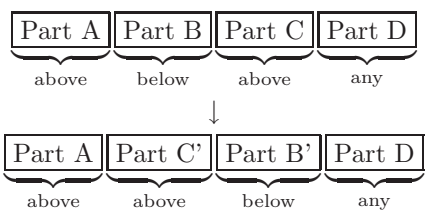
First, we show that for every prefix $u_1, \ldots, u_l$, with $l < k$, of the proof sequence such that $p_i \in \mathrm{thr}_{\Sigma^1}(u_i)$, with $1 \leq i \leq l$, it holds that the applied rule is from F_1 and that $\mathrm{below}_{\Sigma^1}(u_i) = \mathrm{below}_{\Sigma^1}(u_{i+1})$. The latter equality means that the set of subterms below the threshold does not change (although the number of occurrences of identical subterms might change).

We show this by induction over l, i.e. assuming that the conditions hold already for every prefix up to $l - 1$, and $p_l \in \mathrm{thr}_{\Sigma^1}(u_l)$.

By Lemma 10.1, the applied rule must have a top-level symbol from the respective Σ^i. The case $u_{l|_{p_l}} \in \mathcal{V}$ is excluded, as in this case the rule can only be applied when instantiating a variable of u_l (which is impossible as noted above). Therefore $u_{l|_{p_l}}$ is a term that is not a variable; the root symbol must be from Σ^1 as otherwise p_l does not belong to the Σ^1-threshold of u_l. Therefore the top-level symbol of the applied rule must be from Σ^1, i.e. the step is from theory F_1. It remains to show that the new term u_{l+1} has the same set $\mathrm{below}_{\Sigma^1}(u_{l+1})$ as u_l. Let $lhs \approx rhs \in F_1$ be the rule that transforms u_l into u_{l+1}, i.e. there is a substitution σ such that $lhs\sigma = u_{l|_{p_l}}$ and $rhs\sigma = u_{l+1|_{p_l}}$ and $u_l[p_l]rhs\sigma = u_{l+1}$. Suppose there is a $w \in \mathrm{below}_{\Sigma^1}(u_l) \setminus \mathrm{below}_{\Sigma^1}(u_{l+1})$ (the the case $w \in \mathrm{below}_{\Sigma^1}(u_{l+1}) \setminus \mathrm{below}_{\Sigma^1}(u_l)$ is analogous). Since the only change is below position p_l, it must hold that $w \in \mathrm{below}_{\Sigma^1}(u_{l|_{p_l}})$. Let p be the position of w in $u_{l|_{p_l}}$. It holds that w is below the threshold, that lhs cannot contain symbols from Σ^2, and that $lhs\sigma$ matches $u_{l|_{p_l}}$; therefore it follows that there is a prefix p' of p (not necessarily a proper one) such that $lhs_{|p'}$ is a variable. By Lemma 10.2, this variable also occurs in rhs, thus $rhs\sigma$ has w as a subterm, which contradicts that w does not occur in u_{l+1}.

Up to here we have established that an initial part of the equivalence proof that applies rules above the threshold can only apply F_1 rules and does not change the set of subterms below the threshold.

Next we show that if the equivalence proof contains an intermediate sequence of rule applications below the threshold, we can permute the sequence so that all rule applications below the threshold are performed at the end of the proof (after all above-threshold applications). This can be illustrated as follows:



Note that there might be changes in the part B in this transformation (thus we wrote B'), as the number of occurrences of some identical subterms may change and therefore the number of repetitions (at different positions) of certain subparts of the equivalence proof may change. Moreover, the subterms of part C may be different and thus we wrote C'.

Repeated application of this transformation results in an equivalence proof for the equality $f(t_1, \ldots, t_n) \approx_{F_1 \cup F_2} f'(s_1, \ldots, s_m)$ that contains first a sequence of applications above the threshold (that can only contain F_1 rule applications due to what we have shown so far) and a part that contains only rules below the threshold.

First, we show that the steps of part C can precede the steps of part B. The reason is that part B can change only subterms below the threshold: by induction we can show that each rule of part C is still applicable since the changes are at positions that match the variables of the rule. Since we have shown that the above threshold application of rules does not change the set of subterms below the threshold, each subterm affected by part B still exists (though possibly a different number of times) and thus the respective transformations can still be done.

With this transformation, we can conclude the proof, as we have now a sequence of steps from u_1 to some term $u_{l'}$ that contains only above-threshold applications, and therefore rule applications from only F_1, i.e. we have shown $u_1 \approx_{F_1} u_{l'}$ and $u_{l'}$ thus has the form $f'(s'_1, \ldots, s'_m)$ for some $f' \in \Sigma^1$ and suitable s'_i. Moreover, we have a proof with only below threshold applications of $u_{l'} \approx_{F_1 \cup F_2} u_k$. Since all applications are below the threshold, they cannot be applications at the root position (as $f' \in \Sigma^1$) and therefore we have proofs that $s'_i \approx_{F_1 \cup F_2} s_i$ for any $1 \leq i \leq m$. □

This theorem is the basis for constructing an F-unification algorithm for pairs of terms such that one term in each pair is bounded. Consider two terms s and t to F-unify where t is bounded. If at least one of the terms is a variable, say s, the unification problem is handled like unification in the free algebra. Namely, unless t is also a variable, we perform an *occurs check*: if $s \in vars(t)$, then there cannot be an unifier, since $s\sigma \approx_F t\sigma$ implies that $s\sigma$ is a proper subterm of $t\sigma$ and therefore $s\sigma$ has an infinite F-equivalence class. Otherwise, we have the unifier $[s \mapsto t]$. The case that t is a variable and s is not a variable is handled in the same way, but unbounded variables in s must be bounded accordingly.

Consider now the F-unification of two non-variable terms

$$t = f(t_1, \ldots, t_n) \text{ and } s = f'(s_1, \ldots, s_m) \ ,$$

where t is bounded. By Lemma 10.2, it follows that the terms cannot be unified unless f and f' belong to the same disjoint subtheory (in particular, if f is a free symbol, then $f = f'$). Thus consider the case that both f and f' belong to the subtheory F_1. Theorem 10.3 allows us to reduce the problem of finding a complete set of substitutions σ such that

$$t\sigma \approx_F s\sigma$$

to the problem of finding a complete set of substitutions τ such that

$$\begin{array}{lll} t\tau & \approx_{F_1} & f'(x_1, \ldots, x_m)\tau \\ x_i\tau & \approx_F & s_i\tau \quad \text{for all } i \in \{1, \ldots, m\} \end{array}$$

where the x_i are fresh variables.[6]

It is straightforward that this reduction is sound (i.e. it does not introduce substitutions under which s and t are not F-equal). For the completeness, consider any solution σ with $s\sigma \approx_F t\sigma$. By Theorem 10.3, there exist terms $s'_1, \ldots, s'_m$ such that $t\sigma \approx_{F_1} f'(s'_1, \ldots, s'_m)$ and $s'_i \approx_F s\sigma$. Thus, consider $\sigma' = \sigma[x_1 \mapsto s'_1, \ldots, x_m \mapsto s'_m]$. σ' is an instance of some unifier τ of the second problem.

We can thus compute a complete set of F-unifiers for s and t by first computing an F_1-CSTD for t with respect to f' and fresh variables x_i. If available, we can use a specialized

[6]We have used different names σ and τ for the substitutions in the two unification problems, because the substitutions in the second problem may have a larger domain than those in the first problem, containing the fresh variables x_i.

F_1-unification algorithm for this subproblem. Second, if T is the F_1-CSTD, then we recursively solve the unification problem of the subterms for every $\tau_0 \in T$, namely F-unifying each $x_i\tau_0$ with $s_i\tau_0$. Note that the $x_i\tau_0$ are bounded terms, since they result out of a decomposition of a bounded term, if we require that all solutions in the CSTD respect the depth of variables. Observe that this is a generalization of the standard unification algorithm for the free algebra, namely comparing the toplevel symbols f and f' and then recursively unifying the subterms.

It remains to show that the algorithm terminates, in particular, that the problem of unifying the subterms cannot lead into infinite recursion. To that end, we define the weight function for bounded terms

$$w'(t) = \max\{w(t'\sigma) \mid t' \in [t]_F \wedge \mathit{ground}(t\sigma) \wedge \mathit{respect\_depth}(\sigma)\}$$

based on the weight function w defined in the proof of Theorem 6.1. The idea of this weight function is that it computes the weight of the greatest possible ground instance of a bounded term that respects the depth of variables. This is well-defined as there are only finitely many ground substitutions for every bounded term and F-equivalence classes are finite. Consider now the decomposition σ of a bounded term t with respect to f' and fresh variables $x_1, \ldots, x_m$, where σ respects the depth of the variables in t. It holds that

$$w'(t) \geq w'(t\sigma) \geq w'(f'(x_1\sigma, \ldots, x_m\sigma)) > w'(x_1\sigma) + \ldots + w'(x_m\sigma) \ .$$

With respect to w', the weight of the bounded terms in a unification problem is strictly greater than the weight of the bounded terms in every recursive call. Since $w'(t) > 0$, there cannot be an infinite recursion, and thus, the F-unification terminates. This concludes our discussion of F-unification for finite theories F.

10.4 Intruder Deduction Modulo Finite Theories

So far we have considered the problem of unification and matching modulo a finite theory F. We now turn to the intruder deduction problem modulo F, i.e. whether $t \in \mathcal{DY}_F(IK)$ holds for a ground term t and a set of ground terms IK.

Theorem 10.4. *If F is a finite theory, then the problem whether $t \in \mathcal{DY}_F(IK)$ for a term t and a set of terms IK is decidable.*

Proof. Recall that, by definition, $\mathcal{DY}_F(IK) = [\mathcal{DY}^{gen}(IK)]_F$. Therefore, $t \in \mathcal{DY}_F(IK)$ iff there exists $t' \approx_F t$, and $t' \in \mathcal{DY}^{gen}(IK)$. The latter problem (i.e. derivation in the free algebra) is immediate: it holds that $t' \in \mathcal{DY}^{gen}(IK)$ iff $t' \in IK$ or $t' = f(t_1, \ldots, t_n)$, f is intrudable, and $t_i \in \mathcal{DY}^{gen}(IK)$ for all $1 \leq i \leq n$. Since $[t]_{\approx_F}$ is finite, there are finitely many terms t' for which this needs to be checked. □

We now consider a slight generalization of the question, namely when the term to construct contains variables. This is an important question even for a model with only ground terms like IF without the lazy intruder, since IF rules can contain the condition that the intruder generates a particular form of term with variables (while his intruder knowledge IK is ground). We show that this problem is undecidable if we do not bound the variables:

Theorem 10.5. *There is a finite theory F such that it is undecidable, for a term t and a set of ground terms IK, whether there exists a substitution σ such that $t\sigma$ is ground and $t\sigma \in \mathcal{DY}_F(IK)$.*

Proof. We define a finite theory F_{PCP} and show that the PCP problem (see e.g. [90]) can be reduced to derivability modulo F_{PCP} of terms with variables; the undecidability of this problem follows then from the undecidability of the PCP problem. In F_{PCP} occur the function symbols $\langle\cdot,\cdot\rangle$ (concatenation with associativity), $\langle\!\langle \cdot , \cdot \rangle\!\rangle$ (concatenation without associativity), as well as

$\mathsf{res}(\cdot,\cdot,\cdot)$ and $\mathsf{initres}(\cdot)$, which will be used to keep track of partial solutions of the PCP problem:

$$\begin{array}{rcl}
\langle x_1, \langle x_2, x_3\rangle\rangle & \approx & \langle\langle x_1, x_2\rangle, x_3\rangle \\
x_1.x_2 & \approx & x_2.x_1 \\
x_1.(x_2.x_3) & \approx & (x_1.x_2).x_3 \\
\langle x_1,\mathsf{res}(x_2, x_3, \langle x_1,\langle x_4,x_5\rangle\rangle.x_6)\rangle & \approx & \mathsf{res}(\langle x_4, x_2\rangle, \langle x_5, x_3\rangle, \langle x_1,\langle x_4,x_5\rangle\rangle.x_6) \\
\langle x_1,\mathsf{initres}(\langle x_1,\langle x_4,x_5\rangle\rangle.x_6)\rangle & \approx & \mathsf{res}(x_4, x_5, \langle x_1,\langle x_4,x_5\rangle\rangle.x_6)
\end{array}$$

where the x_i are variable symbols. (The intuition behind this theory will become clear below.)

For the encoding of identifiers without bounds, we additionally need the constant $\mathsf{0}$ and the function symbol s (for successor). For simplicity, we will in the following refer to identifiers as $1, \ldots, n$ rather than their actual encoding as terms, namely $\mathsf{s}(\mathsf{0}), \ldots, \mathsf{s}^n(\mathsf{0})$. Similarly, if the underlying alphabet of the PCP is $\{c_1, \ldots, c_n\}$, then the term encoding is also $\mathsf{s}(\mathsf{0}), \ldots, \mathsf{s}^n(\mathsf{0})$, i.e. we do not use disjoint subsets of $\mathcal{T}_\Sigma$ for identifiers and characters over which the PCP problems are built. Let $\mathsf{s}(\cdot), \langle\cdot,\cdot\rangle$ and $\langle\!\langle \cdot, \cdot \rangle\!\rangle$ be the intrudable symbols.

For a given instance $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of the PCP problem, we define the function $symbtab_P$ that maps P to terms over the signature of F_{PCP}:

$$\begin{array}{rcl}
symbtab_P & = & \langle\!\langle 1,\langle\!\langle < x_1 > , < y_1 > \rangle\!\rangle.\ \cdots\ .\langle\!\langle n,\langle\!\langle < x_n > , < y_n > \rangle\!\rangle\rangle\!\rangle \\
< s > & = & \langle c_1, \langle \ldots, c_n\rangle\rangle \text{ for a string } s \text{ of characters } c_1, \ldots, c_n.
\end{array}$$

We first observe that $symbtab_P \approx_{F_{PCP}} \langle\!\langle i,\langle\!\langle x,y\rangle\!\rangle\rangle\!\rangle.s$ iff $x = x_i$ and $y = y_i$ for some $i \in \{1, \ldots, n\}$. Thus, we can use the symbol-table to check whether a certain pair of strings belongs to the PCP problem, and which index it has.

Let now $IK_P = \{\mathsf{initres}(symbtab_P), 1, \ldots, m\}$ for a given PCP P where $\{1, \ldots, m\}$ is the set of constants that identify each pair of strings of P. The central property of our construction is the following. There are ground terms s and t such that

$$\mathsf{res}(s, t, symbtab_P) \in \mathcal{DY}_{Fpcp}(IK_P)$$

iff there are indices $i_1, \ldots, i_k$ with $k > 0$ such that

$$s \approx_{Fpcp} \langle x_{i_1}, \langle \ldots, x_{i_k}\rangle\rangle \text{ and } t \approx_{Fpcp} \langle y_{i_1}, \langle \ldots, y_{i_k}\rangle\rangle\ .$$

In other words, the intruder can derive $\mathsf{res}(s, t, symtab_P)$ iff s and t are a solution of the given PCP problem P.

To show this property, it is immediate that for any sequence $i_1, \ldots, i_k$, with $k > 0$, the intruder can obtain

$$\mathsf{res}(\langle x_{i_1}, \langle \ldots, x_{i_k}\rangle\rangle, \langle y_{i_1}, \langle \ldots, y_{i_k}\rangle\rangle, symtab_P)\ .$$

To show the converse direction of the property, first observe that the intruder does not initially know any term $\mathsf{res}(\cdot,\cdot,\cdot)$ and this symbol is also not intrudable. Thus, the only way to generate a res term is via equivalence in F.

Therefore, we can now reduce any PCP problem P to a symbolic intruder deduction problem modulo F_{PCP}, namely the problem whether there is a substitution σ such that

$$\mathsf{res}(x, x, symbtab_P)\sigma \in \mathcal{DY}_{F_{PCP}}(IK_P)\ .$$

Thus, from the undecidability of the PCP problem, we conclude that the intruder deduction problem modulo F_{PCP} is also undecidable. □

Hence, to decide the intruder deduction problem for terms with variables, we must make further restrictions. By Theorem 10.2, the problem is decidable if t contains only bounded variables.

Chapter 11

Cancelation equations

We now turn to the cancelation equations such as $\{\!|\{\!|x_2|\!\}_{x_1}|\!\}_{x_1} \approx x_2$. Such an equation cannot be formalized as part of a finite theory like F_{ex} since all equivalence classes are infinite. We will therefore consider rewriting for a cancelation theory C modulo a finite theory F, namely the relation $\rightarrow_{C/F}$ defined in Subsection 3.5.1. Recall that cancelation theories are induced by rules of the form $l \rightarrow r$ with $vars(l) \supseteq vars(r)$ according to Definition 10.1.

As mentioned, the principal property that our framework requires is that we consider theories F and C such that the relation $\rightarrow_{C/F}$ is convergent. Convergence is the key to decide the word problem modulo $F \cup C$ as we will see now.

Theorem 11.1. *Let F be a finite theory and C a cancelation theory, and let $\rightarrow_{C/F}$ be convergent. Then the ground word problem for $F \cup C$ is decidable.*

Proof. Since $\rightarrow_{C/F}$ is convergent, $t_1 \approx_{F \cup C} t_2$ holds iff $[t_1]_{\approx_F}\downarrow_{C/F} = [t_2]_{\approx_F}\downarrow_{C/F}$ holds. The normal-form is computable as follows. Let t be a term for which we want to compute $[t]_{\approx_F}\downarrow_{C/F}$. For every term $t' \in [t]_{\approx_F}$, we check whether t' is a redex with respect to $\rightarrow_C$. This check is possible since F is finite, every term has finitely many subterms and F-matching is decidable without bounds in the variables the rules of C. If no $t' \in [t]_{\approx_F}$ is a redex, then $[t]_{\approx_F}$ is already in normal-form with respect to $\rightarrow_{C/F}$, otherwise, we pick any term s such that $t' \rightarrow_C s$ (for some $t' \in [t]_{\approx_F}$) and compute the normal-form of $[s]_{\approx_F}$ with respect to $\rightarrow_{C/F}$. The convergence of $\rightarrow_{C/F}$ tells us that a normal-form is reached after finitely many reductions and that the normal-form does not depend on the choice of the reductions, if there are several. Finally, to compare two normal-forms $[t]_{\approx_F}$ and $[s]_{\approx_F}$ (given by two possibly different representatives t and s), we simply check whether $t \approx_F s$. □

By Theorem 10.2, it follows that we can construct a unification algorithm modulo $F \cup C$ for terms with bounded variables (on both sides). In particular, this implies that the unifiability problem modulo $F \cup C$ for terms with bounded variables is decidable.

The proof of Theorem 11.1 relies on F-matching with unbounded variables (otherwise we would have to bound the variables in C). Thus, we cannot extend this algorithm to $C \cup F$-matching with unbounded variables, since this would lead to F-unification problems with unbounded variables, which is in general not recursive as mentioned in Section 10.1. Indeed, it is straightforward to show that the $F \cup C$-matchability is undecidable:

Theorem 11.2. *There is a finite theory F and cancelation theory C, where $\rightarrow_{C/F}$ is convergent, for which $F \cup C$-matchability is undecidable.*

Proof. Consider any finite theory F for which unifiability is undecidable, e.g. the theory D_A introduced in Chapter 10. Consider $C = \{\mathsf{equal}(x, x) \approx \mathsf{true}\}$ for two symbols equal and true that do not occur in F. It is straightforward that $\rightarrow_{C/F}$ is convergent. For these theories, we can reduce F-unifiability to $C \cup F$-matchability, namely $t\sigma \approx_F s\sigma$ iff $\mathsf{true} \approx_{C \cup F} \mathsf{equal}(t, s)\sigma$. As F-unifiability is undecidable, thus also $F \cup C$-matchability is undecidable. □

This theorem justifies our requirement that for matching and unification modulo $F \cup C$, all variables must be bounded.

11.1 Cancelation as Analysis

The results that we have presented so far allow us to decide, for ground terms or terms with bounded variables, the equality of terms modulo a finite theory F and a cancelation theory C, as well as the intruder deduction problem in the theory F. We now consider how to solve the intruder deduction problem in the theory $F \cup C$. In Section 11.2, we will see that this problem is in general undecidable, so to obtain a decidable problem we must further restrict our model: we bound the number of operations that the intruder can perform.

The idea that we put forth here to solve the intruder deduction problem with respect to $F \cup C$ is to distinguish generation (or composition) and analysis (or decomposition) of messages by the intruder. As we have seen already in Part II, generation and analysis are not completely independent; for instance, if the intruder knows the messages $\{|\mathsf{m}|\}_{\langle \mathsf{k}_1,\mathsf{k}_2\rangle}$ and k_1 and k_2, then he can analyze the encrypted message, but only after composing the key $\langle \mathsf{k}_1, \mathsf{k}_2\rangle$. We now define a general notion of analysis based on an arbitrary cancelation theory C.

Intuitively, we speak of *generation* when the intruder applies the rule G_{comp} to compose terms, excluding the case when the resulting composed term is a redex according to the cancelation theory C (as we can then reduce it to a simpler term). We speak of *analysis* when the intruder applies the rule G_{comp} to obtain a redex whose normal-form cannot be composed from his current knowledge. We can then formalize the notion of the intruder knowledge being completely analyzed based on the notion of cancelation rules present in our framework: we say that the intruder has analyzed his knowledge as far as possible if, by applying the cancelation rules, the intruder can only derive messages (except redices in $\rightarrow_{C/F}$) that he can also derive without cancelation rules. Formally:

Definition 11.1. *Let C be a cancelation theory convergent modulo a finite theory F. We say that a finite set of ground terms IK is* analyzed with respect to C modulo F *if $[t]_{\approx_F}\downarrow_{C/F} \subseteq \mathcal{DY}_F(IK)$ for each $t \in \mathcal{DY}_F(IK)$.*

Example 11.1. Consider again F_{ex} and C_{ex}. The set $IK = \{\{|\mathsf{m}|\}_\mathsf{k}, \mathsf{k}\}$ is not analyzed with respect to C_{ex} modulo F_{ex} as the intruder can generate $t = \{|\{|\mathsf{m}|\}_\mathsf{k}|\}_\mathsf{k} \in \mathcal{DY}_{F_{ex}}(IK)$, and $[t]_{\approx_{F_{ex}}}\downarrow_{C_{ex}/F_{ex}} = [\mathsf{m}]_{\approx_{F_{ex}}}$, but $\mathsf{m} \notin \mathcal{DY}_{F_{ex}}(IK)$. In contrast, $IK' = IK \cup \{\mathsf{m}\}$ is analyzed since all messages that can be obtained only by normalizing terms in $\mathcal{DY}_{F_{ex}}(IK')$ are already contained in $\mathcal{DY}_{F_{ex}}(IK')$. □

We thus have a characterization of analyzed intruder knowledge as a set that contains all messages that can be derived under $\mathcal{DY}_{F\cup C}(\cdot)$ and but not under $\mathcal{DY}_F(\cdot)$. The idea is that when the set of messages known by the intruder is analyzed, then there is no need to consider the cancelation theory in the derivations of the intruder. Hence we can decide the intruder deduction problem $\mathcal{DY}_{F\cup C}(\cdot)$ when the intruder knowledge is analyzed:

Theorem 11.3. *Let F be a finite theory and C a cancelation theory, and let $\rightarrow_{C/F}$ be convergent. Further, let t be a ground term and IK be a finite set of ground terms analyzed with respect to C modulo F. Then it is decidable whether $t \in \mathcal{DY}_{F\cup C}(IK)$.*

Proof. Since IK is analyzed, this problem is equivalent to the problem $[t]_{\approx_F}\downarrow_{C/F} \subseteq \mathcal{DY}_F(IK)$. By Theorem 11.1, we can effectively compute $[t]_{\approx_F}\downarrow_{C/F}$. By Theorem 10.4, we can check whether $t \in \mathcal{DY}_F(IK)$ (which is equivalent to $[t]_F \subseteq \mathcal{DY}_F(IK)$). Thus, $t \in \mathcal{DY}_{F\cup C}(IK)$ is decidable. □

By Theorem 10.2, it follows that the intruder deduction problem is decidable for terms with bounded variables when the intruder knowledge is analyzed.

$$\frac{}{t \in \mathcal{DY}^k(IK)}\ G^k_{\text{axiom}}\ (t \in IK\ ,\ k \geq 0)$$

$$\frac{t_1 \in \mathcal{DY}^k(IK) \quad \cdots \quad t_n \in \mathcal{DY}^k(IK)}{f(t_1, \ldots, t_n) \in \mathcal{DY}^{k+1}(IK)}\ G^k_{\text{comp}}\ (\mathit{intrudable}(f))$$

Figure 11.1: Deduction rules of the k-bounded intruder model (without algebraic properties).

11.2 Undecidability of Analysis

The previous method for solving the intruder deduction problem is restricted to the case where the intruder knowledge is analyzed. The central question thus is how to transform an arbitrary intruder knowledge into an analyzed one.

Theorem 11.4. *There is a finite theory F and a cancelation theory C, where $\to_{C/F}$ is convergent, such that it is undecidable whether a finite set of ground terms IK is analyzed with respect to C modulo F. Moreover, the intruder deduction problem $t \in \mathcal{DY}_{F \cup C}(IK)$ is also undecidable.*

Note that [1, 2] have shown that the intruder deduction problem in a theory E can be undecidable even if unifiability in E is decidable. Our theorem is incomparable to this result as it does not require that unifiability of E is decidable.

Proof. We consider again the finite theory F_{PCP} defined in the proof of Theorem 10.5. Let $C_{PCP} = \{\mathsf{f}(\mathsf{res}(x, x, y)) \approx \mathsf{secret}\}$ for some new unintrudable constant secret and a new intrudable function symbol f that we need for convergence.

We first show convergence. Since f is a new function symbol, there cannot be a conflict between two instances of the single rule of C_{PCP} in a non-variable position, and similarly there cannot be a conflict with F_{PCP} in a non-variable position. As obviously $\to_{C_{PCP}/F_{PCP}}$ terminates, we conclude that it is convergent.

Now, by the proof of Theorem 10.5, we know the intruder can generate a ground instance of the term $\mathsf{res}(x, x, y)$ from IK_P if and only if the given PCP P has a solution. Thus the intruder can derive the constant secret from IK_P iff the PCP P has a solution. Also, the intruder knowledge IK_P is analyzed with respect to C_{PCP} modulo F_{PCP} iff the intruder cannot derive secret. Thus, by the undecidability of the PCP problem, both the intruder deduction problem modulo $F_{PCP} \cup C_{PCP}$ and problem whether the intruder knowledge is analyzed with respect to C_{PCP} modulo F_{PCP} are undecidable. □

We thus need to make further restrictions to obtain a general algorithm for analyzing the intruder knowledge. We proceed by limiting the operations that the intruder can perform when analyzing a single message (i.e. the number of steps before he obtains a new redex). We define a bounded derivation of the intruder as follows:

Definition 11.2. *Given a finite set IK of ground terms, we define the* k-bounded intruder model *as the least set $\mathcal{DY}^k(IK)$ closed under the rules in Figure 11.1. Additionally, for a set E of equations, we define*

$$\mathcal{DY}^k_E(M) = [\mathcal{DY}^k(M)]_{\approx_E}\ .$$

Let F be a finite theory and C a cancelation theory, and let $\to_{C/F}$ be convergent. Given a constant $k \in \mathbb{N}$, we say that the intruder knowledge IK, which is a finite set of ground terms, is k-analyzed *(with respect to C modulo F) iff $[t]_{\approx_F}\!\downarrow_{C/F} \subseteq \mathcal{DY}^k_F(IK)$ for each $t \in \mathcal{DY}^k_F(IK)$.*

Note that this definition limits only the intruder deduction steps, not the derivation of equivalent terms due to the theory E.

Theorem 11.5. *Let F be a finite theory and C a cancelation theory, let $\rightarrow_{C/F}$ be convergent, and let $k \in \mathbb{N}$. Then it is decidable if a finite set of ground terms IK is k-analyzed (with respect to C modulo F).*

Proof. The set $IK' = \mathcal{DY}^k_F(IK)$ is finite, since F is a finite theory and all compositions are bounded by k. We can then check if any message of $t \in IK'$ has a normal-form $[t]_{\approx_F}\downarrow_{C/F}$ that is not contained in IK'. □

Note, however, that given a finite set of ground terms IK, there does not always exist a finite superset IK' of ground terms that is (k-)analyzed. Consider, for example, the theories $F = \{\mathsf{f}(x) \approx \mathsf{g}(\mathsf{h}(x))\}$ and $C = \{\mathsf{g}(x) \approx x\}$. F is a finite theory, C is a cancelation theory, and $\rightarrow_{C/F}$ is convergent. Furthermore, let f be intrudable, and all other symbols be unintrudable. Finally, let IK be a finite set of ground terms that contains a constant c. We then have, for instance, that $\mathsf{h}(\mathsf{c}), \mathsf{h}(\mathsf{h}(\mathsf{c})), \ldots \in \mathcal{DY}_{F\cup C}(IK)$. Thus, there is no finite set $IK' \supseteq IK$ such that IK' is analyzed. For the bounded case, observe that $\mathsf{g}(t) \in \mathcal{DY}^k_{F\cup C}(IK \cup t)$ for any ground term t and $k \geq 1$. Thus, any k-analyzed superset of IK must also contain $\mathsf{g}^n(\mathsf{c})$ for any $n \in \mathbb{N}$, so it must be infinite. Hence, to complete our framework, we must be able to check bounded derivability without first computing an analyzed intruder knowledge. The following theorem tells us that this is possible:

Theorem 11.6. *Let F be a finite theory and C a cancelation theory, let $\rightarrow_{C/F}$ be convergent, and let $k \in \mathbb{N}$. Then it is decidable if a ground term t can be derived from a finite set of ground terms IK, i.e. whether $t \in \mathcal{DY}^k_{F\cup C}(IK)$.*

Proof. By definition, $\mathcal{DY}^k_{F\cup C}(IK) = [\mathcal{DY}^k(IK)]_{\approx_{F\cup C}}$. Further, $\mathcal{DY}^k(IK)$ is a finite set for any k. Thus, we can reduce the problem $t \in \mathcal{DY}^{F\cup C}_k(IK)$ to finitely many word problems modulo $F \cup C$ (which are decidable due to Theorem 11.1). □

With the deduction problem modulo $F \cup C$ for bounded intruder derivations, we have completed the framework for algebraic intruder deductions. We will now turn to the question how to integrate these results into checking IF protocol descriptions modulo $F \cup C$. In particular, we will clarify what role the restrictions play that we have introduced within our framework.

11.3 Integration into IF

Recall that in Theorem 10.5, we have shown that neither the safety nor the insafety of IF protocol descriptions modulo algebraic theories E are semi-decidable. Note that this proof did not require an unbounded number of sessions, but simply exploited the fact that IF allows for negative conditions in transition rules, and the word problem modulo E is undecidable in general.

We now consider the case of a theory $E = F \cup C$ where F is finite, C is a cancelation theory, and $\rightarrow_{C/F}$ is convergent. Using our framework, we show that we can build a semi-decision algorithm for insafety of IF protocol descriptions modulo $C \cup F$ (even for an unbounded number of sessions), while safety cannot be not semi-decidable (even for a bounded number of sessions).

Theorem 11.7. *Let F be a finite theory and C be a cancelation theory such that $\rightarrow_{C/F}$ is convergent. Then the insafety of IF protocol descriptions modulo $F \cup C$ is semi-decidable.*

Proof. It suffices to show that for every IF rule r and ground state S, the set of states T with $S \Rightarrow_{r/(F\cup C)} T$ is enumerable, because then it follows that the set of reachable states modulo $F \cup C$ (using all transition rules and the attack rules) is enumerable and filtering these states for containment of the attack predicate gives a semi-decision algorithm for IF insafety.

Let thus S be a state (which is a set of ground facts) and $r = L \,{-}[V]{\Rightarrow}\, R$ be an IF rule. Let further P be the set of positive facts of L without the iknows facts. Let M be the set of message terms m such that $\mathsf{iknows}(m) \in L$, let N be the set of negative facts of L, and let ϕ be the set of conditions of L.

It is immediate that the set of ground substitutions σ with $dom(\sigma) = vars(L)$ is enumerable. Consider one such ground substitution σ. Since the ground word problem modulo $F \cup C$ is decidable by Theorem 11.1, we can check that $P\sigma$ is contained in S modulo $F \cup C$, and that no fact of $N\sigma$ is contained in S modulo $F \cup C$. The conditions of $\phi\sigma$ are conjunctions and disjunctions of inequalities (interpreted again modulo $F \cup C$). Thus by the decidability of the word problem modulo $F \cup C$, we can also decide whether $\sigma \models_{F \cup C} \phi$.

What remains to check is that $M\sigma \subseteq \mathcal{DY}_{F \cup C}(IK)$ where $IK = \{m \mid \mathsf{iknows}(m) \in S\}$. We know by Theorem 11.4 that this is not decidable. However, since $M\sigma$ is ground, this question is semi-decidable by Theorem 11.6, namely by checking $M\sigma \subseteq \mathcal{DY}^k_{F \cup C}(IK)$ for $k \in \{1, 2, \ldots\}$.

Interleaving the enumeration for ground substitutions with the enumeration of a bound k for deduction steps gives an enumeration algorithm for all ground substitutions such that also $M\sigma$ can be deduced modulo $F \cup C$. Generating successor states for the respective rule matches σ turns this into an enumeration algorithm for the successors of S with rule r.

Therefore, we have obtained a semi-decision algorithm for IF insafety modulo $F \cup C$. □

We now show that from the constructive proof of Theorem 11.7, we can obtain a decision algorithm for the safety of an IF protocol description modulo $F \cup C$ under the following restrictions:

- All variables in the IF rules and attack rules are bounded; this is for instance the case when using a typed-model as described in Subsection 4.4.4.
- Intruder deduction is bounded, i.e. replacing $\mathcal{DY}_{F \cup C}(IK)$ with $\mathcal{DY}^k_{F \cup C}(IK)$ in the IF semantics for a given constant k.
- The transition relation $\Rightarrow_{R/(C \cup F)}$ is terminating, i.e. we have a bounded number of sessions.

The idea is that under these restrictions, the set of successors of a state is finite: if all variables are bounded, there can only be finitely many ground substitutions for the variables in the rules. With the bound on the intruder deduction, it is decidable by Theorem 11.6, whether all terms in left-hand side iknows facts of a rule can be generated by the intruder. Since every state has finitely many successors and the transition relation $\Rightarrow_R$ terminates, there are finitely many reachable states. From that, we obtain a decision algorithm for safety of IF protocol descriptions, namely by checking whether any state contains the fact attack.

Dropping the restriction to a bounded number of reduction steps immediately leads to undecidability of the safety problem:

Theorem 11.8. *Let F be a finite theory and C a cancelation theory such that $\rightarrow_{C/F}$ is decidable. Safety for IF protocol descriptions modulo $F \cup C$ is undecidable, even under the restriction that the number of sessions and all variables of the protocol description are bounded.*

Proof. We reduce the problem of intruder deduction modulo $F \cup C$ to safety of IF protocol descriptions modulo $F \cup C$. To an intruder deduction problem (t, IK), where t is a ground term and IK is a set of ground terms, we assign an IF protocol description $\Pi_{(t,IK)}$ which has the initial state $\{\mathsf{iknows}(m) \mid m \in IK\}$, an empty set of rules, and the attack rule $\{\mathsf{iknows}(t)\}$. It holds that $t \in \mathcal{DY}_{F \cup C}(IK)$ iff $\Pi_{(t,IK)}$ has an attack. Therefore a decision algorithm for IF protocol security modulo $F \cup C$ (even with the restriction to bounded variables and sessions) implies a decision algorithm for the intruder deduction problem modulo $F \cup C$, contradicting Theorem 11.4. □

The theorems we have presented in this section show the relation of the restrictions of our framework and the decidability and semi-decidability of IF protocol descriptions in detail. The bounding of variables for several problems corresponds to bounding the variables in the protocol description (not in the equations of the algebraic theory). This bounding is a generalization of typed models in protocol analysis as described in Subsection 4.4.4. The restriction to a bounded number of deduction steps of the intruder is an additional restriction that we need for deciding the intruder deduction problem. When removing this restriction, the insafety problem for IF protocol descriptions modulo $F \cup C$ is undecidable, even if we limit the number of sessions and bound all variables. This concludes our exposition of the framework for algebraic intruder deductions and its integration into IF.

11.4 The Lazy Algebraic Intruder

With the framework for algebraic intruder deduction that we have proposed, we have a method to build a semi-decision algorithm for IF insecurity modulo an algebraic theory for a large class of theories, namely cancelation theories that are convergent modulo a finite theory. Moreover, with the restriction to bounded sessions and intruder deductions, we obtain a decision algorithm for this problem.

A prototype implementation of this framework is now built into OFMC, allowing the user to specify an algebraic theory as part of the verification problem. To that end, our framework has been integrated with the symbolic lazy intruder technique that we have introduced in Part II. We now sketch the basic ideas for this combination, leaving a precise, formal argumentation for future work.

11.4.1 Using Finite Theory Decomposition

By Theorem 10.2, we already have a generic way to lift ground problems to problems over variables, as long as all variables are bounded. This argument is however based on a complete enumeration, while the lazy intruder is a constraint-based technique to avoid precisely this enumeration. Therefore, Theorem 10.2 does not provide an efficient basis for combining our algebraic intruder deduction framework with the lazy intruder.

However, recall that Theorem 10.3 provides the basis for a more efficient F-unification algorithm with one-side bounded variables. This approach also allows for the integration of specialized algorithms for disjoint subtheories of F. The idea is now that decomposition can be used not only for unification but also for the intruder deduction problem modulo F. Suppose $F = F_1 \cup \ldots \cup F_n$ for disjoint subtheories F_i. Then for the problem

$$f(t_1, \ldots, t_n)\sigma \in \mathcal{DY}_F(IK)$$

we can use a similar decomposition as follows:

- Check whether $f(t_1, \ldots, t_n)$ can be directly F-matched with any term in IK.
- Let F_i be the subtheory to which f belongs. For every intrudable function symbol f' that belongs to F_i, we compute the F_i-CSTD of $f(t_1, \ldots, t_n)$ with respect to f' and fresh variable symbols $x_1, \ldots, x_m$ (where m is the arity of f'), i.e. a complete set of substitutions τ such that
$$f(t_1, \ldots, t_n)\tau \approx_{F_i} f'(x_1, \ldots, x_m)\tau \ .$$
This can be done using a specialized algorithm for F_i-unification when available.
- For each τ in the CSTDs, check recursively that each of the subterms can be constructed, i.e. that for each $i \in \{1, \ldots, m\}$ it holds that
$$x_i\tau \in \mathcal{DY}_F(IK)$$

Note that this algorithm, like our F-unification algorithm, may not terminate in general, unless the variables in the decomposed terms are bounded.

Considering the Red^{gen} algorithm introduced in Definition 6.5. The above idea can be integrated into this algorithm by replacing unification with F-unification, and the function $generate(t)$ with the F_i-CSTD problem for all intrudable symbols f' that belong to the same theory as the root symbol of t. We thus obtain here a generalization of the algorithm for the free algebra, since for a free symbol, the obtained subterms are only the syntactic subterms of t. From this, we obtain a lazy intruder reduction algorithm modulo finite theories.

11.4.2 Cancelation Theories

There are two aspects where cancelation theories play a role: Firstly, there are no analysis rules, but decryption by the intruder is modeled by cancelation theories; thus, analyzing the intruder knowledge requires finding terms that can be composed (using "decryption" operations) and that form a redex whose normal-form is not yet contained in the intruder knowledge. Secondly, the transition rules of the honest agents can produce new redices.

For symbolic terms, the rewriting problem turns into a *narrowing* problem. Consider that variables may be substituted by a term that is a redex, and even if we require that substituted terms are normal-forms, the substitution can make the term a redex. For instance in E_{ex}, the term $\{|\mathsf{X}|\}_\mathsf{k}$ becomes a redex under the substitution $\sigma = [\mathsf{X} \mapsto \{|\mathsf{X}'|\}_\mathsf{k}]$ (the normal-form then being X').

If all variables are bounded, we can also bound the number of such "implicit" redices that can occur in this way. More precisely, for situations of potential redices like $\{|\mathsf{X}|\}_\mathsf{k}$, we perform a case split as follows. The first case is that the term is X' performing the substitution σ (and this is a normal-form if we forbid the substitutions of a variable with a redex). The other case is simply to stick with $\{|\mathsf{X}|\}_\mathsf{k}$ but the additional inequality $\forall \mathsf{X}'.\mathsf{X} \not\approx_E \{|\mathsf{X}'|\}_\mathsf{k}$.

Note that we have introduced here a fresh variable X' which must be universally quantified. This means an extension of our handling of inequalities, namely with universally quantified variables. This extension is implemented as follows. We use the same check as before, using fresh intrudable constants for all *free* variables of an inequality, and then instead of an equality check (modulo E) we perform a unification modulo E; there is no substitution for the universally quantified variables under which the inequalities are satisfied iff there is no such unifier.

In order to avoid symbolic normalization of every term that is created, we use a transformation on the rules of the honest agents that performs the necessary case splits on the rules, i.e. only once before the search. For instance, an honest agent rule of the form

$$\mathsf{iknows}(\mathsf{X}).\mathsf{state}.(\cdot) \Rightarrow \mathsf{iknows}(\{|\mathsf{X}|\}_\mathsf{k}).\mathsf{state}.(\cdot)$$

leads to the case split (as above) of the following two rules:

$$\mathsf{iknows}(\mathsf{X}).\mathsf{state}.(\cdot) \wedge \forall \mathsf{X}'.\, \mathsf{X} \neq \{|\mathsf{X}'|\}_\mathsf{k} \Rightarrow \mathsf{iknows}(\{|\mathsf{X}|\}_\mathsf{k}).\mathsf{state}.(\cdot)$$
$$\mathsf{iknows}(\{|\mathsf{X}'|\}_\mathsf{k}).\mathsf{state}.(\cdot) \Rightarrow \mathsf{iknows}(\mathsf{X}').\mathsf{state}.(\cdot)$$

The state fact in these rules must also be updated accordingly if they contain the variable X.

This takes care of redices that may be produced by the honest agents. We can statically perform a similar operation on the rules that describe the cryptographic abilities of the intruder, namely that he can compose $f(t_1, \ldots, t_n)$ for every known term t_i and intrudable operator f of arity n. For instance, for $f = \{|\cdot|\}.$, we get the standard decryption rule A_{scrypt}. The other rule that comes out of this process is the standard composition rule for scrypt, with the additional restriction that subterms cannot be of a certain form. We chose not to impose this restriction on the intruder generation rules, but filter results later for reducible terms.

By this normalization of the intruder deduction rules, we obtain a set of symbolic analysis rules for the intruder knowledge, which allows us to implement an analysis procedure for the symbolic intruder knowledge, avoiding to enumerate all terms that the intruder can compose to obtain a new redex. To enforce termination, this analysis procedure needs to bound the number of deduction steps in the same way as introduced in Section 11.2.

Based on these ideas, we have implemented the lazy algebraic intruder in OFMC. The entire unification and intruder deduction of OFMC are thus parameterized over the theory, which the user can specify by a *theory file* with which OFMC is invoked, along with the protocol to analyze.

Chapter 12

Application to Modeling Dictionary Attacks

In this chapter we demonstrate that algebraic properties, and in particular the explicit decryption that is possible in an algebraic model, are a good basis for modeling *dictionary attacks*. A guessing, or dictionary, attack means that the intruder exploits the low entropy in data like passwords. Consider a protocol like Microsoft's CHAPv2 (see Figure 12.1): it contains messages like $\mathsf{f}(\mathsf{Pw}, \mathsf{Na}, \mathsf{Nb})$ where Pw is a (possibly poorly chosen) password, and Na and Nb are two nonces that are transmitted in clear-text with this hash-value. As the intruder can see the nonces and knows how the hash-function f works, he can build hash-values for different guesses of the value Pw which he does not yet know; in particular, he can efficiently try out for Pw every entry from a *dictionary* of popular passwords. Suppose that the intruder has observed the hash-value constructed by an honest agent with the correct password, and has found a dictionary entry that would give the same hash-value, then he has a high probability that this dictionary entry is the correct password. (He can of course increase his chance by checking this for several sessions and thus different values of the nonces.) As this kind of guessing attack is not part of the interaction with other agents, we call this an *off-line* guessing attack, and we focus on off-line guessing here.

One may wonder, whether guessing attacks are really interesting to be considered in formal analysis. Indeed our point is not to identify a large variety of protocols, e.g. Kerberos, as weak protocols just because guessing attacks are possible when using bad passwords. Actually that is quite easy to see from the form of messages, and would not require a formal analysis. The point is rather that there are several protocols like the *Encrypted Key Exchange* (EKE) [31] and its successors, or the *Secure Remote Passwords* (SRP) Protocol [131, 132], which is discussed in Section 17.3, that are explicitly designed to be resistant against guessing, so they should provide their security guarantees even when predictable passwords are used. To analyze such protocols appropriately, we need a clear notion of what guessing means, in particular, what additional abilities the intruder has if he can guess passwords.

Previous attempts at formalizing off-line guessing have been proposed to formally analyze protocols under the consist of extending a Dolev-Yao-style intruder model with inference rules to capture the additional capabilities of the intruder concerning guessable messages [55, 59, 61, 67, 100]. While it is easy to convince oneself that the proposed rules are correct, in the sense that an intruder can actually perform such "guessing steps", it is difficult to see whether such a system of inference rules is complete in the sense that it captures all the kinds of attacks that we would intuitively call "guessing attacks". As a consequence, these systems are helpful to discover some off-line guessing attacks but are not fully appropriate for formalizing off-line guessing and obtaining positive results, i.e. verifying that a given protocol is not vulnerable to guessing attacks.

We propose that there is a fairly simple intuition underlying off-line guessing, which captures a generic class of off-line guessing attacks. We formalize this intuition based on the idea of explicitly representing the intermediate states of the computation of off-line guessing attacks. We call these

1. $\mathsf{A} \rightarrow \mathsf{S}:\ \mathsf{A}$
2. $\mathsf{S} \rightarrow \mathsf{A}:\ \mathsf{Ns}$
3. $\mathsf{A} \rightarrow \mathsf{S}:\ \mathsf{Na}, \mathsf{H}(\mathsf{Pw}, \mathsf{Na}, \mathsf{Ns}, \mathsf{A})$
4. $\mathsf{S} \rightarrow \mathsf{A}:\ \mathsf{H}(\mathsf{Pw}, \mathsf{Na})$

Figure 12.1: The MS-CHAPv2 Protocol

intermediate computation states *maps*. Intuitively, a map is the result of the intruder constructing a message from his knowledge and inserting, in place of an unknown value, every value from the dictionary he uses to perform the attack; it thus maps candidate values for guessable data to the concrete message that would result in the respective cases.

12.1 Intuition

First observe that when it comes to guessing attacks, we talk about a slightly different notion of knowledge than it is standard in the Dolev-Yao model. For instance, the intruder may observe a message that is encrypted with a poorly chosen password that appears in a dictionary that the intruder possesses; so, roughly speaking, he "knows" the password. However, if we assume that the password is simply part of the intruder knowledge, then, according to the Dolev-Yao model, the intruder can decrypt everything encrypted with that password. This is, however, not the case in reality: although the intruder possesses a dictionary that contains the password, he does not know a priori which entry in the dictionary it is, and it might be impossible to identify the right entry. In other words, not distinguishing between guessable and known messages in the Dolev-Yao model is like assuming that the intruder "always guesses correctly".

Our intuition of guessing has to do with the operations that the intruder can perform on every entry in his dictionary so that he can uniquely determine one particular entry of the dictionary (which then represents the "correct" guess). Our idea to formalize off-line guessing attacks is to formalize this intuition by making explicit the intermediate states of such a computation, which we express via *maps*. Roughly speaking, a map relates each entry of the dictionary (or n-tuples thereof) to the outcome of a sequence of operations under this guess (or these guesses). The precise formulation is given in Definition 12.1.

Figure 12.2 shows three examples of maps, where the intruder's dictionary contains the entries $\{d_1, \ldots, d_n\}$.[1] The first is the map for the example $\{\!|\mathsf{m}|\!\}_{\mathsf{pw}}$, where pw appears in the dictionary, i.e. $d_c = \mathsf{pw}$ for the "correct" entry $c \in \{1, \ldots, n\}$: the intruder encrypts the known message m with each of the d_i. Only one of the entries $\{\!|\mathsf{m}|\!\}_{d_i}$ corresponds to the original message $\{\!|\mathsf{m}|\!\}_{\mathsf{pw}}$, and this entry reveals the correct password $\mathsf{pw} = d_c$. The second example demonstrates that the concept of maps can handle multiple guesses as well: if the intruder knows a term $\{\!|\mathsf{pw}'|\!\}_{\mathsf{pw}}$ which consists of a guessable message encrypted with a guessable key-term, then the intruder can simply build every encryption of an entry in his dictionary with an entry in his dictionary.

Before we turn to the third example displayed in Figure 12.2, we need to consider in more detail one of the simplifying assumptions mentioned above: what happens when encrypted messages are decrypted with a wrong key.

12.1.1 Decryption with a Wrong Key

In a Dolev-Yao intruder model without guessing, it is not necessary to consider what happens when decrypting messages with a wrong key; this is because the intruder knows whether he knows a particular key. Under the perfect cryptography assumption, he will thus not even attempt to decrypt messages for which he does not know the proper key. Also, we can assume that the

[1] The d_i are in fact meta-variables for the terms in the dictionary and thus not set in sans-serif.

v	$\{\!\lvert \mathsf{m} \rvert\!\}_v$
d_1	$\{\!\lvert \mathsf{m} \rvert\!\}_{d_1}$
d_2	$\{\!\lvert \mathsf{m} \rvert\!\}_{d_2}$
$\vdots$	$\vdots$
d_n	$\{\!\lvert \mathsf{m} \rvert\!\}_{d_n}$

v_1	v_2	$\{\!\lvert v_2 \rvert\!\}_{v_1}$
d_1	d_1	$\{\!\lvert d_1 \rvert\!\}_{d_1}$
d_1	d_2	$\{\!\lvert d_2 \rvert\!\}_{d_1}$
$\vdots$	$\vdots$	$\vdots$
d_n	d_n	$\{\!\lvert d_n \rvert\!\}_{d_n}$

v	$\pi_1(\{\!\lvert\{\!\lvert \mathsf{m_1}, \mathsf{m_2} \rvert\!\}_{\mathsf{pw}}\rvert\!\}_v)$
d_1	$\pi_1(\{\!\lvert\{\!\lvert \mathsf{m_1}, \mathsf{m_2} \rvert\!\}_{\mathsf{pw}}\rvert\!\}_{d_1})$
d_2	$\pi_1(\{\!\lvert\{\!\lvert \mathsf{m_1}, \mathsf{m_2} \rvert\!\}_{\mathsf{pw}}\rvert\!\}_{d_2})$
$\vdots$	$\vdots$
d_n	$\pi_1(\{\!\lvert\{\!\lvert \mathsf{m_1}, \mathsf{m_2} \rvert\!\}_{\mathsf{pw}}\rvert\!\}_{d_n})$

Figure 12.2: Three examples of maps

intruder will send to honest agents only messages that they can receive, i.e. messages that are encrypted with the keys that the honest agents will use for decryption, since the intruder will not gain anything from a situation where an agent reads some random bits after decryption of incoming messages. When considering guesses, however, it is important to model what happens if the intruder decrypts messages with a wrong key, for instance because he might simply try out every entry of his dictionary as the decryption key.

In Section 3.5, we have introduced algebraic theories which allow for avoiding analysis rules entirely and rather use explicit decryption operations with the property that encryption and decryption cancel each other out. For instance, in the algebraic example theory E_{ex}, we have the property

$$\{\!\lvert\{\!\lvert x_2 \rvert\!\}_{x_1}\rvert\!\}_{x_1} \approx x_2 \ .$$

This expresses that when we decrypt with the correct key, we obtain a term that is equivalent to the original clear-text; with a wrong key used for decryption, however, we obtain a term that cannot be simplified, representing the "garbage" that comes out of such an operation in reality when the wrong key is applied.

More formally, for all terms k, k', and m, it holds that $\{\!\lvert\{\!\lvert m \rvert\!\}_k\rvert\!\}_{k'} \approx_E m$ iff $k \approx_E k'$. This is exactly what we want, as the intruder can now attempt the decryption with several keys, which results in the original message m only in the case that he used the right key. In all other cases, he obtains a term of the form $\{\!\lvert\{\!\lvert m \rvert\!\}_k\rvert\!\}_{k'}$, which represents some random bits he obtained after decryption, but which he is not a priori able to distinguish from the "real" message m.[2]

We can now return to the third example of Figure 12.2, where the intruder tries to find out the guessable password pw of the message $\{\!\lvert\langle \mathsf{m_1}, \mathsf{m_2}\rangle\rvert\!\}_{\mathsf{pw}}$ where he knows the message m_1. Recall that we denote with π_1 and π_2 the projections to the first and second component of a pair. The intruder builds a map in two steps: first he encrypts the given message $\{\!\lvert\langle \mathsf{m_1}, \mathsf{m_2}\rangle\rvert\!\}_{\mathsf{pw}}$ with every entry of the dictionary, and then he builds the projection to the first component. Only for the correct guess, i.e. the entry c with $d_c = \mathsf{pw}$, the resulting message has the property

$$\pi_1(\{\!\lvert\{\!\lvert\langle \mathsf{m_1}, \mathsf{m_2}\rangle\rvert\!\}_{\mathsf{pw}}\rvert\!\}_{d_c}) \approx \pi_1(\langle \mathsf{m_1}, \mathsf{m_2}\rangle) \approx \mathsf{m_1} \ ,$$

and thus there is exactly one entry in this map that equals $\mathsf{m_1}$, while all other entries are irreducible terms. Note that the intruder could not verify the correct entry the way we showed, if unless he initially knows the message $\mathsf{m_1}$ or $\mathsf{m_1}$ is guessable.

To summarize, we have so far described a model where the intruder can compose maps by applying operations on every term in a dictionary and on messages that he knows in the classical sense, and compare the outcome of each entry in the map with other messages he knows. The guessing is successful in the sense that he can identify the correct value(s) of the guess(es), only in the case that there is a single entry in the map that he can distinguish from the other ones. Observe that this model is independent from the concrete set of operators and algebraic equations considered (we have only used them for concrete examples) and we can thus entirely avoid specifying a large set of rules to capture all circumstances in which an intruder can guess certain messages. In fact, we have only two "rules": firstly, the intruder can arbitrarily compose

[2]Note that in the case that symmetric encryption and decryption are the same algorithm as in our example, the intruder might even construct the term $\{\!\lvert\{\!\lvert\{\!\lvert m \rvert\!\}_k\rvert\!\}_{k'}\rvert\!\}_{k'}$ by re-encrypting the term with the same key he used for decryption and, modulo the cancelation, obtain the term he started with, i.e. $\{\!\lvert m \rvert\!\}_k$.

messages and maps that he already knows to form new maps, and secondly, he can check if there is a particular entry in a map. In fact, as we will explain in the next section, we do not need to distinguish between messages and maps any more, as we can regard any message as a special case of a map with only one entry.

Before we introduce our formalization of off-line guessing, we conclude this section with some remarks. First, observe that the above discussion does not contain any notion of the probability that a guessing attack is successful. Furthermore, it does not depend on the size of the dictionary, and ignores the case that a password might be guessable (in the sense that it stems from a small set of values), but is not contained in the concrete dictionary of the intruder. The reason why these technical details can be ignored is that one might consider a protocol as vulnerable if there is the mere possibility to mount an attack. In fact, although the vulnerability of a protocol to off-line guessing attacks can be modeled by expressing explicitly the probability of the intruder guessing correctly, in our model we are only concerned with expressing whether this vulnerability exists or not. This makes matters simpler, but it may happen that a protocol is vulnerable to off-line guessing according to our model while it is still safe in practice if all guessing attacks are infeasible. Finally, one may wonder what consequences would arise if we consider an intruder with several dictionaries (instead of just one). It is, however, not difficult to see that this cannot affect our notion of a guessing attack as one could consider the union of all such dictionaries as *the* intruder's dictionary.

12.2 A Formal Model for Off-Line Guessing

We now introduce our formalization of off-line guessing: we give a Dolev-Yao-style deduction system that attempts to capture the intuitions behind off-line guessing in a uniform and general way. As we remarked above, a map is the result of the intruder constructing a message from his knowledge and inserting, in place of an unknown value, every value from the dictionary he uses to perform the attack. The main characteristic of a map is that it is uniform and complete, in the sense that exactly the same sequence of operations is performed for the various values of the guesses, and that all possible guesses in the dictionary are considered. In order to formally define what we mean by "the same sequence of operations" of a map, we introduce the notion of a *pattern term*:

Definition 12.1. *A* dictionary $\mathcal{D}$ *is a finite subset of* Σ_0*, such that none of the entries are intrudable, and also no entry shall appear in the algebraic theory* E*. We will assume a fixed global dictionary in the following. A* pattern term *(or, simply,* pattern*)* P *is an element of* $\mathcal{T}_\Sigma(\mathcal{V})$*. We say that a set* M *is a* map *iff there is a pattern term* P *such that*

$$M = \{(\sigma, P\sigma) \mid dom(\sigma) = vars(P) \wedge \forall v \in vars(P).\ v\sigma \in \mathcal{D}\}\ .$$

We here use variables as place-holders for guesses, not for intruder generated terms as in the context of the lazy intruder technique; for distinction we denote these variables with v, v_1, and v_2. Observe that this formal definition represents exactly the way we described maps informally in the previous section, namely as tables of guesses and the corresponding outcomes. For instance, each of the three maps displayed in Figure 12.2 is represented by the term on top of the right-most column, i.e. $\{\!|\mathsf{m}|\!\}_v$, $\{\!|v_2|\!\}_{v_1}$, and $\pi_1(\{\!|\{\!|\langle \mathsf{m}_1, \mathsf{m}_2\rangle|\!\}_{\mathsf{pw}}|\!\}_v)$, respectively.

It follows straightforwardly from the definition that there is a bijection between maps and patterns, modulo equivalence of patterns and renaming of variables. Hence, a map is uniquely represented by a pattern that is parameterized over a set of data to be guessed, and the intruder's computation of the outcome of this pattern for every possible combination of candidate values for the guesses. Given this bijection, in the following we will identify maps and their corresponding patterns. In particular, for a map M and the corresponding pattern P, we will identify entries of M and substitutions of variables with guesses in P, as well as look-up in M and the term resulting from the application of a substitution of variables with guesses in P.

Maps are expressive constructs: we can view all messages as maps with just one entry, represented by a ground term (so the only substitution possible is the identity). Also, the dictionary

$$\frac{}{P \in \mathcal{GDY}_E(\mathcal{P})}\ G_{\text{axiom}}\ (\text{for } P \in \mathcal{P})\,, \qquad \frac{}{v \in \mathcal{GDY}_E(\mathcal{P})}\ G_{\text{guess}}\ (\text{for } v \in \mathcal{V})\,,$$

$$\frac{P_1 \in \mathcal{GDY}_E(\mathcal{P}) \quad \cdots \quad P_n \in \mathcal{GDY}_E(\mathcal{P})}{f(P_1, \ldots, P_n) \in \mathcal{GDY}_E(\mathcal{P})}\ G_{\text{comp}}\ (f \in \Sigma_n,\ \mathit{intrudable}(f))$$

$$\frac{P_1 \in \mathcal{GDY}_E(\mathcal{P}) \quad P_2 \in \mathcal{GDY}_E(\mathcal{P})}{v\sigma \in \mathcal{GDY}_E(\mathcal{P})}\ G_{\text{ver}}\,,$$

where the rule G_{ver} has the following side conditions:

- $v \in \mathit{vars}(P_1) \cup \mathit{vars}(P_2)$,
- $\Theta = \{\sigma \mid \mathit{dom}(\sigma) = \mathit{vars}(P_1) \cup \mathit{vars}(P_2) \ \wedge\ \forall v \in \mathit{dom}(\sigma).\ v\sigma \in \mathcal{D}\}$,
- $\sigma \in \Theta$,
- $P_1\sigma \approx_E P_2\sigma$,
- $\forall \sigma' \in \Theta.\ P_1\sigma \not\approx_E P_1\sigma' \vee P_2\sigma \not\approx_E P_2\sigma' \implies P_1\sigma' \not\approx_E P_2\sigma'$,
- $\forall P' \in \mathcal{T}_\Sigma(\mathcal{V}).\ P' \approx_E P_1 \implies \mathit{vars}(P') \supseteq \mathit{vars}(P_1)$ and
  $\forall P' \in \mathcal{T}_\Sigma(\mathcal{V}).\ P' \approx_E P_2 \implies \mathit{vars}(P') \supseteq \mathit{vars}(P_2)$.

Figure 12.3: The guessing Dolev-Yao intruder deduction rules.

itself is a map, which is represented by a variable v. This expressiveness allows us to consider a formal model where the intruder knowledge consists only of maps.

We now define an intruder deduction relation over maps, which tells us when the intruder can derive a map from a set of maps. In particular, given that we identify maps with corresponding patterns, we define the deduction relation over patterns for simplicity. Given that "normal" messages are a special case of maps, we can see this system as an extension of the standard Dolev-Yao deduction relation (modulo an algebraic theory). We therefore call this system the *guessing Dolev-Yao intruder* and denote the relation by $\mathcal{GDY}_E$ (where E is the algebraic theory).

Definition 12.2. *For an algebraic theory E and a set of patterns $\mathcal{P}$, the* guessing Dolev-Yao intruder closure $\mathcal{GDY}_E(\mathcal{P})$ *is the smallest set closed under the rules in Figure 12.3 and equivalence in E.*

The first two axiomatic rules are straightforward: from a set of patterns $\mathcal{P}$ we can deduce all patterns that are contained within it. Further we can deduce the pattern v, representing a complete copy of the dictionary. The rule G_{comp} expresses how patterns can be combined. Given n patterns entailed by $\mathcal{P}$, the rule states that they can be combined using an n-ary intrudable function f, yielding a pattern that is the application of f on the constituent patterns.

We now turn our attention to the verification of guesses, rule G_{ver}. In general, verifying a guess requires that the intruder can construct two maps that correspond on exactly one entry (we discuss below how we formalize different substitutions, representing different entries, modulo the algebraic properties). Thus, there is one uniquely determined substitution of dictionary entries for the variables in the two corresponding patterns which yields the same message. This substitution corresponds to the correct guesses, and after verifying his guesses, the intruder learns these correct values for the guesses in the classical sense. Note that this simple definition avoids any notion of "different ways to construct the same message" as is often required in other formal approaches to guessing. Rather, the definition implicitly prevents that the intruder can "cheat himself": suppose he tries to verify something by taking two copies of the same map (which is not forbidden); then either the maps do not contain any variables (so there is nothing guessable to obtain) or the two

maps correspond on all substitutions.

To illustrate this further, let us consider the case where the intruder has derived two patterns P_1 and P_2, and Θ is the set of all substitutions for the variables of P_1 and P_2 with entries of the dictionary. If there is one substitution $\sigma \in \Theta$ on which the two patterns yield the same message, $P_1\sigma \approx_E P_2\sigma$, and the patterns differ for all different substitutions, then the intruder has indeed verified the correct value for any of the data that he has guessed in the two patterns/maps. Thus, for any variable $v \in vars(P_1) \cup vars(P_2)$ in the two patterns, he can derive the correct value $v\sigma$.

Recall that we identify entries of a map and substitutions of variables with guesses in the corresponding pattern. The notion of different substitutions σ and σ' (in the sense that they represent different entries) is not simply $\sigma \neq \sigma'$ for the following reason. A map may contain several different entries that yield the same value. Consider, for instance, the operator $\oplus$, and assume that the intruder knows the message $P_1 = \mathsf{pw}_1 \oplus \mathsf{pw}_2$ for two guessable but unknown passwords pw_1 and pw_2. To find out the passwords, the intruder constructs the map $P_2 = v_1 \oplus v_2$. The problem now is that for the two different entries $\sigma = [v_1 \mapsto \mathsf{pw}_1, v_2 \mapsto \mathsf{pw}_2]$ and $\sigma' = [v_1 \mapsto \mathsf{pw}_2, v_2 \mapsto \mathsf{pw}_1]$ we have $P_2\sigma \approx_E P_2\sigma'$ due to the algebraic properties, and thus there are several entries on which the two maps P_1 and P_2 correspond. Therefore, our notion of different entries is relative to two maps that we want to compare, and we thus consider those pairs of substitutions σ and σ' as being different entries relative to the maps P_1 and P_2, if they indeed make a difference in at least one of the maps, i.e. if $P_1\sigma \not\approx_E P_1\sigma'$ or if $P_2\sigma \not\approx_E P_2\sigma'$.

Finally, note that the last pair of side conditions of the rule G_{ver} ensures that for the verification the intruder uses only maps P_1 and P_2 that are not equivalent to some map P' that comprises fewer variables. For instance, for our example intruder model, the rule G_{ver} cannot be applied to the map $\{|\{|\mathsf{m}|\}_v|\}_v$ as this map is equivalent by $\approx_E$ to the map m. In other words, this condition expresses that we do not consider maps with redundant variables, i.e. those on which the outcome does not depend.

The formal model presented here only allows for derivations that are indeed possible according to the intuition given in the previous section. On the other hand, all operations the intruder could perform according to this intuition have a counter-part in the formal model, since composition and decomposition of messages with guesses is described by respective operations on maps/patterns, and verification of guesses is described by the comparison of maps/patterns. But, of course, one can only informally justify that a formal model captures the intuition underlying it.

Example 12.1. In the first example, the intruder knowledge consists of a set of patterns $\mathcal{P}$ that contains the messages $\{|\mathsf{m}|\}_{\mathsf{pw}}$ and m, and $\mathsf{pw} \in \mathcal{D}_e$ is guessable. The intruder first constructs the map $\{|\mathsf{m}|\}_v$ by encrypting m with every entry of the dictionary, and then compares it with the original message $\{|\mathsf{m}|\}_{\mathsf{pw}}$ to verify his guess of pw. This comparison is possible since there is exactly one entry in the dictionary that equals pw and no other substitution for v that can make $\{|\mathsf{m}|\}_{\mathsf{pw}}$ and $\{|\mathsf{m}|\}_v$ equal:

$$\dfrac{\dfrac{}{\{|\mathsf{m}|\}_{\mathsf{pw}} \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{axiom}} \quad \dfrac{\dfrac{}{v \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{guess}} \quad \dfrac{}{\mathsf{m} \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{axiom}}}{\{|\mathsf{m}|\}_v \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{comp}}}{\mathsf{pw} \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{ver}}$$

In the second example, the intruder knowledge $\mathcal{P}$ contains $\{|\mathsf{m}_2|\}_{\mathsf{m}_1}$, but neither m_1 nor m_2. Further, both m_1 and m_2 are guessable, i.e. $\mathsf{m}_1, \mathsf{m}_2 \in \mathcal{D}_e$. The intruder first encrypts every entry of the dictionary with every entry of the dictionary to obtain the pattern $\{|v_2|\}_{v_1}$, which he can then compare to $\{|\mathsf{m}_2|\}_{\mathsf{m}_1}$. Since again only the substitution with the correct guesses can make the two terms equal, he obtains both m_1 and m_2 by this comparison. Simplifying the presentation, we have joined the derivation of the two messages into one tree:

$$\dfrac{\dfrac{}{\{|\mathsf{m}_2|\}_{\mathsf{m}_1} \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{axiom}} \quad \dfrac{\dfrac{}{v_1 \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{guess}} \quad \dfrac{}{v_2 \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{guess}}}{\{|v_2|\}_{v_1} \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{comp}}}{\mathsf{m}_1, \mathsf{m}_2 \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{ver}}$$

In the third example, the intruder knowledge $\mathcal{P}$ contains the messages m_1 and $\{\!|\langle \mathsf{m}_1, \mathsf{m}_2\rangle|\!\}_{\mathsf{pw}}$, and pw is guessable. The intruder tries to decrypt this message with every entry in his dictionary, obtaining the pattern $\{\!|\{\!|\langle \mathsf{m}_1, \mathsf{m}_2\rangle|\!\}_{\mathsf{pw}}|\!\}_v$, and then builds the projection to the first component for every entry, which yields the pattern $\pi_1(\{\!|\{\!|\langle \mathsf{m}_1, \mathsf{m}_2\rangle|\!\}_{\mathsf{pw}}|\!\}_v)$. He compares this pattern with the known message m_1 and only for the right guess $v = \mathsf{pw}$ the pattern yields this value, so he obtains pw:

$$\dfrac{\dfrac{}{\mathsf{m}_1 \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{axiom}} \qquad \dfrac{\dfrac{}{v \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{guess}} \quad \dfrac{}{\{\!|\langle \mathsf{m}_1, \mathsf{m}_2\rangle|\!\}_{\mathsf{pw}} \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{axiom}}}{\{\!|\{\!|\langle \mathsf{m}_1, \mathsf{m}_2\rangle|\!\}_{\mathsf{pw}}|\!\}_v \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{comp}}}{\pi_1(\{\!|\{\!|\langle \mathsf{m}_1, \mathsf{m}_2\rangle|\!\}_{\mathsf{pw}}|\!\}_v) \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{comp}}}{\mathsf{pw} \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{ver}}$$

□

12.3 A Concrete Example: The MS-CHAPv2 Protocol

As a concrete example, we consider an entire protocol, namely Microsoft's Challenge/Response Authentication Protocol, version 2 (MS-CHAPv2 [134]). MS-CHAPv2 is the authentication mechanism for the Point-to-Point Tunneling Protocol (PPTP [82]), which itself is used to secure PPP connections over TCP/IP. It is well known that this protocol is vulnerable to off-line guessing attacks [123], and we illustrate how one can easily detect this vulnerability using our approach.

Figure 12.1 shows an abstracted version of the MS-CHAPv2 Protocol in the A&B notation that is standard in the literature. Note that, for simplicity, we refrain here from explicitly displaying the pairing operator and simply use commas. The protocol should achieve mutual authentication between a client A and server S based on an initially shared password Pw, which we of course assume to be guessable.

As an illustrative case, let us consider the situation in which the intruder has observed a single run of the protocol between two honest agents a and s. Then the intruder has the following knowledge (assuming that he knows the hash-function h):

$$\mathcal{P} = \{\mathsf{a}, \mathsf{s}, \mathsf{h}, \mathsf{na}, \mathsf{ns}, \mathsf{h}(\mathsf{pw}, \mathsf{na}, \mathsf{ns}, \mathsf{a}), \mathsf{h}(\mathsf{pw}, \mathsf{na})\}\ .$$

Note that we have used lower-case letters to distinguish the concrete data of a protocol run from the protocol variables. The following proof shows that, from this knowledge, the intruder can indeed derive pw, if it is guessable:

$$\dfrac{\dfrac{}{\mathsf{h}(\langle \mathsf{pw}, \mathsf{na}\rangle) \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{axiom}} \qquad \overset{\Pi}{\mathsf{h}(\langle v, \mathsf{na}\rangle) \in \mathcal{GDY}_E(\mathcal{P})}}{\mathsf{pw} \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{ver}}$$

where Π is

$$\dfrac{\dfrac{}{\mathsf{h} \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{axiom}} \qquad \dfrac{\dfrac{}{v \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{guess}} \quad \dfrac{}{\mathsf{na} \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{axiom}}}{\langle v, \mathsf{na}\rangle \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{comp}}}{\mathsf{h}(\langle v, \mathsf{na}\rangle) \in \mathcal{GDY}_E(\mathcal{P})}\, G_{\text{comp}}\ .$$

Here, the intruder simply generates a map for the hash-value of the message under all values of his dictionary in place of pw, and compares the outcome with the observed message. Note that a similar attack would have already been possible after observing only the first three messages of the protocol, since the intruder can similarly build the hash-value of message three under all guesses for the password.

To conclude, observe that all examples we have given are examples of the intruder successfully attacking a protocol by off-line guessing. However, we can also use our formalization as the basis to prove the absence of vulnerabilities of a protocol even under off-line guessing, both in model-checking approaches (where we usually bound the number of sessions considered) or in theorem-proving ones (where we can inductively show the correctness for all situations).

Part IV

Abstraction

In this part, we compare several protocol models that play an important role in the abstraction-based verification of security protocols for an unbounded number of sessions. As security protocols have been intensively studied using formal methods in the past 25 years, a large variety of formalisms, models, and techniques have been developed in order to cope with the different kinds of infinity that arise in verification, in particular with the problem of an unbounded number of sessions. The variety of different models results in a number of problems: it is hard to tell whether the statements proved by different verification tools are equivalent, how tools can be combined, and whether a particular meta-argument about one model (e.g. that certain restrictions are without loss of generality) can be carried over to other models.

We now consider the relationships between variants of two widely used models. The first model is based on set-rewriting, namely a fragment of the Intermediate Format we have used so far. As mentioned, (multi-) set rewriting has been used in various other approaches (e.g. [68, 110]). The second model is inspired by Paulson's approach of message traces based on the Isabelle theorem prover [116]. This latter model has influenced a number of automated verification approaches as discussed below.

Although we consider two particular models, the ideas presented in this chapter are general. There are several works that prove the equivalence of models in different formalisms [45] and the universality of their results [57]. However, it is not our focus here to discuss all design choices and to compare with all existing formalisms. Rather we want to point out that most current protocol models follow the same basic idea as our set rewriting model: the protocol is a set of processes (the honest agents) each of which has a local state that is updated when receiving a message and sending an answer, and these processes are connected via an intruder-controlled network. We thus see the set rewriting approach as a representative of the "standard" way to model protocols. In contrast, the message trace model has no explicit notion of processes or local state; rather we have rules that tell us how a given trace of exchanged messages can be extended with further messages.

The focus of this work is on the *over-approximation* relationship between the models, i.e. when one model allows strictly more traces or reachable states than the other. Over-approximations can induce attacks that are *false positives*, i.e. attacks that work only in the over-approximated model, but not in the original model. The false positives are thus in some sense *artifacts* created by the over-approximation. For falsification (i.e. detecting attacks) this is problematic, as the "real" attacks may be buried under false positives. On the other hand, for verification (i.e. trying to prove a protocol correct) over-approximation does make sense: given a precise model and an over-approximation of it, proving that the over-approximation is safe is often much easier than in the original model and implies that the precise model is safe as well.

The contributions in this part are as follows: We formally prove that the message trace model $\mathbb{T}$ is an over-approximation of the set rewriting model $\mathbb{R}$, i.e. it contains traces that have no counter-part in the set rewriting model (or in reality), while conversely all reachable states of the set rewriting model have their counter part in the message trace model. Moreover, we show that a certain variant $\mathbb{P}$ of the set rewriting model is equivalent to the message trace model in a sense that is made precise in this part, yielding a precise understanding of the relationship between the two models. This relation has never been expressed in the literature before.

We further show how the over-approximations inherent in $\mathbb{R}$ and $\mathbb{T}$ can be used to abstract away a large part of the control structure of protocols, and this control abstraction is the basis of many automated verification approaches [33, 36, 41, 42, 53, 80, 130]. We formally prove that these models ($\mathbb{F}$ and $\mathbb{E}$) are over-approximations of the message trace model, a fact which has also never been formally analyzed before.

These relationships yield a better understanding of these models. While it is immediate that the over-approximations of the different models are sound for reasoning about secrecy, it turns out that they are not sound in general for authentication goals, i.e. a protocol may have an authentication flaw in the set rewriting model, but be correct in the over-approximation. The discovery of this subtlety results from the rigid formal study of the models and their relationship. We also show how to soundly encode authentication goals (based on data abstraction) in the models $\mathbb{F}$ and $\mathbb{E}$.

The work closest related to ours is [35] which considers two models similar to $\mathbb{P}$ and $\mathbb{E}$ and shows

that every attack against secrecy in the original model can be recast in the over-approximation. It does not examine, however, all the relationships between models as we do, and does not consider authentication goals. Also, several papers like [114] deal with the safe over-approximation by data abstraction, which is however orthogonal to this work.

We proceed as follows: we define how the set rewriting model and the message trace model are generated from a strand-space style notation in Chapter 13. In Chapter 14 we formally show that the message trace model over-approximates the set rewriting model. In Chapter 15 we show how to use the over-approximation to perform a complete control abstraction. In Chapter 16 we consider the specification of security goals for the protocols in the different models, based on our results about their relationship.

Chapter 13

Models Based on Set Rewriting and Message Traces

With respect to the previous parts, we consider a few changes. First, the signature Σ of terms is not necessarily finite as in Part III. Moreover, we assume a distinguished subset Σ'_0 of Σ_0 for the creation of fresh data; formally, in an IF rule $l \,=[V]\Rightarrow\, r$, the fresh data for V are chosen from Σ'_0. Also, we do not consider IF rules with negative conditions and negative facts in this part. Also, we have previously interpreted iknows facts on the left-hand side of IF rules by the $\mathcal{DY}$ closure; in contrast, we interpret iknows facts here as normal facts, except for the syntactic sugar that they are implicitly repeated on the right-hand side. Finally, we interpret all terms in the free algebra.

13.1 Message Patterns

In order to be able to make a formal comparison of models, we must define a common basis for describing protocols and translating them into the different formalizations. Also, the results of this part hold only for a restricted class of protocols. Roughly speaking, negative facts and conditions as they are allowed in IF rules can invalidate the arguments behind over-approximations that we are going to use. We therefore define a simple strand-space style formalism, called message patterns, that captures a class of protocols for which the over-approximation arguments hold.

Definition 13.1. *The* message pattern description *of a protocol is given by a finite, non-empty set of role names $\mathcal{R}_1, \ldots, \mathcal{R}_r$ and for every role $\mathcal{R}$ a finite, non-empty list of message terms of $\mathcal{T}_\Sigma(\mathcal{V})$, denoted by $m_0^\mathcal{R}, \ldots, m_n^\mathcal{R}$. We denote the last index n by $last^\mathcal{R}$ and require that it is even. We also require that for every role $\mathcal{R}$, the term $m_0^\mathcal{R}$ is a non-empty concatenation of variables built with the pair operator. This term will later be instantiated with agent names, and the first component is the agent playing the $\mathcal{R}$. We thus denote with $player(m)$ the first component of the message m when m is an instance of $m_0^\mathcal{R}$. We assume that none of the constants of Σ'_0 (which are reserved for fresh data creation) are contained in the protocol description.*

Intuitively, the message terms with an odd index stand for incoming messages, the ones with even index greater than 0 stand for the outgoing messages. Variables that first appear in $m_i^\mathcal{R}$, i.e. not in $m_0^\mathcal{R}, \ldots, m_{i-1}^\mathcal{R}$, are *learned* if i is of odd index, and *freshly created*, if i is of even index greater than 0. The function $fresh_n^\mathcal{R} = vars(m_n^\mathcal{R}) \setminus (vars(m_0^\mathcal{R}) \cup \ldots \cup vars(m_{n-1}^\mathcal{R}))$ abbreviates the set of fresh variables.

Note that we do not require that the combination of the rules yields a meaningful (e.g. executable) protocol. Definition 13.1 gives merely the syntax of a simple formalism. Concerning the semantics, we will in fact define several—the different protocol models that we will formally compare.

Role $\mathcal{A}$		Role $\mathcal{B}$		Role $\mathcal{S}$	
$m_0^{\mathcal{A}}$	$\mathsf{A}, \mathsf{B}, \mathsf{S}$	$m_0^{\mathcal{B}}$	B, S	$m_0^{\mathcal{S}}$	S
$m_1^{\mathcal{A}}$	start				
$m_2^{\mathcal{A}}$	A, B			$m_1^{\mathcal{S}}$	A, B
$m_3^{\mathcal{A}}$	$\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(\mathsf{S})), \mathsf{B}, \mathsf{PKB})$			$m_2^{\mathcal{S}}$	$\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(\mathsf{S})), \mathsf{B}, \mathsf{pk}(\mathsf{B}))$
$m_4^{\mathcal{A}}$	$\{\mathsf{NA}, \mathsf{A}\}_{\mathsf{PKB}}$	$m_1^{\mathcal{B}}$	$\{\mathsf{NA}, \mathsf{A}\}_{\mathsf{pk(B)}}$		
		$m_2^{\mathcal{B}}$	B, A	$m_3^{\mathcal{S}}$	B, A
		$m_3^{\mathcal{B}}$	$\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(\mathsf{S})), \mathsf{A}, \mathsf{PKA})$	$m_4^{\mathcal{S}}$	$\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(\mathsf{S})), \mathsf{A}, \mathsf{pk}(\mathsf{A}))$
$m_5^{\mathcal{A}}$	$\{\mathsf{NA}, \mathsf{NB}, \mathsf{B}\}_{\mathsf{pk(A)}}$	$m_4^{\mathcal{B}}$	$\{\mathsf{NA}, \mathsf{NB}, \mathsf{B}\}_{\mathsf{PKA}}$		
$m_6^{\mathcal{A}}$	$\{\mathsf{NB}\}_{\mathsf{PKB}}$	$m_5^{\mathcal{B}}$	$\{\mathsf{NB}\}_{\mathsf{pk(B)}}$		
		$m_6^{\mathcal{B}}$	end		

Figure 13.1: Example of message patterns: the Needham-Schroeder Public Key Protocol with Lowe's fix (NSL).

Example 13.1. The "canonical example" of the Needham-Schroeder Public Key Protocol with Lowe's fix (NSL) can be described by the message pattern sequences displayed in Figure 13.1. The layout is arranged so that corresponding messages of different roles are on the same line. Since role $\mathcal{A}$ is starting the protocol, the first incoming message is the dummy message start; similarly, as $\mathcal{B}$ has no outgoing message in the last step, we have here the dummy message end. These two dummy messages make the rest of the formalization uniform as we do not have to consider special cases for the first and last message. During the protocol execution, $\mathcal{A}$ freshly creates the nonce NA (since the first appearance of NA in any of $\mathcal{A}$'s messages is outgoing) and learns two values PKB and NB (since these variables first appear in incoming messages for role $\mathcal{A}$).

Observe that the message terms $m_2^{\mathcal{S}}$ and $m_3^{\mathcal{A}}$ are identical except for the subterms PKB and $\mathsf{pk}(\mathsf{B})$, respectively. This reflects the fact that $\mathcal{S}$ knows everybody's public key, but $\mathcal{A}$ does not. Thus $\mathcal{A}$ would accept any message at the position of PKA and use it in subsequent communication. $\mathcal{A}$ only knows her own public/private key-pair and the public key of $\mathcal{S}$. In fact, the private key of $\mathcal{A}$ is never explicitly used, but it is implicitly used, since $\mathcal{A}$ expects to receive a message encrypted with its public key in $m_5^{\mathcal{A}}$. □

In some cases, it is necessary to limit the set of agents that can play a certain role, for instance the server in the NSL protocol cannot be the intruder (otherwise the protocol could not guarantee anything). Thus, the protocol description also includes a predicate $inst_{\mathcal{R}}(m)$ which says whether m is an allowed instance of $m_0^{\mathcal{R}}$. We assume that the set of m with $inst_{\mathcal{R}}(m)$ is finite for every role (otherwise Theorem 15.1 would not hold), which means that the set of agents who can participate in protocol runs is finite; however, we do not limit the number of protocol runs. The restriction to a finite number of agents can be justified as in [57] by identifying the names of honest agents in different protocol runs. Finally, we assume a finite set of ground messages IK_0 associated with the protocol, representing the initial knowledge of the intruder. For the example of the NSL protocol, let $IK_0 = \{\mathsf{i}, \mathsf{pk}(\mathsf{i}), \mathsf{inv}(\mathsf{pk}(\mathsf{i})), \mathsf{s}, \mathsf{pk}(\mathsf{s}), \mathsf{start}\}$.

Since we never consider more than one protocol at a time, we consider the protocol description in the form of Definition 13.1 as fixed for the rest of this work, in order to avoid indexing everything with the protocol description. The formalization of goals for a protocol is addressed in Chapter 16, when the relationship of the models is clear. In the comparison of models, we focus on what the intruder can deduce and a notion of the state of honest agents.

$InitialState$:
$\mathsf{iknows}(m)$ for every $m \in IK_0$

$Rules$:
$\Rightarrow \mathsf{state}_{\mathcal{R}}(m)$ for every role $\mathcal{R}$ and message m such that $inst_{\mathcal{R}}(m)$

$\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{n-2}^{\mathcal{R}}).\mathsf{iknows}(m_{n-1}^{\mathcal{R}}) =[fresh_n^{\mathcal{R}}]\Rightarrow \mathsf{iknows}(m_n^{\mathcal{R}}).\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_n^{\mathcal{R}})$
for every role $\mathcal{R}$ and even index n such that $0 < n \leq last^{\mathcal{R}}$

$\mathsf{iknows}(m_1).\ldots.\mathsf{iknows}(m_n) \Rightarrow \mathsf{iknows}(m)$
for every intruder rule $\dfrac{m_1 \in \mathcal{DY}(IK) \quad \ldots \quad m_n \in \mathcal{DY}(IK)}{m \in \mathcal{DY}(IK)}$

Figure 13.2: The set rewriting model. Note that iknows facts on a left-hand side is implicitly present on the right-hand side.

13.2 The Set Rewriting Model

Figure 13.2 defines the set rewriting model by an initial state and a set of set rewrite rule schemata (i.e. rules that are parameterized over the protocol description); an example is given in Figure 13.4. The initial state for the set rewriting model consisting of an iknows fact for every message that the intruder initially knows. The first rule schema (which has no left-hand side and can therefore be applied to any state) describes the creation of a new protocol run: for every role and for every permitted instantiation of the agent names of that role, we can create a new local state of an agent. The binary fact symbol $\mathsf{state}_{\mathcal{R}}(m)$ represents the local state of an agent in the execution of the protocol in role $\mathcal{R}$; the information that the agent has gathered so far is a concatenation of the instantiations (i.e. his own name and maybe other predetermined agents) and the messages he has sent and received up to now. The next rule schema describes the transitions of honest agents: when an agent is at the stage to have sent the message $n-2$ of his role description, and receives $n-1$ from the network (i.e. the intruder), then he can answer with message n to the network (i.e. the intruder) and update his local state by adding messages $n-1$ and n. (Recall that the state fact of the left-hand side is removed, while the iknows fact remains due to the syntactic sugar we have defined.) Also, in such a transition all fresh variables ($fresh_n^{\mathcal{R}}$) are substituted for constants that have not occurred so far. The last rule schema directly expresses the deduction rules of the intruder (as said before, we do not interpret the iknows facts of set writing rules under the Dolev-Yao closure in this part). We now inductively define the set of reachable states $\mathbb{R}$ as the least set of states that includes the initial state and that is closed under the $\rightarrow_R$ relation induced by the rules.

While we store the entire message terms sent and received in agent state facts for simplicity, one could rather just store the set of variables that occur in these messages (as we have done in previous examples of IF), in order to have smaller state facts for efficiency. Note that in this case the subterm in several steps of the protocol execution may not change since no new incoming and outgoing messages are handled. Therefore one must in these cases add a step number to the list of variables to distinguish different states of the protocol execution.

The reader may wonder whether there can be a problem due to the fact that in set rewriting we cannot have multiple identical state facts. However, we can at any time use the initialization rule to obtain the initial state facts, and from there repeat any execution we had before up to the point where the agent creates fresh data. From that point on, the state facts are already different due to the fresh data. This argumentation allows us to work without *session identifiers*. (However, when there is not such an instantiation rule, but we want to define a finite number of sessions by respective state facts in the initial state, then such session identifiers are indeed necessary.)

$$\frac{}{\epsilon \in \mathbb{T}} \qquad \frac{t \in \mathbb{T} \quad A \in \Sigma_0 \setminus \Sigma_0' \quad M \in \mathcal{DY}(IK_0 \cup \{M \mid \mathsf{snd}\_(M) \in [t]\})}{\mathsf{rcv}_A(M) : t \in \mathbb{T}}$$

$$\frac{\begin{array}{l} t \in \mathbb{T} \quad inst_{\mathcal{R}}(M_0) \\ fresh_n^{\mathcal{R}} \subseteq \Sigma_0' \setminus used(t) \end{array} \qquad \mathsf{rcv}_A(m_1^{\mathcal{R}}) \in [t] \quad \mathsf{snd}_A(m_2^{\mathcal{R}}) \in [t] \quad \ldots \quad \mathsf{rcv}_A(m_{n-1}^{\mathcal{R}}) \in [t]}{\mathsf{snd}_A(m_n^{\mathcal{R}}) : t \in \mathbb{T}}$$

for every $\mathcal{R}$, for every n that is even and $0 < n \leq last^{\mathcal{R}}$, and $A = player(m_0^{\mathcal{R}})$.

Figure 13.3: The Message Trace Model

$$\mathbb{R}: \quad \mathsf{state}_{\mathcal{B}}((\mathsf{B},\mathsf{S}), \{\mathsf{NA},\mathsf{A}\}_{\mathsf{pk(B)}}, (\mathsf{B},\mathsf{A})).\mathsf{iknows}(\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(\mathsf{S})),\mathsf{A},\mathsf{PKA})) \\ =[\mathsf{NB}]\Rightarrow \mathsf{iknows}(\{\mathsf{NA},\mathsf{NB},\mathsf{B}\}_{\mathsf{PKA}}).\mathsf{state}_{\mathcal{B}}(\ldots, \{\mathsf{A},\mathsf{PKA}\}_{\mathsf{inv(pk(S))}}, \{\mathsf{NA},\mathsf{NB},\mathsf{B}\}_{\mathsf{PKA}})$$

$$\mathbb{T}: \quad \frac{\begin{array}{ll} t \in \mathbb{T} & \mathsf{rcv_B}(\{\mathsf{NA},\mathsf{A}\}_{\mathsf{pk(B)}}) \in [t] \quad \mathsf{snd_B}(\mathsf{B},\mathsf{A}) \in [t] \\ \mathsf{NB} \in \Sigma_0' \setminus used(t) & \mathsf{rcv_B}(\{\mathsf{A},\mathsf{PKA}\}_{\mathsf{inv(pk(S))}}) \in [t] \end{array}}{\mathsf{snd_B}(\{\mathsf{NA},\mathsf{NB},\mathsf{B}\}_{\mathsf{PKA}}) : t \in \mathbb{T}}$$

Figure 13.4: Example of the different models: the first transition rule of role $\mathcal{B}$ in NSL.

13.3 The Message Trace Model

The message trace model characterizes the protocol by a set of traces $\mathbb{T}$ that is defined as the least set closed under the rules in Figure 13.3; an example is given in Figure 13.4. A *trace* is a finite sequence of events. In our model, there are two kinds of events, namely $\mathsf{snd}_a(m)$ and $\mathsf{rcv}_a(m)$, where a is a constant and m is a ground term, representing that a sends or receives the message m. The first rule simply says that the empty trace ϵ is in $\mathbb{T}$. The second rule formalizes that the intruder can see all sent messages on the network, and that all messages, which any agent A receives, are chosen by the intruder from the set of messages he can deduce from what he has seen on the network. We allow as an agent name any constant of $\Sigma_0 \setminus \Sigma_0'$ since, by default, there is no notion of typing. Recall that Σ_0' is the distinguished subset of Σ_0 reserved for creating fresh data. The symbol ":" represents concatenating an event to a trace, and $[t]$ gives the set of events of a trace. To describe the behavior of honest agents, we have a rule for each role $\mathcal{R}$ and suitable index n: if a trace containing all send and receive events up to point n in the protocol execution of role $\mathcal{R}$, then the trace can be extended by the next send event of $\mathcal{R}$. Here, $used(t)$ is the set of constants that occur in any message of trace t. This formalization is close to the one of [116], but there are some differences.

One difference is that we do not necessarily consider a typed model here. As we have seen in Subsection 4.4.4, we can enforce well-typed messages by using appropriate function symbols in the messages. The results we present here are therefore independent of the question whether messages are typed or not. The second difference is that [116] uses a combination of send and receive events, denoted $a \to b : m$, which at first sight looks like a synchronous communication model (where, in particular, the intruder cannot intercept messages), but this is not the case. In recent works, similar events as in our model are used, called *Says* and *Gets* [29].

There are further minor differences: in [116], there are two functions *synth* and *analz* that represent each a part of the $\mathcal{DY}$ closure, namely the synthesis and analysis rules, and instead of $\mathcal{DY}(IK)$ the closure $synth(analz(IK))$ is employed, which in general is a proper subset of $\mathcal{DY}(IK)$. We thus use the complete $\mathcal{DY}(IK)$. Also, the way we use the set of permitted instances of agent names is not present in the rules of [116]; this is due to our generalization to arbitrary protocols, while for a concrete protocol, such checks are not necessary in every rule.

Chapter 14

The Over-Approximation and Persistent Set Rewriting

The intuition about why the models $\mathbb{R}$ and $\mathbb{T}$ are so similar is that every state fact of the set rewriting model in some way corresponds to the respective send and receive events in the message trace models. We formalize this intuition by a function $[\![\cdot]\!]$ that maps a state fact to a set of events:

$$[\![\mathsf{state}_{\mathcal{A}}(m_0, \ldots, m_n)]\!] = \{\mathsf{rcv}_A(m_1), \mathsf{snd}_A(m_2), \ldots, \mathsf{rcv}_A(m_{n-1}), \mathsf{snd}_A(m_n)\} \ ,$$

where $A = \mathit{player}(m_0)$. Intuitively the other components of m_0 will later be reflected by the additional condition $\mathit{inst}_{\mathcal{R}}(m_0)$ which is not an event. This mapping, together with conditions on the intruder-deducible messages, is the basis for comparing two models, namely to prove inclusions between models and equivalences between models.

We now discuss an aspect that is inherent in the message trace model. We observe that in all rules we have only positive conditions on the existence of events on a given trace, with the exception of the creation of fresh data. For instance, the NSL example of Figure 13.1 allows the trace shown in Figure 14.1 which demonstrates the over-approximation. For brevity, we have omitted the steps with the server $\mathcal{S}$, and also used the A&B style notation $a \to b : m$ to abbreviate the two events $\mathsf{snd}_a(m)$ and $\mathsf{rcv}_b(m)$; moreover, we have labeled these events with step numbers of the protocol (as meant by the sender) and distinguished "different" sessions using primed labels. In this trace, we have one normal execution of the protocol (steps 1–3). But then in step $2'$, b chooses to react a second time to the first message of a. This is because there is nothing in the rule for $\mathcal{B}$ that says that he may not react to a message, to which he has already reacted. Even worse, at this point he has already received an answer to his first reaction, so in fact the session should already be over at this point. However, b chooses the fresh nonce n_3 which he associates with a's first nonce n_1, so from his point of view there are two different sessions, although a has only sent one message. In step $3'$, agent a reacts to b's second reply to the initial message from a—observe that both step 2 and step $2'$ contain the same nonce n_1. This means that a does not "remember" that the nonce n_1 is already used and that it should rather not accept messages with this nonce anymore. Finally, a sends in step $3''$ another answer to step 2.

More generally, this phenomenon stems from the fact that there is no notion of local states of the honest agents in the message trace model—all "memory" agents have in this model lies in the messages exchanged. Basically that would be enough to reconstruct each agent's local state, however, that would require negative predicates on the messages exchanged. For instance, in the case of NSL (without key-server for simplicity), the reaction of role $\mathcal{B}$ might look like this in a trace model without over-approximation:

$$\frac{\begin{array}{ll} t \in \mathbb{T} \quad \mathit{inst}_{\mathcal{R}}(\mathsf{A}, \mathsf{B}) & \\ \mathsf{NB} \in \Sigma_0' \setminus \mathit{used}(t) & \mathsf{rcv}_{\mathsf{B}}(\{\mathsf{NA}, \mathsf{A}\}_{\mathsf{pk(B)}}) \in [t] \quad \mathsf{snd}_{\mathsf{B}}(\{\mathsf{NA}, \mathsf{NB}', \mathsf{B}\}_{\mathsf{pk(A)}}) \notin [t] \end{array}}{\mathsf{snd}_{\mathsf{B}}(\{\mathsf{NA}, \mathsf{NB}, \mathsf{B}\}_{\mathsf{pk(A)}}) : t \in \mathbb{T}}$$

1.	$\mathsf{a} \to \mathsf{b}:$	$\{\mathsf{n}_1, \mathsf{a}\}_{\mathsf{pk(b)}}$	$2'.$	$\mathsf{b} \to \mathsf{a}:$	$\{\mathsf{n}_1, \mathsf{n}_3, \mathsf{b}\}_{\mathsf{pk(a)}}$
2.	$\mathsf{b} \to \mathsf{a}:$	$\{\mathsf{n}_1, \mathsf{n}_2, \mathsf{b}\}_{\mathsf{pk(a)}}$	$3'.$	$\mathsf{a} \to \mathsf{b}:$	$\{\mathsf{n}_3\}_{\mathsf{pk(b)}}$
3.	$\mathsf{a} \to \mathsf{b}:$	$\{\mathsf{n}_2\}_{\mathsf{pk(b)}}$	$3''.$	$\mathsf{a} \to \mathsf{b}:$	$\{\mathsf{n}_2\}_{\mathsf{pk(b)}}$

Figure 14.1: A trace of NSL demonstrating the over-approximation.

Note that one must use a different variable NB' here to express that B has not sent any answer to A's message with that NA.

In the case of the NSL, this phenomenon does not seem to be a problem—as far as secrecy and certain forms of authentication are concerned. However, the strong variant of authentication as defined in Section 4.5, which additionally checks for replay of messages, is trivially violated for every protocol in this model, since every agent is willing to accept incoming messages of the correct format any number of times. This issue is discussed in Chapter 16. Similarly, many protocols cannot be reasonably modeled with this message trace based model. As an example, the contract signing protocol ASW [12] that we describe in detail Section 17.2 involves a trusted third party (TTP), which is contacted in case of a dispute. In a nutshell, the TTP can give out abort tokens or replacement contracts, and it is essential that the TTP never gives out a replacement contract for a (partial) session that it has already aborted. Thus, the steps of the TTP can only be performed if certain events have *not* yet occurred, and the protocol has trivial false positives otherwise. Therefore, the formalization of this protocol is beyond the scope of the message pattern specification we have introduced in Definition 13.1.

Thus, due to the over-approximation, for a given protocol and goal, we might not be able to prove in the message trace model that the protocol fulfills the goal, even if that is the case in a more precise model. However, we are on the safe side, if we can show that an over-approximated model is indeed safe—as long as this model is indeed an over-approximation of the desired system. Thus, we need to formally prove that the message trace model is indeed an over-approximation of the set rewriting model, i.e. that every state reachable in the set rewriting model of a protocol corresponds to a trace in the message trace model, in a sense to be made precise below.

14.1 Persistent Set Rewriting

We now show that there is an over-approximation of the set rewriting model, the *persistent set rewriting* model, which in turn is over-approximated by the message trace model. The advantage of taking this little "detour" via persistent rewriting is a precise understanding of the relationships of the models.

Intuitively, the phenomenon just described can be understood as follows: in every transition, an agent forks into two incarnations: one incarnation learns the new incoming message, and creates and remembers the outgoing message of this step. The other incarnation just remains in the state it is, to later possibly spawn further incarnations from exactly this local state. Proving formally the inclusion relationships, we can formalize this intuition via the persistent set rewriting model.

For the persistent set rewriting model, we change the honest agent rules of the set rewriting model as follows: every state fact of the left-hand side is also repeated on the right-hand side, formalizing that one incarnation of the agent remains in its present state. The honest agents rules thus look as follows:

$$\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{n-2}^{\mathcal{R}}).\mathsf{iknows}(m_{n-1}^{\mathcal{R}}) \\ -[\mathit{fresh}_n^{\mathcal{R}}]\Rightarrow \mathsf{iknows}(m_n^{\mathcal{R}}).\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{n-2}^{\mathcal{R}}).\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_n^{\mathcal{R}})$$

Note that this is the same construction that we have already implicitly defined for the iknows facts. Now all facts contained in a state are also contained in all successor states, hence the name "persistent". We denote the reachable states of the persistent set rewriting model as $\mathbb{P}$.

14.2 Relation to Set Rewriting

We first show that the persistent set rewriting model is indeed an over-approximation of the set rewriting model, in the sense that every reachable state of $\mathbb{R}$ is subsumed by one in $\mathbb{P}$:

Theorem 14.1. *For every reachable state $s \in \mathbb{R}$, there is a reachable state $s' \in \mathbb{P}$ such that $s \subseteq s'$ and $\mathcal{DY}(s) = \mathcal{DY}(s')$.*

Proof. This is shown by structural induction, where we have to slightly strengthen the induction hypothesis in that corresponding states s and s' have the same set of used fresh identifiers (i.e. constants from Σ_0').

As the induction basis, observe that the initial states are identical in both models. For the induction step, consider a reachable state s of the set rewriting model, a transition rule $LHS{=}[V]{\Rightarrow}RHS$ of the set rewriting model and a substitution σ such that $LHS\sigma \subseteq s$. (Note that σ also assigns values to the fresh data in V.) The induction hypothesis is that there is a state $s' \supseteq s$ reachable in the persistent set rewriting model such that $\mathcal{DY}(s) = \mathcal{DY}(s')$ and s and s' have the same set of fresh constants used.

We show that for the successor state $t = (s \setminus LHS\sigma) \cup RHS\sigma$ in the set rewriting model, there is a corresponding state $t' \supseteq t$ reachable in one transition from s' the persistent set rewriting model. We use the corresponding rule in the persistent set rewriting model, which is $LHS = [V] \Rightarrow LHS.RHS$ (we exploit in this notation that the iknows facts are implicitly persistent). This rule is applicable to s' with the same substitution σ, since $s' \supseteq s \supseteq LHS\sigma$ and s and s' use the same set of fresh constants from Σ_0'. The resulting new state is

$$t' = (s' \setminus LHS\sigma) \cup LHS\sigma \cup RHS\sigma = s' \cup RHS\sigma \supseteq s \cup RHS\sigma \supseteq t \ ,$$

It also follows that s' and t' still use the same set of fresh constants. To see that t and t' have the same intruder knowledge, observe that $\mathcal{DY}(IK_1 \cup IK_2) = \mathcal{DY}(\mathcal{DY}(IK_1) \cup IK_2)$, as $\mathcal{DY}$ is a closure operation. Now we have that

$$\begin{aligned}&\mathcal{DY}(t') = \mathcal{DY}(s' \cup RHS\sigma) = \mathcal{DY}(\mathcal{DY}(s') \cup RHS\sigma) = \mathcal{DY}(\mathcal{DY}(s) \cup RHS\sigma) = \mathcal{DY}(s \cup RHS\sigma)\\ &= \mathcal{DY}((s \setminus LHS\sigma) \cup RHS\sigma) = \mathcal{DY}(t) \ ,\end{aligned}$$

where we have used that iknows facts are implicitly monotonic and thus $\{m \mid \mathsf{iknows}(m) \in s\} = \{m \mid \mathsf{iknows}(m) \in (s \setminus LHS\sigma) \cup RHS\sigma\}$. □

This implies, in particular, that for every reachable state s in the set rewriting model, there is a state s' in the persistent set rewriting model, such that the set of messages the intruder can derive in s' is at least as large as in s, and that every state fact of s is contained in s'.

14.3 Inclusion in the Inductive Message Trace Model

The next step is to show that the persistent set rewriting model is in a sense subsumed in the message trace model. We do this with respect to the intruder knowledge and the local states of honest agents as formalized by the $[\![\cdot]\!]$ function. To this end, we extend the definition of $\mathcal{DY}(\cdot)$ to a set of facts s, namely $\mathcal{DY}(s) = \mathcal{DY}(\{m \mid \mathsf{iknows}(m) \in s\})$, and to a set of events E, namely $\mathcal{DY}(E) = \mathcal{DY}(IK_0 \cup \{\{m \mid \mathsf{snd\_}(m) \in [t]\})$, given the initial intruder knowledge IK_0 attached to a protocol.

Theorem 14.2. *For every reachable state $s \in \mathbb{P}$, there is a trace $t \in \mathbb{T}$ such that $\mathcal{DY}(s) = \mathcal{DY}([t])$ and for every state fact $f \in s$, $[\![f]\!] \subseteq [t]$.*

Proof. We use again structural induction, in this case on the inductive structure of $\mathbb{P}$, and again we strengthen the induction hypothesis so that the states s and the trace t in question use the same set fresh constants from Σ_0'.

Induction Basis. For the initial state of the persistent set rewriting model and the empty trace $\epsilon \in \mathbb{T}$ fulfills the assumption, as there are no state facts in the initial state (and thus no send and receive events need to be on the trace), there are no fresh data and what the intruder can derive is in both cases $\mathcal{DY}(IK_0)$.

Induction Step. Given any reachable state $s \in \mathbb{P}$ and a trace $t \in \mathbb{T}$ that corresponds to s in the sense of the induction hypothesis. We show that for every state s' reachable from s by one application of a rewrite rule, we can find an extension t' of t in $\mathbb{T}$ that fulfills the induction hypothesis.

Consider any transition rule applicable to s and the successor state s', as well as a trace t that corresponds to s according to the induction hypothesis; we distinguish three cases, one for each kind of rule:

1. The rule is an initialization rule, i.e. of the form $\Rightarrow \mathsf{state}_{\mathcal{R}}(m)$ such that $inst_{\mathcal{R}}(m)$ holds. This rule neither creates fresh data nor increases the intruder knowledge. m is a ground instance of $m_0^{\mathcal{R}}$, thus $[\![\mathsf{state}_{\mathcal{R}}(m)]\!] = \emptyset \subseteq [t]$. Therefore, t already satisfies the induction hypothesis.

2. The rule is an intruder rule. This adds $\mathsf{iknows}(m)$ where $m \in \mathcal{DY}(s)$, thus $\mathcal{DY}(s') = \mathcal{DY}(s) = \mathcal{DY}([t])$. Also, no fresh data are created and no state facts are added. So again, t already satisfies the induction hypothesis for s'.

3. The rule is an honest agent state transition rule, i.e. of the form

$$\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{n-2}^{\mathcal{R}}).\mathsf{iknows}(m_{n-1}^{\mathcal{R}}) \\ =[\mathit{fresh}_n^{\mathcal{R}}]\Rightarrow \mathsf{iknows}(m_n^{\mathcal{R}}).\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{n-2}^{\mathcal{R}}).\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_n^{\mathcal{R}}) \ ,$$

   for some $\mathcal{R}$ and $0 < n \leq \mathit{last}^{\mathcal{R}}$. Let σ be a substitution under which the rule is applied. It follows that $m_{n-1}^{\mathcal{R}}\sigma \in \mathcal{DY}(s) = \mathcal{DY}(t)$. Thus, we can apply the intruder rule of the message trace model to obtain the trace $t' = \mathsf{rcv}_a(m_n^{\mathcal{R}}\sigma) : t \in \mathbb{T}$ where $a = \mathit{player}(m_0^{\mathcal{R}}\sigma)$. Note that t and t' have the same intruder knowledge and use the same set of fresh constants. Thus, $\mathit{fresh}_n^{\mathcal{R}}\sigma$ are constants that are not yet used in t'. From the construction, it follows that $\mathit{inst}_{\mathcal{R}}(m_0^{\mathcal{R}}\sigma)$. By the induction hypothesis, $[\![\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{n-2}^{\mathcal{R}})\sigma]\!] = \{\mathsf{rcv}_a(m_1^{\mathcal{R}}\sigma), \mathsf{snd}_a(m_1^{\mathcal{R}}\sigma), \ldots, \mathsf{snd}_a(m_{n-2}^{\mathcal{R}}\sigma)\} \subseteq [t] \subseteq [t']$. This gives us all premises in order to apply the corresponding inductive rule of the message trace model to t', yielding the trace $t'' = \mathsf{snd}_a(m_n^{\mathcal{R}}\sigma) : t' \in \mathbb{T}$. t'' uses the same amount of fresh data as s', moreover for the new state fact of s' we have

$$[\![\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_n^{\mathcal{R}})\sigma]\!] = \\ [\![\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{n-2}^{\mathcal{R}})\sigma]\!] \cup \{\mathsf{rcv}_a(m_{n-1}^{\mathcal{R}})\sigma, \mathsf{snd}_a(m_n^{\mathcal{R}})\sigma\} \subseteq [t''] \ .$$

   For the intruder knowledge we have

$$\mathcal{DY}(s') = \mathcal{DY}(s \cup \{\mathsf{iknows}(m_n^{\mathcal{R}})\sigma\}) = \mathcal{DY}([t] \cup \{\mathsf{snd}_a(m_n^{\mathcal{R}})\sigma\}) = \mathcal{DY}(t'') \ .$$

   Therefore, $t'' \in \mathbb{T}$ corresponds to s' in the sense of the induction hypothesis.

This completes the induction step. □

This shows that $\mathbb{T}$ is an over-approximation of $\mathbb{P}$. We now give an example showing that the converse direction of this inclusion relationship does not hold in general, i.e. there are traces in the $\mathbb{T}$ model which have no counter part in the $\mathbb{P}$ model.

Example 14.1. Consider the protocol displayed in Figure 14.2, which is similar to NSL, but an important difference is that steps 3 and 4 are independent of what happened in step 2, namely from the nonce NB created by B. The message trace model now admits the trace also shown in Figure 14.2. At the first sight this might look like the phenomena we have seen before, but here

$$
\begin{array}{lll}
1. & \mathsf{A} \to \mathsf{B}: & \{\mathsf{NA}, \mathsf{A}\}_{\mathsf{pk(B)}} \\
2. & \mathsf{B} \to \mathsf{A}: & \{\mathsf{NA}, \mathsf{NB}, \mathsf{B}\}_{\mathsf{pk(A)}} \\
3. & \mathsf{A} \to \mathsf{B}: & \{\mathsf{NA}, \mathsf{NA}'\}_{\mathsf{pk(B)}}
\end{array}
$$

1.	$\mathsf{a} \to \mathsf{b}:$	$\{\mathsf{n}_1, \mathsf{a}\}_{\mathsf{pk(b)}}$	$\mathsf{state}_{\mathcal{A}}(\mathsf{n}_1)$	$\mathsf{state}_{\mathcal{B}}(\mathsf{n}_1)$
2.	$\mathsf{b} \to \mathsf{a}:$	$\{\mathsf{n}_1, \mathsf{n}_2, \mathsf{b}\}_{\mathsf{pk(a)}}$	$\mathsf{state}_{\mathcal{A}}(\mathsf{n}_1, \mathsf{n}_2)$	$\mathsf{state}_{\mathcal{B}}(\mathsf{n}_1, \mathsf{n}_2)$
3.	$\mathsf{a} \to \mathsf{b}:$	$\{\mathsf{n}_1, \mathsf{n}_3\}_{\mathsf{pk(b)}}$	$\mathsf{state}_{\mathcal{A}}(\mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_3)$	$\mathsf{state}_{\mathcal{B}}(\mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_3)$
$2'$.	$\mathsf{b} \to \mathsf{a}:$	$\{\mathsf{n}_1, \mathsf{n}_4, \mathsf{b}\}_{\mathsf{pk(a)}}$	$\mathsf{state}_{\mathcal{A}}(\mathsf{n}_1, \mathsf{n}_4)$	$\mathsf{state}_{\mathcal{B}}(\mathsf{n}_1, \mathsf{n}_4)$
			$\mathsf{state}_{\mathcal{A}}(\mathsf{n}_1, \mathsf{n}_4, \mathsf{n}_3)$	$\mathsf{state}_{\mathcal{B}}(\mathsf{n}_1, \mathsf{n}_4, \mathsf{n}_3)$

Figure 14.2: A protocol and a trace in the message trace model. On the right of the trace the (simplified) agent-states that would correspond to the trace.

is something different. Interpreting every line $a \to b : m$ as the events $\mathsf{snd}_a(m)$ and $\mathsf{rcv}_b(m)$, then we obtain the state facts (also shown in Figure 14.2) that correspond to the trace with respect to the $[\![\cdot]\!]$ function. We have simplified the presentation of the state facts so that we only display the involved constants in place of the protocol variables NA, NB, and NA', respectively. Let us have a closer look at the fact $\mathsf{state}_{\mathcal{A}}(\mathsf{n}_1, \mathsf{n}_4, \mathsf{n}_3)$ which is created by step $2'$ of the trace. This follows by interpreting steps 1, $2'$ and 3 as "one session". Observe that there is already $\mathsf{state}_{\mathcal{A}}(\mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_3)$ by interpreting steps 1, 2, and 3 as one session. Thus in the $\mathbb{T}$ model, after a complete run of the protocol, a new message can be interpreted as belonging to the completed run and be processed accordingly.

In the set rewriting and the persistent set rewriting model, there cannot be a counter part for such a trace. The reason is that simulating the run would create fresh nonces for NA', namely once we have $\mathsf{state}_{\mathcal{A}}(\mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_3)$, we cannot obtain the state $\mathsf{state}_{\mathcal{A}}(\mathsf{n}_1, \mathsf{n}_4, \mathsf{n}_3)$ anymore, but the last nonce n_3 is necessarily some other fresh constant in the (persistent) set rewriting model. Thus, the $\mathbb{P}$ and the $\mathbb{T}$ model are in general not equivalent. We will see below, however, that for protocols with data abstraction, we can indeed prove the equivalence of the models. □

Chapter 15

Reachable Facts and Events

In the previous section, we have shown that a large part of the control structure of protocols is abstracted away in the $\mathbb{P}$ and $\mathbb{T}$ models, namely the order in which events have occurred does not play any role and the set of reached local states of honest agents grows monotonically. An obvious idea might thus be to simplify matters further by considering the set of *reachable facts* $\mathbb{F} = \bigcup_{s \in \mathbb{P}} s$ and the set of *reachable events* $\mathbb{E} = \bigcup_{t \in \mathbb{T}} [t]$. This drastically simplifies many search problems, as instead of searching an infinite state search space, we have to consider only the facts that occur in any state.[1]

However, there are two problems related to this. Firstly, authentication goals cannot be formalized in the usual way anymore as we present in Chapter 16. Secondly, the fresh data in every state or trace is chosen based on what has not yet been used. Collecting facts and events from different traces would give a kind of collision, e.g. the fresh data in a session between an honest agent and the intruder (which the intruder may know) and the fresh data in a session between honest agents (which the intruder may not know) can be identical. This renders the $\mathbb{F}$ and $\mathbb{E}$ models useless for protocols with fresh data.

Another view on this is the following: both $\mathbb{P}$ and $\mathbb{T}$ are inductively defined. Usually, all rules of an inductive definition are monotonic in the sense that a rule is applicable to a set s, then it is also applicable to any superset of s. This guarantees that the fixed-point, i.e. the least set closed under the rules, is uniquely defined. If one instead uses an inductive definition of the set of reachable facts and events, then there is the problem of the negative condition that fresh data are constants not used so far. Leaving out this condition renders the model useless (as every protocol will then have trivial attacks). Including the condition, however, leads to non-monotonic induction rules and the fixed-point is in this case not uniquely defined, but rather depends on the order in which the inductive rules have been applied.

However, in many automated verification approaches, a kind of data abstraction is performed [33, 36, 41, 42, 53, 80, 130]—inspired by the idea of abstract interpretation [63]. The idea is the following: if we map the infinite set of (fresh) data to finitely many equivalence classes and instead of each concrete datum rather consider its abstract equivalence class, many search spaces become finite and the abstract model is an over-approximation of the concrete one. This argumentation has, however, a prerequisite: there may not be any negative comparisons in the rules, e.g. that two nonces are not the same, or else the abstract model would not be an over-approximation of the concrete one.

We do not discuss the matters of data abstraction here, but we consider protocol models where no fresh data occur, and show that under these circumstances we can safely perform the simplification to reachable facts and events models, and moreover, that the reachable facts and the reachable events models are then equivalent in the sense we have defined before. As an example how an abstract model of NSL can look like, consider the data abstraction displayed in Figure 15.1: the nonces created by role $\mathcal{A}$ are replaced by $\mathsf{na}(\mathsf{A}, \mathsf{B})$ in the rules for $\mathcal{A}$ and the fresh nonces created

[1] The set of reachable facts is finite under certain conditions, namely when data abstraction is performed, as discussed below, and when the intruder is restricted to well-typed messages.

Role $\mathcal{A}$		Role $\mathcal{B}$	
$m_0^{\mathcal{A}}$	$\mathsf{A}, \mathsf{B}, \mathsf{S}$	$m_0^{\mathcal{B}}$	B, S
$m_1^{\mathcal{A}}$	start		
$m_2^{\mathcal{A}}$	$\{\mathsf{na}(\mathsf{A}, \mathsf{B}), \mathsf{A}\}_{\mathsf{pk}(\mathsf{B})}$	$m_1^{\mathcal{B}}$	$\{\mathsf{NA}, \mathsf{A}\}_{\mathsf{pk}(\mathsf{B})}$
$m_3^{\mathcal{A}}$	$\{\mathsf{na}(\mathsf{A}, \mathsf{B}), \mathsf{NB}, \mathsf{B}\}_{\mathsf{pk}(\mathsf{A})}$	$m_2^{\mathcal{B}}$	$\{\mathsf{NA}, \mathsf{nb}(\mathsf{A}, \mathsf{B}), \mathsf{B}\}_{\mathsf{PKA}}$
$m_4^{\mathcal{A}}$	$\{\mathsf{NB}\}_{\mathsf{pk}(\mathsf{B})}$	$m_3^{\mathcal{B}}$	$\{\mathsf{nb}(\mathsf{A}, \mathsf{B})\}_{\mathsf{pk}(\mathsf{B})}$
		$m_4^{\mathcal{B}}$	end

Figure 15.1: NSL (without key-server) under data abstraction.

by role $\mathcal{B}$ are replaced by $\mathsf{nb}(\mathsf{B}, \mathsf{A})$ in the rules for $\mathcal{B}$, where na and nb are new binary function symbols. Observe that the abstraction is only performed on the side of the agent who creates the nonces, e.g. the nonce NA is only abstracted on $\mathcal{A}$'s side, while $\mathcal{B}$ still accepts any value in this place. As an intuition, one may think of agents who always use the same nonce when talking to the same agents. Intuitively it should be clear that, if the protocol is safe under such a behavior of the agents, then it is also safe in case the agents indeed freshly create the nonces in each session.

Definition 15.1. *We say that a protocol is* data-abstract *(or* has no fresh data*) iff* $fresh_n^{\mathcal{R}} = \emptyset$ *for all roles* $\mathcal{R}$ *and all even indices* $0 < n \leq last^{\mathcal{R}}$.

For data-abstract protocols, we can now show the converse direction of the inclusion relation of Theorem 14.2, namely that every trace of the message trace model has a counter part (in terms of intruder knowledge and local agent states) in the persistent set rewriting model:

Theorem 15.1. *If the protocol is data-abstract, then for all* $t \in \mathbb{T}$ *there exists a state* $s \in \mathbb{P}$ *such that* $\mathcal{DY}([t]) = \mathcal{DY}(s)$ *and all agent states represented by* t *are contained in* s*:*

$$\{\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n) \mid [\![\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n)]\!] \subseteq [t] \wedge inst_{\mathcal{R}}(m_0)\} \subseteq s \ .$$

Proof. This proof uses induction over the inductive definition of $\mathbb{T}$.

Induction Basis. Consider the empty trace $\epsilon \in \mathbb{T}$. Its intruder knowledge is $\mathcal{DY}(IK_0)$, it has no fresh data. Moreover, since there are no send and receive events in the empty trace, the following holds:

$$\begin{aligned}&\{\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n) \mid [\![\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n)]\!] \subseteq [t] \wedge inst_{\mathcal{R}}(m_0)\} \\ &\quad = \{\mathsf{state}_{\mathcal{R}}(m_0) \mid inst_{\mathcal{R}}(m_0)\}\end{aligned}$$

Here it becomes important that the set of all m, for which $inst_{\mathcal{R}}(m)$ holds, is finite, otherwise we could not reach—in finitely many steps—a state of the $\mathbb{P}$ model that corresponds to the empty trace. (And a similar problem would arise in the induction step below.) For finitely many such m, however, a straightforward induction proof shows that we can reach a state s from the initial state s_0 of $\mathbb{P}$, which contains all required state facts, using the initialization rules of the $\mathbb{P}$ model. It holds that $\mathcal{DY}(s) = \mathcal{DY}(s_0) = \mathcal{DY}(IK_0)$, and there are no fresh data used in s. Therefore s corresponds to the empty trace in the sense of the induction hypothesis.

Induction Step. Given any trace $t \in \mathbb{T}$, for which we already have a corresponding state $s \in \mathbb{P}$ in the sense of the induction hypothesis. We now show that for every extension t' of t that we can obtain by using a rule of $\mathbb{T}$, we can find a state $s' \in \mathbb{P}$ that can be reached from s in finitely many steps of the $\mathbb{P}$ model and that also satisfies the induction hypothesis with respect to t'.
Let $states(t) = \{\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n) \mid [\![\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n)]\!] \subseteq [t] \wedge inst_{\mathcal{R}}(m_0)\}$ denote the set of state facts associated with a trace t. First observe that, even though t' is an extension of t with just one event, $states(t') \setminus states(t)$ can contain more than one state fact. In other words, a single event in the $\mathbb{T}$ model may correspond to multiple steps of the $\mathbb{P}$ model, as demonstrated by the example above.

We first show that $states(t)$ is finite for every trace $t \in \mathbb{T}$ (otherwise we would be unable to reach a corresponding state of $\mathbb{P}$ in finitely many steps). This can be shown by a simple inductive proof: the empty trace has a finite corresponding set of states as seen above, and with every extension t' of a trace t by one event, the set remains also finite, as in the worst case this new event can induce a finite number of new state facts for every state fact of $states(t)$.

Next, we remark a monotonicity property of $[\![\cdot]\!]$ in the way a state fact "grows", namely $[\![\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n)]\!] \supseteq [\![\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_{n-2})]\!]$ for all even $0 < n \leq last^{\mathcal{R}}$. This induces a well-founded partial order on $states(t')$, namely the order $[\![\cdot]\!] \supseteq [\![\cdot]\!]$ which cannot have an infinite descending chain (since $[\![t']\!]$ is finite) and, moreover, every descending chain eventually has an end in $[\![t]\!]$. Let $f_1, \ldots, f_n$ be a linearization of the facts of $states(t') \setminus states(t)$ in this partial order, i.e. each f_i is an extension of a state fact in $states(t) \cup \{f_1, \ldots, f_{i-1}\}$.

We now show by an "inner" induction that we can construct a sequence $s = s_0 \Rightarrow^* s_1 \Rightarrow^* \ldots \Rightarrow^* s_n = s'$ of reachable states of $\mathbb{P}$, such that for some sets of messages $IK_i \subseteq \mathcal{DY}([t'])$ with $IK_{i+1} \supseteq IK_i$ it holds that $s_i = s \cup \{f_1, \ldots, f_i\} \cup \{\mathsf{iknows}(m) \mid m \in IK_i\}$.

Inner Induction Basis: for s_0 (and $IK_0 = \emptyset$) nothing is to show.

Inner Induction Step: Suppose we have already shown reachability of s_i with the above conditions and a respective IK_i. We show we can then reach s_{i+1} for some $IK_{i+1} \supseteq IK_i$. The state fact f_{i+1}, for which we have to show a way to reach it, must have the form $\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_n^{\mathcal{R}})\sigma$ for some substitution σ (due to the definition of $[\![\cdot]\!]$) and some even n with $0 < n \leq last^{\mathcal{R}}$. Also, by the inner induction hypothesis, $\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{n-2}^{\mathcal{R}})\sigma$ is already contained in s_i. We thus have to show that the intruder can construct the message $m_{n-1}^{\mathcal{R}}\sigma$, so we can apply the respective transition rule of $\mathbb{P}$. The respective receive event, namely $\mathsf{rcv}_a(m_{n-1}^{\mathcal{R}}\sigma)$ for $a = player(m_0^{\mathcal{R}}\sigma)$ must be present in t already (otherwise the extension to t' is not possible). Therefore $m_{n-1}^{\mathcal{R}}\sigma \in \mathcal{DY}([t]) = \mathcal{DY}(s)$, by the outer induction hypothesis. Thus, we can obtain the fact $\mathsf{iknows}(m_{n-1}^{\mathcal{R}}\sigma)$ by finitely many applications of intruder deduction rules on s_i, i.e. there is a state s_i' with $s_i \rightarrow^* s_i' \in \mathbb{P}$ and $s_i' = s_i \cup \{\mathsf{iknows}(m) \mid m \in IK_i'\}$ for some set $IK_i \subseteq IK_i' \subseteq \mathcal{DY}([t])$. We can now finally apply the transition rule to s_i' which gives us $s_{i+1} = s_i' \cup \{\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_n^{\mathcal{R}}\sigma), \mathsf{iknows}(m_n^{\mathcal{R}}\sigma)\}$. Define $IK_{i+1} = IK_i' \cup \{m_n^{\mathcal{R}}\sigma\}$. Then s_{i+1} and IK_{i+1} satisfy the inner induction hypothesis: $IK_{i+1} \subseteq \mathcal{DY}([t'])$, s_{i+1} is reachable in finitely many steps from s_i and

$$s_{i+1} = s_i \cup \{f_{i+1}\} \cup \{\mathsf{iknows}(m) \mid m \in IK_{i+1}\} .$$

From the inner induction proof, we can now show that the outer induction hypothesis holds for t' and s': $\mathcal{DY}(s') = \mathcal{DY}(s \cup IK_n) = \mathcal{DY}([t'])$ and $states(t') \subseteq s'$. □

By the combination of Theorem 14.2 and Theorem 15.1, the persistent set rewriting model and the inductive trace model thus coincide for protocols under data abstraction.

We now formally investigate the relationship between the persistent set rewriting model to the reachable facts model. The over-approximation relationship directly follows from the definition, in the sense that every fact of a reachable state is present in $\mathbb{F}$. What we prove now is that the reachable facts model does not "merge too many facts", in the sense that every subset of the reachable facts is contained in some reachable state of the persistent set rewriting model. The following Theorem states this and an analogous statement for the reachable events and message traces models:

Theorem 15.2. *If the protocol has no fresh data, then for every* finite *set of reachable facts $F \subseteq \mathbb{F}$, there is a reachable state $S \in \mathbb{P}$ such that $F \subseteq S$. Similarly, for every* finite *set of reachable events $E \subseteq \mathbb{E}$, there is a trace $t \in \mathbb{T}$ such that $E \subseteq [t]$.*

Proof. Reachable states and facts. Given a finite set $F = \{f_1, \ldots, f_n\} \subseteq \mathbb{F}$, then by definition of $\mathbb{F}$, for each f_i there is a state $S_i \in \mathbb{P}$, such that $f_i \in S_i$. Thus it is sufficient to show that for any reachable states $S_1, S_2 \in \mathbb{P}$, their union $S_1 \cup S_2 \in \mathbb{P}$ is reachable. Then, the proposition follows by inductively applying this argument to all S_i for each f_i. Let $S_1, S_2 \in \mathbb{P}$, and let $S_0 \in \mathbb{P}$ be the initial state of the persistent set rewriting model. Then $S_0 \subseteq S_1$ and $S_0 \subseteq S_2$, as the rewriting is persistent. Since no fresh data are created, the same rule under the same substitution can be applied to S_1 as to S_0, so to obtain the facts of S_2. Thus $S_1 \cup S_2 \in \mathbb{P}$.

$$\frac{m_1^{\mathcal{R}} \in \mathbb{M} \quad \ldots m_{n-1}^{\mathcal{R}} \in \mathbb{M}}{m_n^{\mathcal{R}} \in \mathbb{M}} \qquad \begin{array}{l}\text{for every role } \mathcal{R}\text{, for every even } 0 < n \leq last^{\mathcal{R}} \\ inst_{\mathcal{R}}(m_0^{\mathcal{R}})\end{array}$$

Figure 15.2: The most abstract model as pure intruder deduction.

Reachable traces and events. The argument is completely analogous with $[\cdot]$ applied to the traces. □

Note that from this proof we can see that the reachable facts and events fixed-point are only meaningful because $\mathcal{DY}(IK)$ is itself monotone in IK (i.e. the more messages the intruder has seen, the more he can deduce).

We conclude this section with a consequence of all previous Theorems, namely that for protocols under data abstraction, also the reachable facts and reachable events models coincide:

Theorem 15.3. *Consider a data-abstract protocol. Then* $\mathcal{DY}(\mathbb{F}) = \mathcal{DY}(\mathbb{E})$*, and moreover it holds that* $\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n) \in \mathbb{F}$ *iff* $inst_{\mathcal{R}}(m_0)$ *and* $[\![\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n)]\!] \subseteq \mathbb{E}$*.*

Proof. $\mathcal{DY}(\mathbb{F}) \overset{\text{Def. } \mathbb{F}}{=} \mathcal{DY}(\cup_{s \in \mathbb{P}} s) \overset{\text{Th.14.2 \& Th.15.1}}{=} \mathcal{DY}(\cup_{t \in \mathbb{T}}[t]) \overset{\text{Def. } \mathbb{E}}{=} \mathcal{DY}(\mathbb{E})$.

Let $\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n) \in \mathbb{F}$. Then $inst_{\mathcal{R}}(m_0)$, and there is $s \in \mathbb{P}$ with $\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n) \in s$. By Theorem 14.2, there is a trace $t \in \mathbb{T}$, such that $[\![\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n)]\!] \subseteq [t]$. Thus

$$[\![\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n)]\!] \subseteq \mathbb{E}\ .$$

Let $[\![\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n)]\!] \subseteq \mathbb{E}$ and $inst_{\mathcal{R}}(m_0)$. Then, by Theorem 15.2, there is a trace $t \in \mathbb{T}$ such that $[\![\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n)]\!] \subseteq [t]$. By Theorem 15.1, there is a reachable state $s \in \mathbb{P}$, such that $\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n) \in s$. Thus $\mathsf{state}_{\mathcal{R}}(m_0, \ldots, m_n) \in \mathbb{F}$. □

This concludes the argumentation on the relationship between the persistent rewriting model and the message trace model: if we have no fresh data in messages, we can move to a reachable facts and reachable events model, respectively, without any loss of generality, and then they are equivalent.

Several abstract verification approaches, e.g. [36, 53, 80], go yet a step further in the approximation and consider only message terms, i.e. ignoring the information whether messages were sent or received. The idea is to not even distinguish between the messages the intruder sends and the messages an honest agent sends, thus we focus on a pure intruder deduction problem. We define the model $\mathbb{M}$ as the least set of messages that contains IK_0 and that is closed under the rules shown in Figure 15.2 and the rules of $\mathcal{DY}$. With this model, we have finally arrived at a stage where the "entire protocol is abstracted away" resulting in a pure intruder deduction problem: in fact, we see the behavior of honest agents just as additional intruder deduction rules and the honest agents are now nothing more than *oracles* in the terminology of [53] for the intruder: the intruder can ask them *questions* of a certain form (the messages in the premises of a rule), and they will give a *reply* as a function of the question (the consequence of the rule). The model $\mathbb{M}$ is again an over-approximation of the previous models:

Theorem 15.4. $\mathbb{M} \supseteq \mathcal{DY}(\mathbb{E})$.

Proof. Consider a total order on the events in $\mathbb{E} = \{e_1, e_2, \ldots\}$ such that e_i is the consequence of a rule of the $\mathbb{E}$ model for some subset of $\{e_1, \ldots, e_{i-1}\}$ as premises. (This is possible due to the inductive definition of $\mathbb{E}$.) We show $\mathcal{DY}(\{e_1, \ldots, e_i\}) \subseteq \mathbb{M}$ by induction over i.

For the base case $i = 0$ we have $\mathcal{DY}([\epsilon]) \subseteq \mathbb{M}$. For the induction step, assume $\mathcal{DY}(\{e_1, \ldots, e_i\}) \subseteq \mathbb{M}$. It is to show that this also holds for $i + 1$. We distinguish the cases that e_{i+1} is a send or receive event.

- If $e_{i+1} = \mathsf{rcv}_a(m)$, then $m \in \mathcal{DY}(\{e_1, \ldots, e_i\}) \subseteq \mathbb{M}$, by the induction hypothesis. Since $\mathbb{M}$ is closed under the rules of $\mathcal{DY}$, also $m \in \mathbb{M}$.
- If $e_{i+1} = \mathsf{snd}_a(m)$, then $\mathsf{rcv}_a(m_1^{\mathcal{R}})\sigma, \mathsf{snd}_a(m_2^{\mathcal{R}})\sigma, \ldots, \mathsf{rcv}_a(m_{n-1}^{\mathcal{R}})\sigma \in \mathbb{F}$ for some $\mathcal{R}$, even index $0 < n \leq \mathit{last}^{\mathcal{R}}$ and substitution σ such that $a = \mathit{player}(m_0^{\mathcal{R}}\sigma)$ and $m = m_n^{\mathcal{R}}\sigma$ and $\mathit{inst}_{\mathcal{R}}(m_0\sigma)$. By the induction hypothesis we therefore have $m_1^{\mathcal{R}}\sigma, \ldots, m_{n-1}^{\mathcal{R}}\sigma \in \mathbb{M}$, thus by the rules of $\mathbb{M}$, also $m_n^{\mathcal{R}}\sigma \in \mathbb{M}$.

□

Chapter 16

Goals

We have so far not considered the question of how to describe, in the different models, the properties that a protocol is supposed to ensure, or conversely, what states or traces would count as attacks to the protocol. Based on the results from the previous chapters, namely the correspondence between local states of honest agents on the one side, and receive and send events on the other side, we now investigate how one can formalize standard goals in the different models.

While in Section 4.5, we have already defined secrecy and authentication for the security properties for the Intermediate Format, we do not want to integrate all the different fact-symbols (or respective events) into each of the models here, in particular as this would have made the theorems of this part more complicated. Also, many existing verification tools for an unbounded number of sessions do not have such special events and facts for security goals. We thus give definitions based on local agent states, exchanged messages, or intruder knowledge.

16.1 Secrecy

We first consider secrecy goals which are straightforward to handle: we can apply Theorem 14.1, Theorem 14.2, Theorem 15.2, and Theorem 15.3 to show that, for instance, if the intruder can find out a secret in a reachable state of $\mathbb{R}$, then there is a trace of $\mathbb{T}$ in which the intruder can also find out that secret.

For the message pattern specification, we can specify a secrecy goal by means of a role $\mathcal{R}$, a subterm of a message pattern of $\mathcal{R}$, and a set of protocol variables (the latter representing the principals who may know it). For instance, in the example of NSL, we could specify that for role $\mathcal{A}$ both the variables NA and NB are secrets shared between the principals instantiating the protocol variables A and B of role $\mathcal{A}$. (And one can specify similar goals from $\mathcal{B}$'s point of view.)

In the set rewriting model, a violation of secrecy is thus defined as a state in which the intruder knows a message that is supposed to be secret between a set of agents that he does not belong to. More concretely, for the NSL example, an attack state for secrecy of NA and NB from $\mathcal{A}$'s point of view is thus formulated as follows:

$$\begin{aligned}&\exists s \in \mathbb{R}.\, \exists \sigma.\\ &\quad \mathsf{state}_{\mathcal{A}}(\mathsf{A}, \mathsf{B}, \ldots, \{\mathsf{NA}, \mathsf{A}\}_{\mathsf{PKB}}, \ldots, \{\mathsf{NA}, \mathsf{NB}, \mathsf{A}\}_{\mathsf{pk(A)}}, \{\mathsf{NB}\}_{\mathsf{PKB}})\sigma \in s \wedge\\ &\quad \mathsf{A}\sigma \neq \mathsf{i} \wedge \mathsf{B}\sigma \neq \mathsf{i} \wedge (\mathsf{iknows}(\mathsf{NA}\sigma) \in s \vee \mathsf{iknows}(\mathsf{NB}\sigma) \in s)\end{aligned}$$

More generally, for each of the different models, we now define a predicate that defines, what a violation of a secrecy goal means. The particular secrecy goal is specified by a triple $(\mathcal{R}, M, \{A_1, \ldots, A_n\})$ where $\mathcal{R}$ is a role, M is a subterm of one of $\mathcal{R}$'s message patterns (i.e. $m_0^{\mathcal{R}}, \ldots, m_{last^{\mathcal{R}}}^{\mathcal{R}}$), and each A_i is a variable that appears in these message patterns. Intuitively, it means that every message that instantiates M in a concrete run of the protocol of $\mathcal{R}$ must be kept secret between the agents instantiating the variables $\{A_1, \ldots, A_n\}$; in particular, the intruder

may not find out the instantiation of M unless he is one of the A_i. The formal definition for the different models is as follows:

$$\begin{aligned}
&\mathit{secrecyFlaw}_{\mathbb{R}}(\mathcal{R}, M, \{A_1, \ldots, A_n\}) = \exists s \in \mathbb{R}.\exists \sigma.\\
&\quad \mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{last^{\mathcal{R}}}^{\mathcal{R}}\sigma) \in s \wedge \mathsf{i} \notin \{A_1\sigma, \ldots, A_n\sigma\} \wedge \mathsf{iknows}(M\sigma) \in s\\
&\mathit{secrecyFlaw}_{\mathbb{P}}(\mathcal{R}, M, \{A_1, \ldots, A_n\}) = \exists s \in \mathbb{P}.\exists \sigma.\\
&\quad \mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{last^{\mathcal{R}}}^{\mathcal{R}}\sigma) \in s \wedge \mathsf{i} \notin \{A_1\sigma, \ldots, A_n\sigma\} \wedge \mathsf{iknows}(M\sigma) \in s\\
&\mathit{secrecyFlaw}_{\mathbb{T}}(\mathcal{R}, M, \{A_1, \ldots, A_n\}) = \exists t \in \mathbb{T}.\exists \sigma.\\
&\quad [\![\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{last^{\mathcal{R}}}^{\mathcal{R}}\sigma)]\!] \subseteq [t] \wedge \mathsf{i} \notin \{A_1\sigma, \ldots, A_n\sigma\} \wedge M\sigma \in \mathcal{DY}([t])\\
&\mathit{secrecyFlaw}_{\mathbb{F}}(\mathcal{R}, M, \{A_1, \ldots, A_n\}) = \exists \sigma.\\
&\quad \mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{last^{\mathcal{R}}}^{\mathcal{R}}\sigma) \in \mathbb{F} \wedge \mathsf{i} \notin \{A_1\sigma, \ldots, A_n\sigma\} \wedge \mathsf{iknows}(M\sigma) \in \mathbb{F}\\
&\mathit{secrecyFlaw}_{\mathbb{E}}(\mathcal{R}, M, \{A_1, \ldots, A_n\}) = \exists \sigma.\\
&\quad [\![\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{last^{\mathcal{R}}}^{\mathcal{R}}\sigma)]\!] \subseteq \mathbb{E} \wedge \mathsf{i} \notin \{A_1\sigma, \ldots, A_n\sigma\} \wedge M\sigma \in \mathcal{DY}(\mathbb{E})\\
&\mathit{secrecyFlaw}_{\mathbb{M}}(\mathcal{R}, M, \{A_1, \ldots, A_n\}) = \exists \sigma.\\
&\quad \{m_0^{\mathcal{R}}\sigma, \ldots, m_{last^{\mathcal{R}}}^{\mathcal{R}}\sigma\} \subseteq \mathbb{M} \wedge \mathsf{i} \notin \{A_1\sigma, \ldots, A_n\sigma\} \wedge M\sigma \in \mathbb{M}
\end{aligned}$$

Note that in this definition we use the relationships between the different models that we have established so far, in particular, we use the $[\![\cdot]\!]$ function to "translate" the state facts into the models that talks about events instead.

The following Theorem tells us that the over-approximations are safe in the sense that if the more precise models have an attack to secrecy, then also the approximated models have an attack to secrecy:

Theorem 16.1. *For any secrecy goal G, the following implications hold:*

- $\mathit{secrecyFlaw}_{\mathbb{R}}(G)$ *implies* $\mathit{secrecyFlaw}_{\mathbb{P}}(G)$,
- $\mathit{secrecyFlaw}_{\mathbb{P}}(G)$ *implies* $\mathit{secrecyFlaw}_{\mathbb{T}}(G)$ *and* $\mathit{secrecyFlaw}_{\mathbb{F}}(G)$,
- $\mathit{secrecyFlaw}_{\mathbb{T}}(G)$ *implies* $\mathit{secrecyFlaw}_{\mathbb{E}}(G)$,
- $\mathit{secrecyFlaw}_{\mathbb{E}}(G)$ *implies* $\mathit{secrecyFlaw}_{\mathbb{M}}(G)$.

Proof. The proof is immediate from the previous Theorem 14.1-Theorem 15.4, and we just give the first implication as an example. Suppose $\mathit{secrecyFlaw}_{\mathbb{R}}(G)$ holds; then there is a state $s \in \mathbb{R}$ and a substitution σ such that the state predicate that corresponds to the final state of $\mathcal{R}$ under σ, none of the variables A_i is substituted by intruder, and $M\sigma$ is known to the intruder. By Theorem 14.1, we thus have that this state fact and this iknows fact are present in some reachable state $s' \in \mathbb{P}$ of the persistent set rewriting model, and this s' therefore satisfies the conditions of $\mathit{secrecyFlaw}_{\mathbb{P}}(G)$, and thus $\mathbb{P}$ also has this attack. □

Using Data Abstraction for Secrecy Specifications. As explained previously, the models $\mathbb{E}$, $\mathbb{F}$, and $\mathbb{M}$ only make sense in the context of data abstraction. This data abstraction can be used for a simpler formulation of the secrecy goals. Let us consider the example of the NSL protocol under the data abstraction, i.e. that for NA, $\mathcal{A}$ chooses $\mathsf{na}(\mathsf{A}, \mathsf{B})$ for a nonce intended for B in role $\mathcal{B}$. Then we can specify for the $\mathbb{F}$ model (similarly for the $\mathbb{E}$ and $\mathbb{M}$ model) a violation of secrecy of NA between A and B as follows:

$$\exists \sigma.\mathsf{na}(\mathsf{A}, \mathsf{B})\sigma \in \mathcal{DY}(\mathbb{F}) \wedge \mathsf{i} \notin \{\mathsf{A}\sigma, \mathsf{B}\sigma\}$$

This is equivalent to the previous definition of $\mathit{secrecyFlaw}_{\mathbb{F}}(\cdot)$ for this example, since every term that can be substituted during any protocol execution for the protocol variable NA of an agent in

role $\mathcal{A}$ has the form $\mathsf{na}(\mathsf{A},\mathsf{B})$ due to the data abstraction. Thus every violation of secrecy of NA from $\mathcal{A}$'s point of view is described by the underivability of the secret $\mathsf{na}(\mathsf{A},\mathsf{B})$ in case i is neither A nor B.

We note that in a similar way, we can specify the secrecy of NB from $\mathcal{B}$'s point of view. However, using the data abstraction one cannot similarly specify the secrecy of NA from $\mathcal{B}$'s point of view or the secrecy of NB from $\mathcal{A}$'s point of view. That is because the messages that agents receive in these positions are not necessarily of the form $\mathsf{na}(\mathsf{A},\mathsf{B})$ and $\mathsf{nb}(\mathsf{B},\mathsf{A})$, respectively.

16.2 Authentication

We will first discuss a common way to express authentication goals and show that this causes a serious problem with the over-approximation. As an example for authentication, first consider again the NSL protocol.

For the persistent set rewriting model $\mathbb{P}$, we can formulate that $\mathcal{A}$ fails to authenticate $\mathcal{B}$ on nonce NB as follows (where $\sigma(dom(\sigma) = V)$ expresses the restriction that σ is a substitution with domain V):

$$\begin{aligned}&\exists s \in \mathbb{P}.\ \exists \sigma(dom(\sigma) = vars(\mathsf{A})).\ \forall \tau(dom(\tau) = vars(\mathsf{B}) \setminus vars(\mathsf{A})).\ \mathsf{A}\sigma \neq \mathsf{i} \wedge \mathsf{B}\sigma \neq \mathsf{i} \wedge \\ &\ \mathsf{state}_{\mathcal{A}}(\mathsf{A},\mathsf{B},\mathsf{S},\ldots,\{\mathsf{NA},\mathsf{A}\}_{\mathsf{PKB}},\{\mathsf{NA},\mathsf{NB},\mathsf{B}\}_{\mathsf{pk(A)}},\{\mathsf{NB}\}_{\mathsf{PKB}})\sigma \in s \wedge \\ &\ \mathsf{state}_{\mathcal{B}}(\mathsf{B},\mathsf{S},\{\mathsf{NA},\mathsf{A}\}_{\mathsf{pk(B)}},\ldots,\{\mathsf{NA},\mathsf{NB},\mathsf{B}\}_{\mathsf{PKA}})\sigma\tau \notin s\end{aligned}$$

This formalizes that there is a reachable state in the $\mathbb{P}$ model, such that for certain values of the protocol variables, an agent A in role $\mathcal{A}$ has finished the protocol while in that state no agent B in role $\mathcal{B}$ exists who agrees with A on their names and the exchanged nonces. Note that we do not demand here that B must have finished his run of the protocol, he must have only progressed far enough to have created the nonce NB. (For the standard set rewriting model, one has to additionally require that no extension of $\mathcal{B}$'s state fact is in s, or else the goal would lead to false positives.) Also, the formula requires $\mathsf{A} \neq \mathsf{i}$ and $\mathsf{B} \neq \mathsf{i}$, as sessions have no security guarantees, if the intruder is a participant.

Let us now consider how such a goal would be encoded in the message trace model $\mathbb{T}$. Using the $[\![\cdot]\!]$ function, we may attempt to formulate the above authentication goal for $\mathbb{T}$ as follows:

$$\begin{aligned}&\exists t \in \mathbb{T}.\ \exists \mathsf{A},\mathsf{B},\mathsf{S},\ldots,\mathsf{NA},\mathsf{NB},\mathsf{PKA},\mathsf{PKB} \in \mathcal{T}_{\Sigma}.\ \mathsf{A} \neq \mathsf{i} \wedge \mathsf{B} \neq \mathsf{i} \wedge \\ &\ [\![\mathsf{state}_{\mathcal{A}}(\mathsf{A},\mathsf{B},\mathsf{S},\ldots,\{\mathsf{NA},\mathsf{A}\}_{\mathsf{PKB}},\{\mathsf{NA},\mathsf{NB},\mathsf{B}\}_{\mathsf{PKA}},\{\mathsf{NB}\}_{\mathsf{PKB}})]\!] \subseteq [t] \wedge \\ &\ [\![\mathsf{state}_{\mathcal{B}}(\mathsf{B},\mathsf{S},\{\mathsf{NA},\mathsf{A}\}_{\mathsf{PKB}},\ldots,\{\mathsf{NA},\mathsf{NB},\mathsf{B}\}_{\mathsf{PKA}})]\!] \not\subseteq [t]\end{aligned}$$

This is in fact the standard way to express authentication for the message trace model: by the $[\![\cdot]\!]$ function, all state facts get translated into messages sent or received by some agent. The important question is now whether or not this formulation of authentication goals for $\mathbb{T}$ is sound. In other words, does verifying $\mathbb{T}$ for an authentication goal imply verifying $\mathbb{R}$ for that goal?

Before we look at this question, let us give a general definition of authentication flaws for the $\mathbb{R}$, $\mathbb{P}$, and $\mathbb{T}$ model, independent from the concrete protocol example.

An authentication goal is a five-tuple $(\mathcal{A}, A, \mathcal{B}, B, M)$, pronounced as "$\mathcal{A}$ authenticates $\mathcal{B}$ on M". The components are as follows: $\mathcal{A}$ and $\mathcal{B}$ are roles, and A and B are meta-variables for the names of the players of $\mathcal{A}$ and $\mathcal{B}$. M is a term on which the agents shall agree upon at the end of the protocol run. In the following, let $n = last^{\mathcal{A}}$ and let $l(M) = \min\{i \in \{1,\ldots,last^{\mathcal{B}}\} \mid M \text{ subterm of } m_i^{\mathcal{B}}\}$ denote the final step in the protocol execution of $\mathcal{A}$ (i.e. where $\mathcal{A}$ accepts M)

as well as the first step in the protocol execution of $\mathcal{B}$ where $\mathcal{B}$ knows M:

$$
\begin{aligned}
&\mathit{authFlaw}_{\mathbb{R}}(\mathcal{A}, A, \mathcal{B}, B, M) = \\
&\quad \exists s \in \mathbb{R}.\exists \sigma(\mathit{dom}(\sigma) = \mathit{vars}(\mathcal{A})).\forall \tau(\mathit{dom}(\tau) = \mathit{vars}(\mathcal{B}) \setminus \mathit{vars}(\mathcal{A})). \\
&\qquad A\sigma \neq \mathsf{i} \wedge B\sigma \neq \mathsf{i} \wedge \\
&\qquad \mathsf{state}_{\mathcal{A}}(m_0^{\mathcal{A}}, \ldots, m_n^{\mathcal{A}})\sigma \in s \wedge \\
&\qquad \mathsf{state}_{\mathcal{B}}(m_0^{\mathcal{B}}, \ldots, m_{l(M)}^{\mathcal{B}})\sigma\tau \notin s \wedge \ldots \wedge \mathsf{state}_{\mathcal{B}}(m_0^{\mathcal{B}}, \ldots, m_n^{\mathcal{B}})\sigma\tau \notin s
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{authFlaw}_{\mathbb{P}}(\mathcal{A}, A, \mathcal{B}, B, M) = \\
&\quad \exists s \in \mathbb{P}.\exists \sigma(\mathit{dom}(\sigma) = \mathit{vars}(\mathcal{A})).\forall \tau(\mathit{dom}(\tau) = \mathit{vars}(\mathcal{B}) \setminus \mathit{vars}(\mathcal{A})). \\
&\qquad A\sigma \neq \mathsf{i} \wedge B\sigma \neq \mathsf{i} \wedge \\
&\qquad \mathsf{state}_{\mathcal{A}}(m_0^{\mathcal{A}}, \ldots, m_n^{\mathcal{A}})\sigma \in s \wedge \mathsf{state}_{\mathcal{B}}(m_0^{\mathcal{B}}, \ldots, m_{l(M)}^{\mathcal{B}})\sigma\tau \notin s
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{authFlaw}_{\mathbb{T}}(\mathcal{A}, A, \mathcal{B}, B, M) = \\
&\quad \exists t \in \mathbb{T}.\exists \sigma(\mathit{dom}(\sigma) = \mathit{vars}(\mathcal{A})).\forall \tau(\mathit{dom}(\tau) = \mathit{vars}(\mathcal{B}) \setminus \mathit{vars}(\mathcal{A})). \\
&\qquad A\sigma \neq \mathsf{i} \wedge B\sigma \neq \mathsf{i} \wedge \\
&\qquad [\![\mathsf{state}_{\mathcal{A}}(m_0^{\mathcal{A}}, \ldots, m_n^{\mathcal{A}})\sigma]\!] \subseteq [t] \wedge [\![\mathsf{state}_{\mathcal{B}}(m_0^{\mathcal{B}}, \ldots, m_{l(M)}^{\mathcal{B}})\sigma\tau]\!] \not\subseteq [t]
\end{aligned}
$$

For the original set rewriting model, we need to enumerate all states of $\mathcal{B}$ from the state where he first learns M to his final step, since $\mathcal{B}$ may have progressed in the protocol execution already, while for the persistent set rewriting model, it is sufficient to see whether $\mathcal{B}$ was once in the state where he learned M. For the trace model, we use again the $[\![\cdot]\!]$ function to translate the goal into events. The situation is not as simple as for the secrecy goals: we cannot show directly from the Theorems of the previous chapters that these goals imply each other. The reason is that the formulation of the authentication goals has also negative conditions about the containment of facts, while the Theorems of the previous chapters tell us that the models are in an over-approximation relation, i.e. one model contains at least as much facts/events as the other.

First, we can see that the over-approximation of the persistent set rewriting model is safe:

Theorem 16.2. *For any authentication goal G, it holds that $\mathit{authFlaw}_{\mathbb{R}}(G)$ implies $\mathit{authFlaw}_{\mathbb{P}}(G)$.*

Proof. Re-inspecting the proof of Theorem 14.1, we make a stronger statement on the relationship of the set rewriting and the persistent set rewriting models: For every reachable state $s \in \mathbb{R}$, there is a state $s' \in \mathbb{P}$, such that $s' = s \cup \{\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{n-2}^{\mathcal{R}}\sigma) \mid \mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_n^{\mathcal{R}})\sigma \in s \wedge n \geq 2\}$, in other words s' is the least closure of s under prefixes of local agent states. We first prove this statement by induction over the structure of $\mathbb{R}$.

For the induction basis, the initial state is identical in $\mathbb{R}$ and $\mathbb{P}$. As it contains no state facts it is trivially prefix-closed for the agent states.

For the induction step, let $s \in \mathbb{R}$ be a reachable state, and $s' \in \mathbb{P}$ be the prefix-closure of s. To show is that every extension $t \in \mathbb{R}$ which is reachable in one transition corresponds to a similar transition from s' to $t' \in \mathbb{P}$ which is the prefix-closure of t'. If the rule applied in the transition is one of the initialization rules, then the same rule can be applied to t' adding the same new initial state fact. Since this state fact contains only the first message of the approach, it is also prefix-closed. For the case of an intruder deduction rule, the situation is the same: the same rule can be applied to s' introducing the same new facts, which cannot introduce further facts according to the prefix-closure. For the case of a transition rule of an honest agent, there may be a new state fact introduced in t and t' of the form $\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_n^{\mathcal{R}})\sigma$ such that the fact $f = \mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_{n-2}^{\mathcal{R}})\sigma$ is in s. By the induction hypothesis, $f \in s'$ and thus s' is also prefix-closed for f. As $\mathbb{P}$ is persistent, $f \in t'$, and also the other state facts from t are in t', thus t' is prefix-closed for $\mathsf{state}_{\mathcal{R}}(m_0^{\mathcal{R}}, \ldots, m_n^{\mathcal{R}})\sigma$. Since no other state facts are added and t is already prefix-closed for them, t' is prefix-closed for all state facts.

Finally, the over-approximation between the models can be shown as follows. For every state $s \in \mathbb{R}$ that violates an authentication goal G, there is a state $s' \in \mathbb{P}$ which is the prefix-closure of s by the previous induction proof. Suppose s' is not an attack state. Then there is a $\sigma(\mathit{dom}(\sigma) = \mathit{vars}(\mathcal{A}))$ such that for every $\tau(\mathit{dom}(\tau) = \mathit{vars}(\mathcal{B}) \setminus \mathit{vars}(\mathcal{A}))$ we have $\mathsf{state}_{\mathcal{A}}(m_0^{\mathcal{A}}, \ldots, m_{last^{\mathcal{A}}}^{\mathcal{A}}\sigma) \in s$, while $\mathsf{state}_{\mathcal{B}}(m_0^{\mathcal{B}}, \ldots, m_i^{\mathcal{B}}\sigma\tau) \notin s$ for any even $i \in \{l, last^{\mathcal{B}}\}$. Since s' is not an attack state, for

$$
\begin{array}{lll}
1. & \mathsf{A} \to \mathsf{B}: & \{\mathsf{NA}, \mathsf{A}\}_{\mathsf{pk(B)}} \\
2. & \mathsf{B} \to \mathsf{A}: & \{\mathsf{NA}, \mathsf{NB}, \mathsf{B}\}_{\mathsf{pk(A)}} \\
3. & \mathsf{A} \to \mathsf{B}: & \{\mathsf{NB}\}_{\mathsf{pk(B)}} \\
4. & \mathsf{B} \to \mathsf{A}: & \{\mathsf{NA}, \mathsf{NC}\}_{\mathsf{pk(A)}} \\
5. & \mathsf{A} \to \mathsf{B}: & \{\mathsf{NC}\}_{\mathsf{pk(B)}}
\end{array}
$$

$$
\begin{array}{lll}
1. & \mathsf{a} \to \mathsf{b}: & \{\mathsf{n}_1, \mathsf{a}\}_{\mathsf{pk(b)}} \\
2. & \mathsf{b} \to \mathsf{a}: & \{\mathsf{n}_1, \mathsf{n}_2, \mathsf{b}\}_{\mathsf{pk(a)}} \\
3. & \mathsf{a} \to \mathsf{b}: & \{\mathsf{n}_2\}_{\mathsf{pk(b)}} \\
1'. & \mathsf{i} \to \mathsf{b}: & \{\mathsf{n}_1, \mathsf{a}\}_{\mathsf{pk(b)}} \\
2'. & \mathsf{b} \to \mathsf{a}: & \{\mathsf{n}_1, \mathsf{n}_3, \mathsf{b}\}_{\mathsf{pk(a)}} \\
3'. & \mathsf{a} \to \mathsf{b}: & \{\mathsf{n}_3\}_{\mathsf{pk(b)}} \\
4. & \mathsf{b} \to \mathsf{a}: & \{\mathsf{n}_1, \mathsf{n}_4\}_{\mathsf{pk(a)}} \\
5. & \mathsf{a} \to \mathsf{b}: & \{\mathsf{n}_4\}_{\mathsf{pk(b)}}
\end{array}
$$

Figure 16.1: Example protocol (extension of NSL without key-server) and a trace of the protocol to demonstrate a difference in the authentication goals for $\mathbb{R}$ and $\mathbb{T}$

every $\tau[\ldots]$, we have that $\mathsf{state}_{\mathcal{B}}(m_0^{\mathcal{B}}, \ldots, m_l^{\mathcal{B}}\sigma\tau) \notin t$. This contradicts the previous conclusion that there is no such fact in t for any $i \in \{l, \ldots, last^{\mathcal{B}}\}$ and any $\tau[\ldots]$. Therefore, if s is an attack state, s' is an attack state. □

16.3 A Subtle Problem About Authentication

We will now show that the first formulation of authentication for $\mathbb{T}$ and $\mathbb{R}$ are not in general equivalent. This is due to the negation in the predicate for the local state of $\mathcal{B}$. Given an attack state of $\mathbb{R}$, we cannot conclude that the "corresponding" trace in $\mathbb{T}$ is still an attack trace: as $\mathbb{T}$ is an over-approximation of $\mathbb{R}$, it may happen that the negative condition (that $\mathcal{B}$ is not in a certain state) does no longer hold in $\mathbb{T}$. Thus, more generally, we cannot be sure that verifying authentication in the $\mathbb{T}$ model indeed implies authentication in the $\mathbb{R}$ model.

It is relatively straightforward to come up with an example of a protocol and an attack state $s \in \mathbb{R}$ (for authentication) such that the corresponding trace $t \in \mathbb{T}$ is not a violation of the authentication goal. However, when constructing such an example, the protocol usually has many different similar attack states, and usually some of them can be recast into an attack trace in the $\mathbb{T}$ model. The difficulty is thus to find an example of a protocol with an attack state in $\mathbb{R}$, but no attack trace in $\mathbb{T}$. We could only find quite contrived examples of protocols with this property, one being displayed in Figure 16.1. The point of this example is not to provide an interesting protocol or attack, but to demonstrate that from authentication in $\mathbb{T}$, we cannot conclude authentication in $\mathbb{R}$. More formally, the example demonstrates:

$$authFlaw_{\mathbb{P}}(G) \not\Rightarrow authFlaw_{\mathbb{T}}(G) \ .$$

Example 16.1. Consider again the NSL protocol (without key-server), but now augmented with an additional challenge-response exchange as in Figure 16.1. Consider further the goal that $\mathcal{B}$ wants to authenticate $\mathcal{A}$ on the concatenation of nonces $(\mathsf{NA}, \mathsf{NB}, \mathsf{NC})$, i.e.

$$G = (\mathcal{B}, \mathsf{B}, \mathcal{A}, \mathsf{A}, (\mathsf{NA}, \mathsf{NB}, \mathsf{NC})) \ .$$

This goal is violated in the $\mathbb{R}$ model as shown by the attack in Figure 16.1. The attack begins with a normal exchange of the first three messages between a and b, after which the intruder sends the first message again to b to obtain answers ($2'$ and $3'$) with a new nonce n_3. The final two steps 4 and 5 can now be interpreted as a continuation either of steps 1–3 or of steps $1'$–$3'$. (Thus, labeling the final steps as $4'$ and $5'$ in the example would also be correct.) In particular, the two agents may have a different interpretation: while b could have meant step 4 as an answer to step 3, a might think it was a continuation of step $3'$. This discrepancy can be detected, i.e. there is a reachable state of $\mathbb{R}$ that contains agent b in a local state while a is not in a corresponding local state.

For this example protocol and goal, however, the message trace model does not have any flaws, as we show in the following proof sketch. We use the fact that the intruder knows no

private keys but his own. It follows that, if an agent creates a nonce n and sends it as part of a message m encrypted with the public-key of an honest agent a, then n can only appear in a later event, if the message m was received and processed by a. Consider the situation that an honest agent b in role $\mathcal{B}$ received $\{\mathsf{n}_2\}_{\mathsf{pk(b)}}$ and $\{\mathsf{n}_3\}_{\mathsf{pk(b)}}$ for two nonces n_2 and n_3 that he sent earlier as part of messages 2 and 4, i.e. he earlier sent $\{\mathsf{n}_1, \mathsf{n}_2, \mathsf{b}\}_{\mathsf{pk(a)}}$ and $\{\mathsf{n}_1, \mathsf{n}_3\}_{\mathsf{pk(a)}}$ for some a and n_1, and also b has previously received the message $\{\mathsf{n}_1, \mathsf{a}\}_{\mathsf{pk(a)}}$. Then a must have eventually received the said messages of step 2 and 4 from b, namely $\{\mathsf{n}_1, \mathsf{n}_2, \mathsf{b}\}_{\mathsf{pk(a)}}$ and $\{\mathsf{n}_1, \mathsf{n}_3\}_{\mathsf{pk(a)}}$ and answered to it. This answer must be $\{\mathsf{n}_2\}_{\mathsf{pk(b)}}$ and $\{\mathsf{n}_3\}_{\mathsf{pk(b)}}$, respectively, due to the protocol. Thus, $[\![\mathsf{state}_{\mathcal{A}}(\mathsf{a}, \mathsf{b}, \mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_3)]\!]$ must be a subset of the trace. So when $\mathcal{B}$ has reached a successful end of the protocol run in $\mathbb{T}$, the $\mathcal{A}$ must agree with it on the agent identities and nonces.

It thus holds for the example protocol that the message trace model contains no attack trace, while there is an attack state in the $\mathbb{P}$ model. This demonstrates that $\mathit{authFlaw}_{\mathbb{P}}(G)$ does not imply $\mathit{authFlaw}_{\mathbb{T}}(G)$. □

While neither the protocol nor the attack are of practical interest, this example demonstrates that for some goals the over-approximation is not sound, i.e. we cannot rely that safety of the over-approximated model implies safety of the original model. However, one may wonder how serious this problem is in practice; in particular, whether a soundness result for authentication goals can still be obtained for a restricted class of protocols. For instance, one could require that every message of the protocol must contain a hash-value of all previous messages of the protocol run. Such a restriction would however severely limit the class of protocols that can be considered.

It is worth to note that we have discovered this problem with authentication only after formally defining the models and goals, and failure to prove the claim that $\mathbb{T}$ has at least as many attacks as $\mathbb{R}$. Indeed we see this as part of understanding better the models that we are working with.

16.4 Alternative Formulation of Authentication

The conclusions that might be drawn from the problem with authentication is that the message trace model $\mathbb{T}$ is not suitable for authentication goals at all: for authentication we need to know which message was sent in which "context"—i.e. the local state of the honest agent sending it—while in general we cannot unambiguously recover this context from the exchanged messages alone, as we have seen in the examples.

To avoid any such problems with goals for message traces, it seems obvious to add more explicit information to the message trace model. We recall how authentication was defined for the Intermediate Format in Section 4.5. The witness facts and request facts act as a kind of interface between protocol and goals, in particular, the formulation of the goals is independent from the particularities of the protocol (like the exchanged messages).

To easily integrate these two kinds of facts into $\mathbb{R}$, $\mathbb{P}$, and $\mathbb{T}$, without changing the models and such that we can reuse the Theorems proved about them, we make the special events part of the exchanged messages. To that end, let $\mathsf{witness}(\cdot)$ and $\mathsf{request}(\cdot)$ be two function symbols of Σ that are used merely for this purpose. Further, there are no intruder deduction rules for the symbols $\mathsf{witness}(\cdot)$ and $\mathsf{request}(\cdot)$, i.e. the intruder can neither "decrypt" such a "message", nor construct one himself. Figure 16.2 shows how the witness facts and request facts can be added to the messages of the protocol itself, appearing only in the message patterns of the role that creates the event. pNA and pNB are constants used to identify the purpose of the exchanged messages in the protocol, in the example distinguishing whether a nonce was meant as NA or NB.

Using the new events, we can now formulate authentication for the models $\mathbb{R}$ and $\mathbb{T}$ in a protocol-independent way:[1]

[1]Note that in contrast to the formalization of authentication goals in Section 16.2, there are no parameters in this case; this difference results from the fact that in the new formalization the events $\mathsf{witness}(\cdot)$ and $\mathsf{request}(\cdot)$ carry enough information to get all the information from there, which was necessarily a parameter in the original formulation.

Role $\mathcal{A}$		Role $\mathcal{B}$	
$m_0^{\mathcal{A}}$	$\mathsf{A}, \mathsf{B}, \mathsf{S}$	$m_0^{\mathcal{B}}$	B, S
$m_1^{\mathcal{A}}$	start		
$m_2^{\mathcal{A}}$	$\{\mathsf{NA}, \mathsf{A}\}_{\mathsf{pk(B)}}, \mathsf{witness}(\mathsf{A}, \mathsf{B}, \mathsf{pNA}, \mathsf{NA})$	$m_1^{\mathcal{B}}$	$\{\mathsf{NA}, \mathsf{A}\}_{\mathsf{pk(B)}}$
$m_3^{\mathcal{A}}$	$\{\mathsf{NA}, \mathsf{NB}, \mathsf{B}\}_{\mathsf{pk(A)}}$	$m_2^{\mathcal{B}}$	$\{\mathsf{NA}, \mathsf{NB}, \mathsf{B}\}_{\mathsf{PKA}}, \mathsf{witness}(\mathsf{B}, \mathsf{A}, \mathsf{pNB}, \mathsf{NB})$
$m_4^{\mathcal{A}}$	$\{\mathsf{NB}\}_{\mathsf{pk(B)}}, \mathsf{request}(\mathsf{A}, \mathsf{B}, \mathsf{pNB}, \mathsf{NB})$	$m_3^{\mathcal{B}}$	$\{\mathsf{NB}\}_{\mathsf{pk(B)}}$
		$m_4^{\mathcal{B}}$	$\mathsf{end}, \mathsf{request}(\mathsf{B}, \mathsf{A}, \mathsf{pNA}, \mathsf{NA})$

Figure 16.2: NSL (without key-server) augmented with the special events for the authentication goals.

$$\begin{aligned} &\mathit{authFlaw}_{\mathbb{R}} = \exists s \in \mathbb{R}.\, \exists \sigma(\mathit{dom}(\sigma) = \{\mathsf{A}, \mathsf{B}, \mathsf{P}, \mathsf{M}\}). \\ &\quad \mathsf{A}\sigma \neq \mathsf{i} \wedge \mathsf{B}\sigma \neq \mathsf{i} \wedge \mathsf{request}(\mathsf{A}, \mathsf{B}, \mathsf{P}, \mathsf{M})\sigma \in \mathcal{DY}(s) \wedge \mathsf{witness}(\mathsf{B}, \mathsf{A}, \mathsf{P}, \mathsf{M})\sigma \notin \mathcal{DY}(s) \end{aligned}$$

$$\begin{aligned} &\mathit{authFlaw}_{\mathbb{T}} = \exists t \in \mathbb{T}.\, \exists \sigma(\mathit{dom}(\sigma) = \{\mathsf{A}, \mathsf{B}, \mathsf{P}, \mathsf{M}\}). \\ &\quad \mathsf{A}\sigma \neq \mathsf{i} \wedge \mathsf{B}\sigma \neq \mathsf{i} \wedge \mathsf{request}(\mathsf{A}, \mathsf{B}, \mathsf{P}, \mathsf{M})\sigma \in \mathcal{DY}([t]) \wedge \mathsf{witness}(\mathsf{B}, \mathsf{A}, \mathsf{P}, \mathsf{M})\sigma \notin \mathcal{DY}([t]) \end{aligned}$$

The formulation for $\mathbb{P}$ is identical with the one for $\mathbb{R}$, except that $s \in \mathbb{P}$.

We can now finally conclude that all authentication flaws of the $\mathbb{R}$ model can also be found in the $\mathbb{T}$ model, and thus see the completeness of the over-approximation for authentication flaws formulated over $\mathsf{witness}(\cdot)$ and $\mathsf{request}(\cdot)$ events:

Theorem 16.3.

1. *$\mathit{authFlaw}_{\mathbb{R}}$ implies $\mathit{authFlaw}_{\mathbb{P}}$, and*
2. *$\mathit{authFlaw}_{\mathbb{P}}$ implies $\mathit{authFlaw}_{\mathbb{T}}$.*

Proof.

1. Let $s \in \mathbb{R}$ be an attack state that satisfies $\mathit{authFlaw}_{\mathbb{R}}$. By Theorem 14.1, there is a state $s' \in \mathbb{P}$ with $s' \supseteq s$ and $\mathcal{DY}(s') = \mathcal{DY}(s)$. Since both states have the same intruder knowledge, they in particular contain the same $\mathsf{witness}(\cdot)$ and $\mathsf{request}$ facts. Thus s' is an attack state, and $\mathit{authFlaw}_{\mathbb{P}}$ holds.
2. Let $s \in \mathbb{P}$ be an attack state that satisfies $\mathit{authFlaw}_{\mathbb{P}}$. By Theorem 14.2, there is a trace $t \in \mathbb{T}$, such that $\mathcal{DY}(s) = \mathcal{DY}([t])$. Again, since $[t]$ has the same intruder knowledge as s, it contains the same $\mathsf{witness}(\cdot)$ and $\mathsf{request}$ facts, thus $[t]$ is an attack trace and $\mathit{authFlaw}_{\mathbb{T}}$ holds.

□

Authentication in the Reachable Facts/Events Models. In the reachable facts and events models, the formulation of authentication goals is even more difficult. The reason is that in these models, all states (or all traces) are merged into one. Suppose there is a reachable attack state $s \in \mathbb{P}$, i.e. there are certain facts f_1 and f_2, such that $f_1 \in s$ and $f_2 \notin s$. If there is a reachable state $s' \in \mathbb{P}$ such that $f_2 \in s'$, then $f_1, f_2 \in \mathbb{F}$, and the attack predicate would fail on $\mathbb{F}$. Note that Theorem 15.2 guarantees for this case only that there is a state $s \in \mathbb{P}$ such that $f_1, f_2 \in s$, i.e. that these facts are not exclusive on all states.

We thus need different ways to formulate authentication goals in the reachable facts and event models. When data abstraction is performed, i.e. when the protocol has no fresh data anymore,

then we can exploit the way the abstract terms that replace the fresh data are generated. As an example, consider again the NSL example, the goal that $\mathcal{A}$ authenticates $\mathcal{B}$ on NB, and the data abstraction $\mathsf{nb}(\mathsf{B},\mathsf{A})$ as the nonce created by B for A. Then this authentication goal can be expressed as follows:

$$\exists\sigma(dom(\sigma)=\{\mathsf{A},\mathsf{B},\mathsf{M}\}).\mathsf{A}\sigma\neq\mathsf{i}\wedge\mathsf{B}\sigma\neq\mathsf{i}\wedge\mathsf{request}(\mathsf{A},\mathsf{B},\mathsf{pNB},\mathsf{M})\sigma\in\mathcal{DY}(\mathbb{F})\wedge\mathsf{M}\sigma\neq\mathsf{nb}(\mathsf{B},\mathsf{A})\sigma$$

This goal states that it is an authentication flaw if $\mathcal{A}$ finishes the protocol with any value other than $\mathsf{nb}(\mathsf{B},\mathsf{A})$ for the nonce from $\mathcal{B}$. Intuitively, the form $\mathsf{nb}(\mathsf{B},\mathsf{A})$ of nonce NB can be seen similarly to a witness-event: NB has this form iff it has been created by B for A for the purpose pNB (i.e. for the protocol variable NB). The construction of the abstract data thus allows us to declaratively see who created the data for whom and for what purpose.

More generally, consider a protocol with data abstraction (i.e. no fresh data) and labeled with witness facts and request facts as desired. We require that we are given a description of the abstraction by a function $\lceil\cdot\rceil$ that maps from purposes to terms, and two functions $sndr(\cdot)$ and $rcvr(\cdot)$ from purposes to variables. These functions express which protocol variables get abstracted into which terms, and who is the creator and the intended recipient. E.g. in the NSL example we might have $\lceil\mathsf{pNA}\rceil=\mathsf{na}(\mathsf{A},\mathsf{B})$, $\lceil\mathsf{pNB}\rceil=\mathsf{nb}(\mathsf{B},\mathsf{A})$, $sndr(\mathsf{pNA})=rcvr(\mathsf{pNB})=A$, and $sndr(\mathsf{pNB})=rcvr(\mathsf{pNA})=A$.

We say that a protocol description is *in accordance with the data abstraction* $\lceil\cdot\rceil$, iff the following holds. For every reachable state of $s\in\mathbb{R}$, for every substitution σ, and for every purpose p, if $\lceil p\rceil\sigma$ is a subterm of s, then $\mathsf{witness}(sndr(p)\sigma,rcvr(p)\sigma,p,\lceil p\rceil\sigma)\in s$. For instance, NSL is in accordance with $\lceil\cdot\rceil$, since every occurrence of $\mathsf{na}(\cdot)$ or $\mathsf{nb}(\cdot)$ is accompanied by a suitable witness fact. Intuitively, the accordance condition ensures everything we need to get rid of the witness facts in the reachable facts and events models: they express that from the value used for purpose p, we can recognize whether there is a witness fact for this term, i.e. whether this term was indeed produced in the required way.

Let P be the set of all purposes (for which witness facts are created). We formulate authentication for the reachable facts model as follows:

$$\begin{aligned}&authFlaw_{\mathbb{F}}=\exists\mathcal{R}.\,\exists p\in P.\,\exists\sigma(dom(\sigma)=vars(\mathcal{R})\cup\mathsf{T}\}).\\&\quad\mathsf{request}(rcvr(p)\sigma,sndr(p)\sigma,p,\mathsf{T}\sigma)\in\mathcal{DY}(\mathbb{F})\wedge\mathsf{T}\sigma\neq\lceil p\rceil\sigma\wedge rcvr(p)\sigma\neq\mathsf{i}\wedge sndr(p)\sigma\neq\mathsf{i}\,.\end{aligned}$$

The formulation for the reachable events model is identical except when replacing $\mathbb{E}$ for $\mathbb{F}$.

Finally, we can show that also the reachable facts and events models are complete for the checking of authentication goals in this formulation:

Theorem 16.4. *For a protocol with data-abstraction in accordance with the abstraction function* $\lceil\cdot\rceil$*,* $authFlaw_{\mathbb{P}}$ *implies* $authFlaw_{\mathbb{F}}$ *and* $authFlaw_{\mathbb{E}}$*.*

Proof. Let $s\in\mathbb{P}$ be an attack state. In particular, s contains a request term with honest participants and without matching witness. By Theorem 15.2, we know that there is a subset $s'\subseteq\mathbb{F}$ of the reachable facts, with $s\subseteq s'$, thus s' contains the same request fact. Note that this request fact has honest agents as its first two arguments. To arrive at a contradiction, assume $authFlaw_{\mathbb{F}}$ does not hold. Since s' already contains a request-fact between two honest principals, we conclude $\mathsf{T}\sigma=\lceil p\rceil\sigma$ (otherwise $authFlaw_{\mathbb{F}}$ where true). This means that s and s' contain a term $\lceil p\rceil\sigma$, which is an abstraction. By the accordance condition, we have that s must contain the fact $\mathsf{witness}(sndr(p)\sigma,rcvr(p)\sigma,p,\lceil p\rceil\sigma)$. This means that s is not an attack state, contradicting our assumption. □

This concludes our formal analysis of the relationships between protocol models and the goals between them. The main result is that there is an over-approximation relation from the set rewriting model $\mathbb{R}$ over the persistent set rewriting model $\mathbb{P}$ to the message trace model $\mathbb{T}$ and, for data-abstract protocols, all these models are over-approximated by the reachable facts model $\mathbb{F}$ and the reachable events model $\mathbb{E}$. The over-approximation relation can directly be used for verifying secrecy under over-approximation, i.e. if secrecy holds for the over-approximation then

it holds in all more precise models. The same does not hold for authentication goals due to the negation present in authentication goals, but we have shown that, with the help of auxiliary messages for authentication goals, we obtain a way to use the over-approximation relation in the same way as for secrecy goals.

Part V

Experimental Results, Related Work, and Conclusions

Chapter 17

Case Studies

As stated in the introduction, the main goal of this thesis is to develop and improve methods for the automated analysis of security protocols, and to provide designers of security protocols with an efficient, flexible, and widely applicable tool to validate, or find flaws in, their protocols before deployment. We now demonstrate that we have achieved this goal with the tool OFMC that implements the methods we have described in this thesis.

We first summarize a selection of the case studies that we have performed with OFMC. While we have used small academic examples like the Yahalom protocol to illustrate our ideas in the previous parts, we now consider "real-world" protocols that are considerably more complex and that are relevant in the sense that they are currently deployed in the Internet and in mobile communication. We consider the following examples:

- The H.530 protocol is an example of a protocol with complex message terms on which OFMC has detected a flaw that lead to a change of the protocol by its developer Siemens.
- The contract signing protocol ASW is an example of a protocol that has a complex structure in that it consists of several subprotocols and uses a trusted third party that must maintain a data-base of contracts it has seen. Also it has the non-trivial goal of a fair exchange of contracts.
- The SRP protocol is an example of a protocol that relies on a number of algebraic properties, and we thus illustrate the algebraic intruder deductions of OFMC by this example.

Other case studies such as [20] have been omitted for brevity.

After the case studies we have a closer look at the performance of OFMC on two large libraries of protocols, namely the Clark/Jacob library [54] and the AVISPA library [13], both of which have been used as a test-suite during the development of OFMC. This also gives us the opportunity to compare OFMC's performance with the other analysis tools developed during the AVISPA project [7] and which are also based on the Intermediate Format.

17.1 H.530

H.530 [92, 93], which has been developed by Siemens, provides mutual authentication and key agreement in mobile roaming scenarios in multimedia communication. H.530 is deployed as shown in the left part of Figure 17.1: a mobile terminal (MT) wants to establish a secure connection and negotiate a Diffie-Hellman key with the gatekeeper (VGK) of a visited domain. As they do not know each other in advance, the authentication is performed using an authentication facility AuF within the home domain of the MT. Both MT and VGK initially have shared keys with AuF.

Figure 17.2 shows a simplified description of H.530 in A&B notation. The protocol works as follows. There is initially a shared key between the mobile terminal MT and its home server AuF, denoted $\mathsf{k}(\mathsf{MT}, \mathsf{AuF})$, as well as a shared key between the visited gate-keeper VGK and AuF,

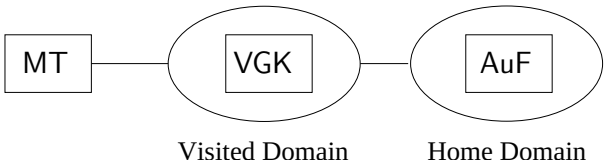


Figure 17.1: The deployment of the H.530 protocol.

denoted $\mathsf{k}(\mathsf{VGK}, \mathsf{AuF})$. The goal is to authenticate a Diffie-Hellman key-exchange between MT and VGK. Observe that this goal cannot be achieved if AuF is not honest, AuF thus acts as a trusted third party.

In the first message, the MT sends out a request that contains a fresh Diffie-Hellman half-key $\mathsf{exp}(\mathsf{g}, \mathsf{DHx})$. This message, like all the following, is "MACed" (Message Authentication Code): a hash-value of the message using a *keyed* hash-function is added to the message. A keyed hash-function is like a normal hash-function, but that has as an extra parameter a "symmetric" key, so that only participants who know the key can construct—or check—the hash-value. Our model of a MAC is based on using a simple implementation using an unkeyed hash-function: the key is concatenated together with the message to hash. We do not need to discuss the cryptographic requirements and implications of such an implementation, since in our model the MAC has exactly the properties we want: one can build a MAC iff one knows the key and the MACed message, and one cannot recover from a MAC the MACed message (even if one knows the key).

To continue with the protocol description, the first message from MT is MACed using the key $\mathsf{k}(\mathsf{MT}, \mathsf{AuF})$, thus the receiver VGK can read the Diffie-Hellman half-key and the name of MT (at least what it seems to be), but cannot check the authenticity of the messages. In the second message, VGK forwards this request to AuF, including his own fresh Diffie-Hellman half-key $\mathsf{exp}(\mathsf{g}, \mathsf{DHy})$ which is XOR-ed to the half-key from MT.[1]

After having checked that all MACs are "adding up", the AuF answers in the third message with an acknowledgment, which contains both half-keys and the name of the participants. As in message 2, we have two nested MACs, the outer one with $\mathsf{k}(\mathsf{VGK}, \mathsf{AuF})$ and the inner one with $\mathsf{k}(\mathsf{MT}, \mathsf{AuF})$. Observe, however, that this time the inner one is without a copy of the clear-text. That is exactly the weakness of the protocol that we will describe below. The last two messages between MT and VGK are MACed using the new Diffie-Hellman key of MT and VGK, proving that both can construct the key. Note that MT receives the half-key from VGK also in clear-text, so that he can build the key to check the hash-value used here.

A research group at Siemens had informally analyzed the protocol, and also tried for months to formally verify the protocol. They used Casper/FDR that failed at the complexity of even drastically simplified versions, or found nonsensical attacks when the input specification was simplified too far.

The specification in OFMC took one work-day[2] and after an analysis time of 1.3 seconds, OFMC reported a replay attack, displayed in Figure 17.3, which works as follows. First the intruder listens to session between a set of honest agents mt, vgk, and auf. He uses the recorded messages later to "steal" the identity of mt, i.e. to pose mt towards vgk and negotiate a new Diffie-Hellman key with it. More in detail, in message $1'$, vgk can only see the clear-text part of the message, the intruder can thus insert anything for the keyed hash (denoted rand here). Note that the intruder creates a fresh Diffie-Hellman secret dhz here. The intruder intercepts message $2'$ from vgk to auf and for the reply $3'$ from auf, he replays the message 3 from the previous session, which is based on the old Diffie-Hellman secrets dhx and dhy. Observe that vgk cannot

[1] This is probably not to hide half-keys from an observer, since both the half-key from MT as well as the XOR-ed pair of half-keys are transmitted in clear-text, so that every observer can obtain also $\mathsf{exp}(\mathsf{g}, \mathsf{DHy})$. Probably the reason is rather to bind both half-keys together in case the MAC has some unfortunate streaming block cipher properties. We chose not to go into a more cryptographically precise model here to analyze this for more low-level cryptographic attacks but sticked with the black-box view of a MAC.

[2] At the time, OFMC did not support algebraic properties, so a work-around for Diffie-Hellman had to be implemented and XOR was replaced with concatenation.

1. $\mathsf{MT} \to \mathsf{VGK}: \quad \mathsf{req}(\mathsf{MT}, \mathsf{VGK}, \mathsf{AuF}, \mathsf{exp}(\mathsf{g}, \mathsf{DHx}))$
2. $\mathsf{VGK} \to \mathsf{AuF}: \quad \mathsf{mac}(\mathsf{k}(\mathsf{VGK}, \mathsf{AuF}), (\mathsf{req}(\mathsf{MT}, \mathsf{VGK}, \mathsf{exp}(\mathsf{g}, \mathsf{DHx})), \mathsf{VGK}, \mathsf{ver}(\mathsf{DHx}, \mathsf{DHy})))$
3. $\mathsf{AuF} \to \mathsf{VGK}: \quad \mathsf{mac}(\mathsf{k}(\mathsf{VGK}, \mathsf{AuF}), (\mathsf{VGK}, \mathsf{ack}(\mathsf{MT}, \mathsf{VGK}, \mathsf{AuF}, \mathsf{DHx}, \mathsf{DHy})))$
4. $\mathsf{VGK} \to \mathsf{MT}: \quad \mathsf{mac}(\mathsf{dhk}(\mathsf{DHx}, \mathsf{DHy}), (\mathsf{ack}(\mathsf{MT}, \mathsf{VGK}, \mathsf{AuF}, \mathsf{DHx}, \mathsf{DHy}), \mathsf{exp}(\mathsf{g}, \mathsf{DHy})))$
5. $\mathsf{MT} \to \mathsf{VGK}: \quad \mathsf{mac}(\mathsf{dhk}(\mathsf{DHx}, \mathsf{DHy}), (\mathsf{MT}, \mathsf{VGK}))$

where
$$\begin{aligned}
\mathsf{mac}(\mathsf{K}, \mathsf{M}) &= (\mathsf{M}, \mathsf{f}(\mathsf{K}, \mathsf{M})) \\
\mathsf{dhk}(\mathsf{DHx}, \mathsf{DHy}) &= \mathsf{exp}(\mathsf{exp}(\mathsf{g}, \mathsf{DHx}), \mathsf{DHy}) \\
\mathsf{ver}(\mathsf{DHx}, \mathsf{DHy}) &= \mathsf{xor}(\mathsf{exp}(\mathsf{g}, \mathsf{DHx}), \mathsf{exp}(\mathsf{g}, \mathsf{DHy})) \\
\mathsf{req}(\mathsf{MT}, \mathsf{VGK}, \mathsf{AuF}, \mathsf{DHx}) &= \mathsf{mac}(\mathsf{k}(\mathsf{MT}, \mathsf{AuF}), (\mathsf{MT}, \mathsf{VGK}, \mathsf{exp}(\mathsf{g}, \mathsf{DHx}))) \\
\mathsf{ack}(\mathsf{MT}, \mathsf{VGK}, \mathsf{AuF}, \mathsf{DHx}, \mathsf{DHy}) &= \mathsf{f}(\mathsf{k}(\mathsf{MT}, \mathsf{AuF}), (\mathsf{VGK}, \mathsf{ver}(\mathsf{DHx}, \mathsf{DHy})))
\end{aligned}$$

Figure 17.2: A&B description of H.530 (simplified).

[Normal session of the protocol (recorded by the intruder)]

$1'$. $\mathsf{i} \to \mathsf{vgk}: \quad \mathsf{mt}, \mathsf{vgk}, \mathsf{auf}, \mathsf{exp}(\mathsf{g}, \mathsf{dhz}), \mathsf{rand}$
$2'$. $\mathsf{vgk} \to \mathsf{i}: \quad \mathsf{mac}(\mathsf{k}(\mathsf{vgk}, \mathsf{auf}), (\mathsf{mt}, \mathsf{vgk}, \mathsf{auf}, \mathsf{exp}(\mathsf{g}, \mathsf{dhz}), \mathsf{rand}, \mathsf{vgk}, \mathsf{ver}(\mathsf{z}, \mathsf{exp}(\mathsf{g}, \mathsf{dhy}_2))))$
$3'$. $\mathsf{i} \to \mathsf{vgk}: \quad \mathsf{mac}(\mathsf{k}(\mathsf{vgk}, \mathsf{auf}), (\mathsf{vgk}, \mathsf{ack}(\mathsf{mt}, \mathsf{vgk}, \mathsf{auf}, \mathsf{dhx}, \mathsf{dhy})))$
$4'$. $\mathsf{vgk} \to \mathsf{i}: \quad \mathsf{mac}(\mathsf{dhk}(\mathsf{dhz}, \mathsf{dhy2}), (\mathsf{ack}(\mathsf{mt}, \mathsf{vgk}, \mathsf{auf}, \mathsf{dhx}, \mathsf{dhy}), \mathsf{exp}(\mathsf{g}, \mathsf{dhy2})))$
$5'$. $\mathsf{i} \to \mathsf{vgk}: \quad \mathsf{mac}(\mathsf{dhk}(\mathsf{dhz}, \mathsf{dhy2}), (\mathsf{mt}, \mathsf{vgk}))$

Figure 17.3: An Attack on H.530, where rand and dhz are random values created by the intruder, and dhy2 is value created by vgk for DHy in the second run of the protocol.

detect this, because these old values are only contained in the ack part of the message, which is a keyed hash using $\mathsf{k}(\mathsf{mt}, \mathsf{auf})$. Thus, from vgk's point of view, the auf has just acknowledged the key-exchange with mt, namely the using the Diffie-Hellman key $\mathsf{dhk}(\mathsf{dhz}, \mathsf{dhy2})$. This key is known to the intruder, since he created dhz himself and can read $\mathsf{exp}(\mathsf{g}, \mathsf{dhy2})$ for instance from the clear-text part of message $2'$. Therefore he can complete the attack and vgk believes to share the new Diffie-Hellman key with mt.

Due to this attack, Siemens has changed the protocol following our suggestion to include the Diffie-Hellman half-keys in the MAC for VGK in message 3 [93]. The point of this example is not the precise running time, but to demonstrate that OFMC scales well with the complexity of protocols. In particular, a large variety of tools scale poorly with the complexity of message terms, because this blows up the set of possible message terms that the intruder can send or the encoding of this set. For OFMC, the complexity of message terms is not a problem, thanks to the lazy intruder technique. Therefore, we can analyze protocols without first making several simplifications which bear the danger that the protocol gets trivialized and possible attacks cannot be detected in the simplified version anymore. Also the fact that the response time for OFMC on many problems is within seconds, has the advantage that it is possible to experiment with different specifications, trying slightly different versions of the specification, which is not done by users if they have to wait a day for the results.

17.2 Asokan-Shoup-Waidner

We come to an example of a protocol that is challenging in terms of modeling, for what concerns both the honest agents and the goals: Asokan-Shoup-Waidner (ASW), an optimistic fair exchange protocol for contract signing [12]. The main objective of contract signing protocols is to allow their users to digitally sign contracts without having to meet. *Fair exchange* means here that neither

Exchange subprotocol:

1. $\mathsf{O} \to \mathsf{R}: \quad me_1 = \mathsf{Sig}_{\mathsf{O}}(\mathsf{V}_{\mathsf{O}}, \mathsf{V}_{\mathsf{R}}, \mathsf{T}, \mathsf{text}, \mathsf{h}(\mathsf{N}_{\mathsf{O}}))$
2. $\mathsf{R} \to \mathsf{O}: \quad me_2 = \mathsf{Sig}_{\mathsf{R}}(\mathsf{me}_1, \mathsf{h}(\mathsf{N}_{\mathsf{R}}))$
3. $\mathsf{O} \to \mathsf{R}: \quad \mathsf{N}_{\mathsf{O}}$
4. $\mathsf{R} \to \mathsf{O}: \quad \mathsf{N}_{\mathsf{R}}$

Abort subprotocol:

1. $\mathsf{O} \to \mathsf{T}: \quad ma_1 = \quad \mathsf{Sig}_{\mathsf{O}}(\mathsf{aborted}, me_1)$
2. $\mathsf{T} \to \mathsf{O}: \quad ma_2 = \quad \mathsf{if}\ \mathit{resolved}(me_1)\ \mathsf{then}\ \mathsf{Sig}_{\mathsf{T}}(me_1, me_2)$
   $\mathsf{else}\ \mathsf{Sig}_{\mathsf{T}}(\mathsf{aborted}, ma_1)\ ;\ \mathit{aborted}(ma_1) \ = \ \mathit{true}$

Resolve subprotocol:

1. $\mathsf{O} \to \mathsf{T}: \quad mr_1 = \quad me_1, me_2$
2. $\mathsf{T} \to \mathsf{O}: \quad mr_2 = \quad \mathsf{if}\ \mathit{aborted}(me_1)\ \mathsf{then}\ \mathsf{Sig}_{\mathsf{T}}(\mathsf{aborted}, me_1)$
   $\mathsf{else}\ \mathsf{Sig}_{\mathsf{T}}(me_1, me_2)\ ;\ \mathit{resolved}(me_1) \ = \ \mathit{true}$

Figure 17.4: The Subprotocols of ASW

party gains an advantage over the other. For instance, consider a simple email-based exchange of digitally signed contracts. Then the party who has first received the contract from the other (but not yet sent a signed copy itself) is in advantage, since it can now take any time it wants to decide whether to sign it or not, while the other party cannot pull out anymore. This could be abused by a dishonest party to negotiate a better contract with a competitor. It is intuitively clear that a trusted third party (T3P) is needed to obtain a fair exchange.

In ASW, a trusted third party is involved *only* if dispute resolution is required (hence the term *optimistic*, which differentiates this approach from others in which an online trusted party is involved in every exchange). In resolving disputes, the T3P issues either a *replacement contract* asserting that he recognizes the contract in question as valid, or an *abort token* asserting that he has never issued, and will never issue, a replacement contract. An important requirement of the protocol is that the intruder cannot block messages between an honest agent and the T3P forever.

The requirements for fair exchange are often stated in terms of liveness properties of the form "if one agent has a valid contract, then the other either has one as well or is in the position to eventually obtain one." Liveness properties are problematic for a variety of verification approaches, in particular those involving infinite state-spaces. One often approximates liveness properties via safety properties, i.e. if the protocol satisfies the safety property, then it also satisfies the liveness property that was approximated, as it is for instance done in [124].

The protocol, shown in Figure 17.4, consists of three subprotocols: *exchange*, *abort*, and *resolve*. The former involves only the two protocol participants, the originator O and the responder R, while the latter two are only executed if the trusted third party T is called upon to resolve a dispute. Our notational conventions are as follows: $\mathsf{Sig}_A(M)$ abbreviates $\mathsf{sign}(\mathsf{inv}(\mathsf{V}_A), M)$, i.e. the digital signature of message M by agent A, whose public key for signature verification is V_A. The contractual text we call *text*. During the protocol, each party generates a nonce, which we write N_{O} and N_{R} for the originator and responder, respectively. Finally, the function h is a cryptographic hash function which is assumed to be collision resistant. We note that the protocol defines two kinds of valid contracts: either the *standard contract* as it is obtained by the exchange subprotocol, or a *replacement contract* issued by the T3P, and both hold equal validity.

The Exchange Subprotocol: If both participants are honest and in the absence of network failures or intruder intervention, after execution of the exchange subprotocol, both will be in possession of a valid standard contract.

Both originator and responder generate nonces N_{O} and N_{R} which are called their respective *secret commitments* to the contract. Given these, they compute their so-called *public commitments* by hashing these values, yielding $\mathsf{h}(\mathsf{N}_{\mathsf{O}})$ and $\mathsf{h}(\mathsf{N}_{\mathsf{R}})$, respectively. The protocol then proceeds in two rounds: in the first, each party expresses his public commitment to the agreed-upon contract but does not disclose his secret commitment. In the second round, they then exchange their

$exchange_1$. $\mathsf{O} \rightarrow \mathsf{R}$:	me_1
if *timeout* then	$abort_1$. $\mathsf{O} \rightarrow \mathsf{T} : ma_1$
	$abort_2$. $\mathsf{T} \rightarrow \mathsf{O} : ma_2$ (*abort token* or *replacement contract*)
else	
$exchange_2$. $\mathsf{R} \rightarrow \mathsf{O}$:	me_2
$exchange_3$. $\mathsf{O} \rightarrow \mathsf{R}$:	$\mathsf{N_O}$
if *timeout* then	$resolve_1$. $\mathsf{O} \rightarrow \mathsf{T} : mr_1$
	$resolve_2$. $\mathsf{T} \rightarrow \mathsf{O} : mr_2$ (*abort token* or *replacement contract*)
else	
$exchange_4$. $\mathsf{R} \rightarrow \mathsf{O}$:	$\mathsf{N_R}$

Figure 17.5: Originator role under our unified view of the protocol.

respective secret commitments. Each party can then hash this latter and thus verify that the purported secret commitment he receives indeed corresponds to the public commitment from the first protocol stage. At the end of this exchange, each party is in possession of a valid standard contract of the form $me_1, me_2, \mathsf{N_O}, \mathsf{N_R}$.

The Abort Subprotocol: If O does not receive R's reply me_2 within an acceptable time frame (where the definition of "acceptable" is left entirely up to O), he may abort the protocol by invoking the trusted third party. He sends a signed abort request ma_1 indicating that he wishes to abort the exchange.

The T3P is assumed to maintain a permanent database of contracts for which he has been called upon to arbitrate. If he has already asserted the validity of the contract (indicated by $resolved(me_1)$), then he sends the originator a *replacement contract* of the form $Sig_T(me_1, me_2)$. Otherwise, he replies with a so-called *abort token*, signing the originator's abort request and adding an entry in his database of aborted contracts. Such a token does not render an existing contract invalid, but rather serves merely as a promise from the T3P that he has not previously resolved the contract in question and will not do so in the future.

The Resolve Subprotocol: The resolve subprotocol is analogous to the abort but can be invoked by either participant. The parties will request resolution of a contract from the T3P if they do not receive the secret commitment nonce of the other party within a reasonable amount of time. A resolution request includes both messages from the first stage of the exchange subprotocol, me_1 and me_2. If the T3P has already issued an abort token for the contract in question (indicated by $aborted(me_1)$), he replies in kind with an abort token. Otherwise, he issues a replacement contract and indicates in his database that he has resolved the contract.

A Unified View. The key idea behind our model is to regard ASW's subprotocols not in isolation, but rather as *one* protocol. This view is implicit in the construction of the protocol and accordingly also in the models of the protocol built by [46, 124]. We explicitly exploit this view to reason about properties of the protocol. More specifically, we consider the abort and resolve subprotocols to be *part* of the main exchange protocol. The originator role, for instance, looks then as shown in Figure 17.5 (the responder role is similar), where *timeout* represents the event that the agent playing O did not receive a reply to his last message within a reasonable amount of time. We avoid specifying the concrete amount of time after which the timeout shall occur: it may be just a few seconds or a full hour—important for the security of the protocol is only that there *is* such a timeout, so the agent will not wait for an answer forever. Figure 17.6 illustrates the internal states of an agent playing the originator role: after sending his initial message, he is in the state in which he waits for a reply until the timeout. If the timeout occurs, then he tries to abort the protocol and thus waits for the answer of the trusted third party (which can be either a replacement contract or the signed abort token). Otherwise (if he receives a reply in time), he carries on with the regular protocol execution and sends his nonce, arriving in a state similar to

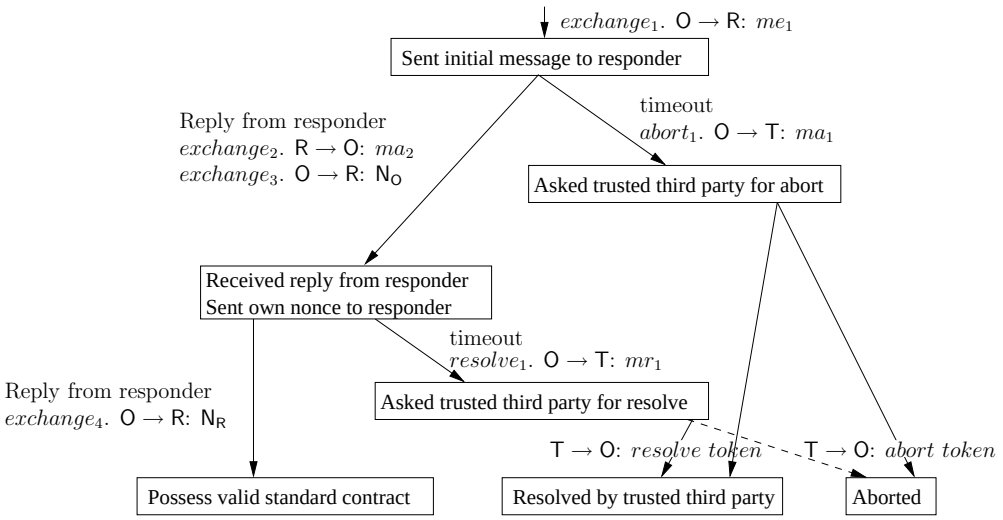


Figure 17.6: A state transition view of the originator role. The dashed line represents a transition that can never occur if the trusted server is honest, as we will show below as part of our analysis.

the one he was in after sending the first message: either there is a reply within the allotted time or he contacts the trusted third party.

This model, though abstract, is thus a faithful representation of a real implementation of the protocol, as agents indeed protect themselves with such internal timeouts. Note that this is related to the possibility of abuse in this contract signing protocol: when the originator has made the first step, the responder has the freedom to either accept the contract by sending the second message, or to reject it by ignoring it. In particular, a dishonest responder could abuse the originator-signed part of the contract in negotiations with other agents (for instance, by soliciting more advantageous contracts from competitors). Note that, unlike for instance the similar GJM [79] protocol, the ASW protocol has no means to prevent this abuse by special cryptographic primitives such as private contract signatures. Thus, the timeout is the only way to narrow the window of the originator's vulnerability to abuse attacks as argued in [47].

Although we assume that the intruder can control the entire network according to the Dolev-Yao model, the protocol requires that he cannot block messages between an honest agent and the T3P forever. One could intuitively imagine this situation as follows: all network connections could crash, but an honest agent can still transmit the necessary messages over other media (e.g. ordinary mail) to the T3P and this process cannot take forever.

One could say that we thus have two kinds of fairness assumptions: first, an honest agent will not wait forever for an answer from the other party, and second, the "emergency" protocols with the T3P will eventually succeed. Looking once again at Figure 17.6, we can interpret this combination of fairness constraints as the guarantee that an honest agent playing the originator role (and a similar guarantee holds for the responder role) will not stay forever in any of the *intermediate states* (the states of the figure with an outgoing arrow), but will eventually reach one of the three *final states*, (i) where he received the responder's nonce[3] and thus now possesses a valid standard contract, (ii) where he received a valid replacement contract from the trusted third party, or (iii) where he received an abort token.

Thus, the two fairness constraints (timeout and guaranteed reply from the T3P) are sufficient to conclude that every honest agent playing either the originator or the responder role will eventually end up with either a valid (standard or replacement) contract or an abort token. Roughly speaking, if the agent receives a valid contract, then his interests are ensured, but if he receives an abort token, then it remains to show that nobody else can obtain a valid contract.

[3]If the responder sends a secret commitment that, when hashed, does not correspond with the public commitment, then this is treated as if he had not sent any message at all (which will probably result in a timeout).

This gives us a fresh view on the protocol, as with a simple meta-argumentation we can now go from a model with fairness constraints to a state-reachability property in an infinite state transition system without fairness constraints.

The idea is essentially that we need only to check that if an honest agent reaches his final state of the protocol execution, then the guarantees he should obtain through the protocol are indeed satisfied. In other words, we do not need to consider the guarantees of agents in their intermediate states, since they will eventually reach their final state and we thus spare ourselves any considerations of the form "if the agent can eventually reach a certain state" in the properties we check.

The encoding of the objectives as safety properties is the basis for the deployment of automatic and semi-automatic methods for infinite-state analysis. Also in finite-state analysis, the restriction to safety properties is often essential, e.g. [124] use a similar argumentation that checking the protocol with fairness conditions can be reduced to checking properties of "terminal states" of agents.

Encoding Fair Exchange. We now contrast two models: on the one hand, the model with the fairness constraints described above (i.e. that the agent will eventually get the timeout and the reply from the trusted third party), and on the other hand a model without fairness constraints.

In the model without fairness, the state transitions of the honest agents as shown in Figure 17.6 are interpreted as follows: there is no timeout and no guaranteed replies, thus an agent can remain in any intermediate state forever. An agent's local state transition system is thus non-deterministic, as in the states where an honest agent waits for the reply of the other party, he can at any time (i.e. without timeout) begin the abort or resolve protocol, as appropriate.

It follows immediately that, if there is a violation of a safety property in the model with fairness, then there is also a violation in the model without fairness. This shows that our approach is sound in the sense that if we can prove properties in the model without fairness constraints, then they must also hold in the model with fairness constraints. The challenge is to find appropriate safety properties that indeed hold without the fairness constraints and that imply the safety and liveness properties that we wish to check.

The fair exchange objective can now be decomposed into the following two:

1. If an honest agent receives an abort token, then nobody (except the trusted third party) can ever obtain a valid standard or replacement contract.

2. If an agent A (who is not necessarily honest) has obtained a valid standard or replacement contract signed by an honest agent B, then B also possess a valid contract or can obtain one from the trusted third party.

Objective (1.) is the main objective of our analysis, as it reflects the basic guarantee linked with the abort token. The nice aspect of this objective is that it refers to the final state of an honest agent (which will not subsequently change), not to an intermediate state. It is thus possible to check in the model without fairness that in all states where an agent has reached a final state with an abort token, nobody except the T3P can generate a valid standard or replacement contract matching that abort token. The inability to generate these messages can be expressed in standard protocol analysis approaches by secrecy properties. We can thus reduce the main objective of the protocol to a standard property in protocol analysis (though there is a technical difficulty in the direct application of tools as we will discuss below).

Objective (2.) is a consequence of objective (1.): if an agent A possesses a valid contract signed by an honest agent B, then (due to our model of timeouts) B will also eventually reach either an abort token or a valid contract, and by (1.), if he gets an abort token, then A cannot possess a valid contract, which is a contradiction. Thus B will eventually obtain a valid contract.

Note that the circumstance in which the intruder or a dishonest agent playing the role of the originator can obtain both a valid contract and an abort token (by performing a normal run with an honest agent and asking the T3P for an abort) is not a violation of the objectives above: the

1.	$U \to H:$	$U, \exp(g, a)$
2.	$H \to U:$	$s, \exp(g, x) + \exp(g, b)$
		$K := h(\exp(g, a \cdot b) * \exp(g, b \cdot x))$
		$M := h(s, \exp(g, a), \exp(g, x) + \exp(g, b), K)$
3.	$U \to H:$	M
4.	$H \to U:$	$h(\exp(g, a), M, K)$

Figure 17.7: A&B notion of the SRP protocol.

abort token only guarantees that the T3P has never and will never resolve this contract but does not render an existing contract invalid.

Results. OFMC discovers several authentication problems that were already identified in [124], e.g. the protocol does not provide strong authentication, as it has no explicit protection against replay. However, we think one should assume an implicit replay-protection as part of the contract.

OFMC has further discovered a subtle attack that had not been reported before. The idea is that a malicious initiator first runs a standard contract signing, then performs an abort with the T3P (unbeknownst to the responder) and then sends the first message again. (One may think of a "social engineering attack" where the malicious initiator tells the responder that something on his side went wrong and asks for running the contract signing protocol again with the same contract.) After sending again the first message, the malicious initiator does not communicate anymore, so the responder will eventually ask the T3P for resolve, which he does not get as the contract is marked as aborted there.

Though this is not a serious vulnerability (the responder has a valid contract from the first exchange), it shows the subtleties of the exchange and forced us to slightly change the goal to reflect contracts with a different responder nonce, but identical contractual text and initiator nonce. With this modified goal, we discovered no further attacks. For practice, one may conclude from this attack that an implementation may not accept the exact same contractual text a second time, but rather require a unique transaction number or similar means in every session.

17.3 Secure Remote Passwords

The Secure Remote Passwords (SRP) Protocol [131, 132] is a challenging example for algebraic intruder deductions: It uses modular addition, multiplication and exponentiation, and in a model that does not take several algebraic properties of these operations into account, SRP is not even executable.

For this reason, we have used the SRP protocol as an example for testing our implementation of algebraic intruder deduction. In particular, we have checked that each agent (including the intruder) can execute all steps of the protocol. Such an executability test yields a greater test coverage than testing whether a known flaw can be found, if the flaw is based only on a single aspect of the algebraic properties.

As part of the AVISPA project [7], SRP was modeled in a drastically simplified version, basically reducing it to a Diffie-Hellman key-exchange. The implementation of the algebraic intruder deductions allows us to analyze SRP without simplifications. However, one aspect of SRP is currently not supported by OFMC. As the name suggests, SRP is designed to be secure even if passwords are guessable. Since the guessing intruder is not yet implemented in OFMC, we can only check the protocol for the case of unguessable passwords.

The protocol, displayed in Figure 17.7, works as follows. A user U wants to login at a host H using a shared password p. The host additionally has, in his password file, a random value for each user, called the salt s (while p may be guessable in reality, s is random). The salt is sent in clear-text during the authentication process. User and host build a hash value, called the *verifier*,

$x_1 + x_2$	$\approx$	$x_2 + x_1$	$x + (-x)$	$\approx$	0
$x_1 + (x_2 + x_3)$	$\approx$	$(x_1 + x_2) + x_3$	$x + 0$	$\approx$	x
$x_1 \cdot x_2$	$\approx$	$x_2 \cdot x_1$	-0	$\approx$	0
$x_1 \cdot (x_2 \cdot x_3)$	$\approx$	$(x_1 \cdot x_2) \cdot x_3$	$-(-x)$	$\approx$	x
$-(x_1 + x_2)$	$\approx$	$(-x_1) + (-x_2)$			
$\mathsf{exp}(\mathsf{exp}(x_1, x_2), x_3)$	$\approx$	$\mathsf{exp}(x_1, x_2 \cdot x_3)$			
Finite theory			Cancelation theory		

Figure 17.8: The algebraic theory used for the analysis of the SRP protocol.

from salt and password:

$$\mathsf{x} = \mathsf{h}(\mathsf{s}, \mathsf{h}(\mathsf{U}, \mathsf{p})) \ .$$

This verifier is then linked with a Diffie-Hellman key-exchange (with fresh numbers a and b) to provide authentication based on the password without opening the door for guessing attacks. Note that all additions, multiplications and exponentiations are modulo a commonly known number n.

We give here only a highlight of the specification, namely how the user receives message 2 and constructs the reply message 3. The user obtains the value $\mathsf{exp}(\mathsf{g}, \mathsf{b})$ from the second part of message 2 by *subtracting* the verifier $\mathsf{exp}(\mathsf{g}, \mathsf{x})$, and will continue the protocol with whatever results he obtains from it. It is crucial that the agent does not have a way to recognize whether he has obtained a "valid" half-key $\mathsf{exp}(\mathsf{g}, \mathsf{b})$, otherwise one would open the door for guessing attacks if the password is guessable.

For checking SRP with OFMC, we use the algebraic theory displayed in Figure 17.8 which is sufficient for modeling the protocol as described below. Note that we have not included distributivity in this theory, i.e. the property that $x_1 \cdot (x_2 + x_3) \approx (x_1 \cdot x_2) + (x_1 \cdot x_3)$. In principle this is possible, as the finite part of the theory is still finite when adding distributivity. However, distributivity leads to non-confluence of the modular rewriting relation (which cannot be repaired by addition of further equations). Another problem is that OFMC currently does not support finite theories where an equation contains *non-linear terms*, i.e. where a variable occurs several times on one side of an equation. As part of future work, it is planned to allow non-linear terms in the equations of finite theories.

Using this theory, we can model how a user receives the message 2 from the host and forms the new shared key K out of it. Namely, the user accepts any value in place of the sum $\mathsf{exp}(\mathsf{g}, \mathsf{x}) + \mathsf{exp}(\mathsf{g}, \mathsf{b})$. Let us denote this arbitrary value with the variable GXGB. The user constructs the Diffie-Hellman half-key by subtracting the verifier $\mathsf{exp}(\mathsf{g}, \mathsf{x})$, i.e. building the term

$$\mathsf{GB} = \mathsf{GXGB} + (-(\mathsf{exp}(\mathsf{g}, \mathsf{x}))) \ .$$

Observe that the construction of GB is independent of the form of GXGB (which the agent cannot check), and similarly, he will construct the new key based on the term GXGB as

$$\mathsf{K} = \mathsf{h}(\mathsf{exp}(\mathsf{GB}, \mathsf{a}) \cdot \mathsf{exp}(\mathsf{GB}, \mathsf{x})) \ .$$

Observe that $\mathsf{K} \approx \mathsf{h}(\mathsf{exp}(\mathsf{g}, \mathsf{a} \cdot \mathsf{b}) \cdot \mathsf{exp}(\mathsf{g}, \mathsf{b} \cdot \mathsf{x}))$ if $\mathsf{GXGB} \approx \mathsf{exp}(\mathsf{g}, \mathsf{x}) + \mathsf{exp}(\mathsf{g}, \mathsf{b})$. In other words, if the agent receives a message of the form that the protocol describes, then also the outgoing message is in accordance with the protocol. However, the intruder can send any term in place of GXGB and the agent will construct the key as above (which reflects what happens in reality). It is now a matter of the protocol analysis whether the intruder can construct a term GXGB such that the resulting key K has a value that leads to an attack. For instance, if the intruder can also construct the resulting key K, then he can impersonate the host.

In the experiments on the SRP protocol we did not find any attacks. Also, we have established that OFMC indeed recognizes that the protocol is executable within the algebraic theory of Figure 17.8, in particular the model of both honest agents and intruder are sufficient to execute the protocol.

In further experiments with algebraic properties, we have also considered examples of known attacks that require algebraic properties such as in [127]. Also we have considered other operators like commitment schemes and modeled them using algebraic properties. Since OFMC is not specialized to a particular theory, we were able to experiment with several algebraic theories; namely, we can choose any theory between the following two extremes:

- Modeling as many properties of the real operators as possible.
- Modeling the operators as abstract as possible such that the protocol is still executable.

As an example for the second extreme, consider modeling Diffie-Hellman by *two* different exponentiation operators exp_1 and exp_2, the first for constructing half-keys, and the second for constructing the full keys. We then use the following property for exponentiation:

$$\mathsf{exp}_2(\mathsf{exp}_1(x_1, x_2), x_3) \approx \mathsf{exp}_2(\mathsf{exp}_1(x_1, x_3), x_2) \ .$$

Observe that this way of modeling Diffie-Hellman is much more restrictive than in models that do not distinguish "two exponentiation operations". While this restriction potentially excludes attacks, we are getting close to a black-box interpretation of algebraic operators, which can be helpful for an efficient analysis of protocols. The flexibility of custom algebraic theories allows us to perform the analysis with different focuses, namely on the properties of the operators or on other aspects like a larger number of sessions.

Chapter 18

Protocol Libraries

During the development of OFMC, we have used two large libraries of protocols as a test-suite. After every change to the implementation, this test-suite was run and the results compared to the results of the previous version. The results consisted not only of the question whether an attack was found or not, but also of the precise attack trace reported (for flawed protocols), and the number of states visited during the analysis, as well as the total analysis time of the library. In this way, we have discovered and fixed several subtle implementation mistakes.

Moreover, the broad scope of protocols covered by the test-suite allowed us to easily assess the improvement on the overall performance of the tool resulting from changes in the implementation. Although we refrained from using heuristics that are tailored towards particular examples, several improvements result in an additional overhead for data-structures and checks, so that they might even decrease the overall performance on the test-suite, while being helpful only on a few examples.

18.1 The Clark/Jacob Library

The initial test-suite we used was composed from protocols of the Clark/Jacob Library [54] which consists of 51 authentication protocols, most of them relative small protocols of the 1980s and early 1990s. As many of these protocols were standard examples in formal analysis, for 29 of these protocols, attacks were already known at the beginning of OFMC's development [73]. We were thus able to check that OFMC can find an attack to each of these flawed protocols. For some protocols, several different attacks are reported in the literature; we have thus considered it sufficient for the test that OFMC finds *some* attack on such a protocol.

OFMC indeed satisfies this criterion, and also discovered a new attack on the Yahalom protocol we have presented in Figure 2.2. As the performance times in Table 18.1 show, OFMC is a state-of-the-art tool: for each of the flawed protocols, a flaw is found in under 4 seconds and the total analysis of all flawed protocols takes less than one minute of CPU time. The time displayed for each attack is the one measured for the minimum number of protocol sessions such that the attack can be performed (which is two sessions in most cases).[1]

Note that the analysis can optionally be performed in a typed model as described in Subsection 4.4.4, which may result in different (non-type-flaw) attacks. When this is the case, we report in Table 18.1 both attacks found. In all other cases, the times are obtained using the untyped model. Also note that the table contains four variants of protocols in the library, marked with a "*", that we have additionally analyzed.

To our knowledge, NPA [108], CL-AtSe [128], and Scyther [64] are the only other automated analysis tools that have the same coverage as OFMC for the Clark/Jacob library, in the sense that they can find an attack on all protocols for which a flaw is known. One reason is that several protocols are vulnerable to subtle type-flaw attacks while they are safe a the typed model such

[1] Experiments where performed on a 2,4 GHz Pentium-4 PC (with 512 MB RAM, but OFMC is not memory intensive).

Protocol Name	**Attack**	**Time (s)**
ISO symm. key 1-pass unilateral auth.	Replay	0.0
ISO symm. key 2-pass mutual auth.	Replay	0.0
Andrew Secure RPC prot.	Type flaw	0.0
	Replay	0.1
ISO CCF 1-pass unilateral auth.	Replay	0.0
ISO CCF 2-pass mutual auth.	Replay	0.0
Needham-Schroeder Conventional Key	Replay	0.3
Denning-Sacco (symmetric)	Type flaw	0.0
Otway-Rees	Type flaw	0.0
Wide Mouthed Frog	Parallel-session	0.0
Yahalom	Type flaw	0.0
Woo-Lam Π_1	Type flaw	0.0
Woo-Lam Π_2	Type flaw	0.0
Woo-Lam Π_3	Type flaw	0.0
Woo-Lam Π	Parallel-session	0.2
Woo-Lam Mutual auth.	Parallel-session	0.3
Needham-Schroeder Signature prot.	Man-in-the-Middle	0.1
* Neuman Stubblebine initial part	Type flaw	0.0
* Neuman Stubblebine rep. part	Replay	0.0
Neuman Stubblebine (complete)	Type flaw	0.0
Kehne Langendorfer Schoenw. (rep. part)	Parallel-session	0.2
Kao Chow rep. auth., 1	Replay	0.5
Kao Chow rep. auth., 2	Replay	0.5
Kao Chow rep. auth., 3	Replay	0.5
ISO public-key 1-pass unilateral auth.	Replay	0.0
ISO public-key 2-pass unilateral auth.	Replay	0.0
* Needham-Schroeder Public-Key NSPK	Man-in-the-Middle	0.0
NSPK with key server	Man-in-the-Middle	1.1
* NSPK with Lowe's fix	Type flaw	0.0
SPLICE/AS auth. prot.	Replay	4.0
Hwang and Chen's modified SPLICE	Man-in-the-Middle	0.0
Denning Sacco Key Distr. with Public Key	Man-in-the-Middle	0.5
CCITT X.509	Type flaw	0.1
Shamir Rivest Adelman Three Pass prot.	Type flaw	0.0
Encrypted Key Exchange	Parallel-session	0.1
Davis Swick Private Key Certificates	Type flaw	0.1
(DSPKC), prot. 1	Replay	1.2
DSPKC, prot. 2	Type flaw	0.2
	Replay	0.9
DSPKC, prot. 3	Replay	0.0
DSPKC, prot. 4	Replay	0.0

Table 18.1: Performance of OFMC over the flawed protocols of the Clark/Jacob library.

Protocol	CL-AtSe	OFMC	SATMC	TA4SP
ISO1	0.02	0.02	0.04	-
ISO3	0.03	0.03	0.39	-
EKE	0.03	0.10	0.09	2.86
IKEv2-DS	0.29	2.38	-	-
LPD-MSR	0.02	0.02	0.06	0.61
PBK	0.01	0.34	0.25	-
PBK-fix	0.03	0.14	0.09	-
ASW-abort	0.20	1.98	65.75	-
SET-purchase	23.66	1.24	TO	-
h.530	TO	0.70	-	-

Table 18.2: Performance on the Flawed Protocols of the AVISPA Library (Typed Model).

as the one described in Subsection 4.4.4; since many methods are based on a typed model or are limited in finding type flaw attacks, they cannot obtain full coverage in this regard.

18.2 AVISPA-Library

The AVISPA library [13] is a suite of modern industrial-scale protocols that are of practical relevance in todays Internet communication (unlike most protocols from the Clark/Jacob library). The protocols of the suite were selected and formalized during the AVISPA project [15]. The formalization language is the already mentioned HLPSL [48], which is automatically translated by the HLPSL2IF compiler to the Intermediate Format that we have used as a basis throughout this thesis. IF is the input language also to the analysis tools CL-AtSe [128], SATMC [10, 11], and TA4SP [36, 114], which we describe below. This gives us the basis for a fair comparison of tools, as they are all analyzing the same problem, rather than using slightly different models, formalizations, and assumptions.[2] Note, however, that TA4SP is based on over-approximations similar to the ones described in Part IV and can therefore verify secrecy properties for protocols without a bound on the number sessions, i.e. it aims at a much more difficult problem that cannot be handled by the other tools.

The performance (in seconds CPU time) on the different problems shown in Table 18.2 and Table 18.3 were taken from the final project assessment of the AVISPA tool [16].[3] We have splitted the table into flawed protocols (where an attack was found) and safe protocols (where no attack was found).

In the tables, "TO" denotes a time-out, i.e. the analysis was not finished within one hour of CPU time. Moreover, "-" denotes that a protocol could not be handled by a back-end. For instance, there are several protocols that involve Diffie-Hellman, and only CL-AtSe and OFMC have support for algebraic properties. Observe that OFMC is the only of the four back-ends that has *total coverage*, i.e. that can handle each of the analysis problems, and, moreover, it is the only back-end that could find all attacks.

Despite the fact that we have compared only four tools here, the results give us a representative picture of the state-of-the-art in automated protocol analysis. In particular, OFMC is among the tools with the best performance and coverage, however other tools exist that have complementary strengths, e.g. they scale better to a larger number of sessions or even allow for the verification of an unbounded number of sessions, while having limitations in the class of protocols they can

[2]It is possible that due to bugs, one tool might analyze a slightly different problem than another, but the comparison is as fair as possible.

[3]Results are obtained with a resource limit of one hour CPU time and 1GB memory, on a Pentium IV 2.4GHz. We refer to each protocol with the names used in the AVISPA library.

Protocol	**CL-AtSe**	**OFMC**	**SATMC**	**TA4SP**
UMTS_AKA	0.01	0.03	0.01	0.56
ISO2	0.02	0.07	0.63	-
ISO4	0.03	0.38	208.31	-
CHAPv2	0.02	0.18	0.10	16.29
SRP	0.02	0.07	-	-
EKE2	0.03	0.05	-	-
SPEKE	0.07	1.49	-	-
IKEv2-CHILD	0.07	0.51	-	-
IKEv2-DSx	3.74	17.28	-	-
IKEv2-MAC	0.05	3.01	-	-
IKEv2-MACx	5.27	15.94	-	-
TLS	0.05	0.29	1018.28	TO
LPD-IMSR	0.04	0.04	0.10	3.25
Kerb-basic	0.07	0.61	6.24	TO
Kerb-Cross-Realm	0.52	2.22	5.70	-
Kerb-Ticket-Cache	0.08	0.60	28.90	-
Kerb-PKINIT	0.06	0.47	27.21	-
Kerb-Forwardable	0.16	7.12	TO	-
Kerb-preauth	0.12	0.39	20.54	-
CRAM-MD5	0.04	0.23	0.17	0.97
PBK-fix-weak-auth	0.49	3.47	0.33	-
hip	0.09	0.23	-	-
DHCP-delayed-auth	0.02	0.06	0.12	6.84
lipkey-spkm-knw-init.	0.06	0.17	-	-
lipkey-spkm-unknw-init.	0.12	4.72	-	-
TSIG	0.05	0.19	0.38	-
ASW	0.10	0.35	TO	-
FairZG	0.37	8.76	0.28	-
SET-p.-hon.-payment-gw	0.61	0.83	TO	-
AAAMobileIP	0.03	0.14	0.11	754.11
h.530-fix	TO	1391.66	-	-
Simple	100.17	78.50	0.50	-
CTP-non_predictive-fix	0.06	0.23	TO	-
geopriv	0.04	0.25	0.08	-
pervasive	47.67	30.52	4.00	TO
two_pseudonyms	0.06	0.30	0.07	-
QoS-NSLP	32.16	16.01	0.21	-
sip	0.05	1.86	810.01	-

Table 18.3: Performance on the Unflawed Protocols of the AVISPA Library (Typed Model).

support. To obtain a more precise picture, we now consider the methods of the other back-ends in more detail.

18.2.1 The Methods of the Back-Ends

CL-AtSe (Constraint Logic Attack Searcher [128]) is the back-end that is the most similar to OFMC. It is also based on the lazy intruder technique, but does not incorporate constraint differentiation. However, it uses a pre-processing phase of IF rules that OFMC does not have. This pre-processing is similar to step compression, trying to statically identify rules that can be composed without loss of attacks. For instance, usually the first transition of the initiator role of a protocol requires only the presence of a respective state-fact (which is usually part of the initial state); in most cases, the respective rule can therefore be "applied" to the initial state, adding the initial message of the sessions that come from honest agents and updating the respective states. Such a pre-processing phase has a good effect on the performance and explains why CL-AtSe is in many cases faster than OFMC. For some protocols, however, OFMC performs better. The reason is probably that the constraint differentiation technique can cut down the search spaces considerably, and the effect of constraint differentiation is the stronger the more sessions and protocol steps there are, as explained in Section 9.4. This hypothesis is also supported by further experiments [9].

SATMC (SAT Model-Checker [10, 11]) is based on a completely different method, namely encoding the problem (for bounded lengths of traces, which is possible for a bounded number of sessions) into a Boolean formula (that is satisfiable iff there is an attack) and feeding this formula into modern SAT-checkers. SATMC has a drawback concerning protocols with complex messages, as they have to be encoded into boolean formulae. On the other hand, SATMC is less sensitive to the explosion of interleavings that results for a larger number of sessions, as recent experiments by the developers suggest [9].

TA4SP (Tree Automata for Security Protocols [36, 114]) is based on the use of tree-automata to represent an over-approximation of the intruder knowledge that can be reached, similar to the models described in the previous part. For the flawless protocols, it proves a much stronger result than the other other tools, namely that the protocol is safe for an unbounded number of sessions. It has however other limitations in that it cannot check authentication goals (only secrecy) and cannot handle other more complex features like negative conditions and sets.

18.2.2 The Flaws

We conclude this section with an overview of the flaws that were detected.

IKEv2 The attack to IKEv2-DS (IKE version 2, variant with Digital Signatures) suffers from a "flaw" that has previously been reported on the Station-2-Station protocol [100] and the first version of IKE [109]. As both IKE and IKEv2 are "descendants" of the Station-2-Station protocol, we do not consider this weakness as new either. The point of the "attack" is the following. In order to protect the identity of the users, a Diffie-Hellman key is negotiated first without authentication, and then the agent names are only transmitted in encrypted messages; while this does not offer perfect protection, an intruder needs to be active in such an exchange to find out the name of an agent. However, in several variants of these protocols it is possible that an agent reaches the end of the protocol execution in the belief to share a key with another honest agent who actually does not know this key. This is a violation of the authentication goals. However, in all these situations, nobody else knows the key, and thus there cannot be any payload transmitted using this key. One could say that there is *implicit key-authentication* as soon as the key is successfully used for the first time. In order to formally verify that there is no problem with these protocols, we have thus extended these protocols by a final step where the receiver of the last message of

the key-exchange sends a fresh nonce (representing some arbitrary payload) using the new key; adapting the authentication goal to this new last message of the protocol, we were able to verify that there are no attacks (examples IKEv2-DSx and IKEv2-MACx Table 18.3).

SET The Secure Electronic Transactions (SET) Protocol Suite is designed for securing E-commerce. The key feature is to hide the customer's credit card details from the merchant, and the customer's purchase details from the payment gateway. There is an attack where a dishonest payment gateway forwards payment authorization requests to another payment gateway. This is due to the fact that the part of the message signed by the card-holder (as well as the one signed by the merchant) does not contain the name of the desired payment gateway. This weakness of the protocol was already mentioned in the analysis of the SET protocol by Bella, Massacci, and Paulson using the interactive theorem prover Isabelle [30]. They argue that the attack is not very interesting as a dishonest payment gateway "has more interesting crimes to commit", however we believe that this vulnerability is not uncritical as it may lead to the situation that two payment gateways charge the account of the card-holder and both posses messages that seem to prove that the card-holder authorized the transaction. Like [30], we suggest to include the name of the desired payment gateway into the messages to fix this problem.

Others Attacks The attacks to ISO1 (ISO Public-Key One-Pass Unilateral Authentication Protocol), ISO3 (ISO Public-Key Two-Pass Mutual Authentication Protocol), and EKE (Encrypted Key Exchange) were also present in the Clark/Jacob library and had already been reported by [73]. The attacks to ASW and H.530 have been described above already. The LPD-MSR (Low-Powered Devices, Modulo Square Root) cannot provide authentication unless the intruder is passive, as there is no a priori relationship between participants, thus there is trivially an attack. PBK (Purpose-Built Keys) so far exists only as a framework without a detailed specification of the messages. The attacks we discovered demonstrate two possible mistakes that might be made in a more concrete design of the protocol: firstly, unless tagging of signed messages is used, the initiator can be exploited as a *signing oracle*, violating authentication, and secondly, unless a hash-value of the payload message is included in the last exchange, the protocol is vulnerable to replay. PBK also refers to a variant of authentication goals, called sender invariance, which we have formalized in [85].

Our results therefore indicate that our goal to provide a flexible, practically feasible tool for protocol analysis is fully achieved. OFMC has detected a number of vulnerabilities in complex protocols like H.530. In the case of H.530 this lead to a modification of the protocol by the designer Siemens [93]. The performance of OFMC allows protocol designers to experiment with different protocol designs, goals, and algebraic properties, as the answers are within seconds for most problems. Finally, as part of the AVISPA tool, OFMC is being deployed by dozens of researchers and protocol designers world-wide for the validation of their protocols.

Chapter 19

Conclusions and Related Work

This thesis covers a broad range of aspects in the formal analysis of security protocols, namely the design of formal models and description languages for security protocols, the development of efficient techniques for the automated analysis, and the realization and deployment of a tool that is practically feasible for the analysis of modern, relevant security protocols. We now summarize our work and contributions for each part and discuss related and future work.

19.1 Modeling

The OFMC tool employs the set-rewriting based Intermediate Format (IF) as the formal specification language for security protocols and their goals. This formalism is similar to the CAPSL Intermediate Language CIL [70], which is based on multi-set rewriting. Concerning the transition systems that can be described in this way, IF is more expressive than CIL thanks to the inclusion of negative facts and inequalities. Nonetheless, as rewriting in itself is a powerful concept that can describe, e.g., Turing machines or combinatory logic [65], it is, in principle, possible to simulate all IF-expressible transition systems in CIL as well. However, such a simulation is often not suitable for automated analysis tools.

There are several other formalisms that can be used to describe security protocols in an unambiguous way. We give a brief overview of the most relevant examples.

A&B For the end-user of analysis tools, a low-level formalism like IF is not very suitable. As papers and text-books often describe protocols in a high-level way using A&B notation, it has become the basis for many formal description languages like Casper [102], CAPSL [70] and Casrul [94]. While being very intuitive to use, these languages suffer from a limited expressiveness: the relationship between sessions (like a data-base shared over all sessions of a participant) and the control-flow (e.g. presence of several subprotocols, conditional execution of protocol parts, loops, etc.) require extensions that, to our knowledge, have never been incorporated into any of the formal description languages based on A&B. Since these languages are very abstract, it is often desirable not to directly take them as input languages for analysis tools, but to first translate the descriptions into a low-level formalism like IF. Factoring such a translation step out of the tools, allows for an easier integration of a variety of back-ends. To use a high-level language for the user and a low-level language for the analysis tools is the basic idea behind the architecture of CAPSL/CIL [70], as well as the AVISS tool and its successor AVISPA tool [7, 8].

Strand Spaces Strand spaces [76] can be regarded as a compromise between high-level and low-level languages. On the one hand, the formalism shares with A&B notation both the simplicity and the limited expressiveness.[1] On the other hand, strand spaces are already close to the view of

[1] Strand Spaces are somewhat more limited as A&B notation as they do not allow, for instance, to model a protocol where one participant receives an undecipherable message and can later check and use the contents after

honest agents as processes, namely sender and receiver of a message are different (and in general unrelated) strands, and, moreover, they already reflect the way an agent parses a message (i.e. what subterms can be checked to have a certain form). For this reason, we have used in Part IV a similar notation (introduced in Section 13.1) as a basis from which several protocol descriptions in different models can be generated. We have thereby avoided the translation steps necessary from A&B to low-level formalisms, and also the formalism is abstract enough to be used as the generic description that is independent from the different models.

Process Calculi The fact that honest agents are seen as (usually independent) processes suggests the use of process calculi to describe protocols. This formalism is used in a variety of works, e.g. [33, 34, 81, 102]. There are several similarities with rewriting-based approaches, however there is a major conceptual difference: properties of protocols are modeled by indistinguishability of traces (after hiding several events). This has lead to several interesting concepts that can be helpful, for instance, in modeling anonymity [4] or guessing attacks [2].

Message Trace Models The approach of Paulson [115] to model protocols as inductive sets of messages may at the first look be related to strand spaces, but in fact this purely "behavioral" specification has very different properties from all models considered so far. As we have shown in Part IV, there is a large difference between the models: the lack of an explicit notion of state leads to an over-approximation in the control structure. While this leads to a subtle problem in authentication goals, we have formally shown that with additional goal-related events, this control abstraction is sound in the sense that the protocol without abstraction is safe, if the abstracted version is. The control abstraction can be easily combined with data abstraction, and this combination has proved to be very successful in the verification of protocols for an unbounded number of sessions, e.g. [33, 36, 41, 42, 53, 80, 130].

Temporal Logic Finally, several formalisms have been developed on the basis of temporal logics, e.g. the AVISPA High-Level Protocol Specification Language (HLPSL) [48] is based on Lamport's Temporal Logic of Actions (TLA) [98]. It is arguable whether the focus of such a formalism on the temporal structure is actually the best one for protocol analysis. As our formal analysis of models based on control abstractions in Part IV has shown, the temporal aspects of security protocols are relatively simple and can often be abstracted away. At the same time, several aspects that are essential to security protocol analysis need to be added as extra-logical concepts. For instance, during the development of HLPSL, we have discovered that we cannot appropriately express within TLA the interpretation of terms in the quotient algebra of terms with the algebraic properties (i.e. that two different terms denote different values unless their equality is a consequence of the algebraic theory). Therefore we can express the semantics of HLPSL in TLA only with a reinterpretation of TLA and prevents us from using model-checkers for TLA off-the-shelve.

19.1.1 Contributions

The contributions in the modeling part are twofold. First, we have formally defined a formalism for modeling security protocols that has many advantages over other approaches. It is based on a small number of concepts, namely set-rewriting with the extensions of negative conditions, generation of fresh constants, and a special predicate for intruder knowledge that is related to the employed intruder model. It is therefore both sufficient to model a large number of protocols and straightforward to work with; in particular, it is suitable for a large variety of analysis tools.

Second, we have described in detail how to model security protocols; we have both discussed modeling issues in general (independent of any particular formalism) and given a formalization

receiving the encryption key.

in IF. We have thereby highlighted many subtle aspects of modeling the honest agents, the intruder, the communication medium, and the goals. We have also shown that the step compression technique is a correct optimization.

19.2 Automated Methods

We now review some of the most influential automated approaches and tools that are currently available, and compare them with ours. Note that a direct comparison of running times as in the previous chapter is difficult, especially as the tools are based on different models and assumptions.

Explicit State Model-Checking We begin our comparison with the standard model-checking algorithms, where we use the term model-checking in a broad sense, namely for algorithms that explore a (usually finite) state-space. As a prominent example, we compare our approach with Casper [73, 102], which translates protocols from an A&B-style specification language to descriptions in the process algebra CSP. The approach uses finite-state model-checking with FDR2, a general purpose model-checker that is not specialized to protocol analysis. Casper/FDR2 successfully discovered flaws in a wide range of protocols: among the protocols of the Clark/Jacob library, it found attacks on 20 protocols previously known to be insecure, as well as attacks on 10 other protocols originally reported as secure. The approach does not include a special representation of the intruder, like the lazy intruder technique, and due to the restriction to finite state-spaces, it is thus necessary to limit the depth of terms that an intruder can create. Therefore, in terms of performance and the complexity of protocols that can be handled, Casper/FDR is weaker than many modern analysis tools that employ techniques specialized to the problems of protocol analysis like the lazy intruder. Nonetheless, Lowe et al. have presented several techniques and results to overcome the limitations. First, they have argued that under certain conditions, one can restrict the depth of terms [87] and the number of sessions [103] without loosing attacks. Second, they have considered checking protocols for an infinite number of sessions in the context of abstraction-related techniques called data-independence [120].

Backward Search To overcome the restriction to finite state-spaces in standard model-checking, several approaches consider backward search [64, 108, 126]. More in detail, the search starts with an attack state and explores backwards from which states the attack states are reachable. If the search reaches an initial state, an attack is found; if the search does not find "new" predecessor states, the protocol is safe. The advantage of these methods is that they can in some cases prove the correctness of a protocol without limiting either the intruder or the number of sessions. The disadvantage is that in many cases the search does not terminate without introducing such limits. Several techniques have been proposed to obtain termination, for instance characterizing languages of terms that the intruder can never obtain (e.g. the private keys of honest agents), so the search can prune states that require the intruder to obtain such a term.

Lazy Intruders Several works have used symbolic constraint-based approaches like the lazy intruder technique to efficiently address the problem of the prolific Dolev-Yao intruder [56]. The idea of a symbolic intruder model has undergone a steady evolution, becoming increasingly simpler and general. In the earliest work [91], both the technique itself and the proofs of its correctness were of substantial complexity. [6] drastically simplified the technique and its formal presentation (in particular, the proofs of correctness). [52] presented the first approach that can handle non-atomic keys (proved correct in [53]), and [121] showed that, for a bounded number of sessions, the protocol insecurity problem with non-atomic keys is NP-complete. The mentioned CL-AtSe [128] was developed out of these works. [111] independently proposed a similar generalization to non-atomic keys (although in their case the public key infrastructure is fixed) and gave simpler proofs than [6]. [60] improved the approach of [6] by increasing the expressiveness and providing a more efficient implementation.

Our contribution to the lazy intruder technique (within the free algebra) are as follows.

- We give a precise algorithmical account of the lazy intruder technique that is close to the real implementation. In contrast, most other publications have described the technique by a calculus which admits a much larger number of reductions than the real implementations actually consider. In fact, checking all reductions of such a calculus is often worse than a naïve enumeration of ground terms, defying the argument that the lazy intruder is a technique to efficiently address the intruder deduction problem. The completeness proof that we give for our algorithmic presentation of the lazy intruder technique therefore closes a significant gap between the usual way of presentation and the implementation.

- We have introduced the handling of inequalities into the reasoning of the lazy intruder. The introduction of inequalities, together with the notion of simple constraints, has a very natural interpretation: "the intruder can generate as many different terms as he likes". Handling inequalities has several applications. First, inequalities are necessary to handle negative conditions in the rules of IF. Negative conditions are crucial for a number of advanced protocols and goals, e.g. to express strong authentication in a declarative way (see Section 4.5). Second, they can be used to exclude substitutions during reduction and are therefore helpful to efficiently implement the lazy intruder. Third, the symbolic session technique is also based on the use of inequalities. Inequalities have also been considered independently by [6, 106].

- We have devised an abstract data-type for the lazy intruder that provides an interface for integrating the lazy intruder into the exploration of the transition system. The abstraction hides the technical aspects of the lazy intruder to the rest of the approach. Due to this interface, the lazy intruder can be easily integrated with the implementation of the IF transition rules. We formally showed that the integration is correct, i.e. that the symbolic transition system with the lazy intruder is equivalent to the original transition system defined by the semantics of the IF description. Moreover, we have showed how to use lazy evaluation to straightforwardly combine the advantages of early and late constraint reduction, i.e. pruning of states with unsatisfiable constraints and re-using the results from constraint reduction performed on predecessor states.

- We have introduced the technique of symbolic sessions which uses the lazy intruder to efficiently solve the problem of searching the space of "scenarios", i.e. instantiations of a given set of sessions with concrete agent names. The idea is that the choice of the scenario is given to the intruder who can decide *lazily*, i.e. driven by the constraints arising during search.

- To handle the interleaving problem that arises from a larger number of sessions, we have introduced the technique of constraint differentiation. It uses ideas from partial-order reduction to reduce the state-space without excluding (or introducing) attacks. Partial-order reduction is not directly applicable to the symbolic state-space of the lazy intruder (since different interleavings necessarily lead to different states due to the constraints), and constraint differentiation introduces information about redundancies into the constraint reduction process, namely by declaring that certain solutions are redundant unless a particular part of the intruder knowledge is used. Similar techniques were used later by [129].

Implementing these extensions of the lazy intruder has significantly improved the scope and performance of OFMC, as for instance our experiments in Section 9.4 demonstrate. Further related automated methods, namely in algebraic reasoning and abstraction based analysis, will be summarized in the following sections.

19.3 Algebraic Properties

The use of algebraic properties is a challenging problem that arises in the analysis of many modern protocols in the sense that, for instance, protocols using the Diffie-Hellman key-exchange cannot be reasonably modeled without taking algebraic properties into account (e.g. those of modular exponentiation).

In Part III, we have presented a framework that handles algebraic properties in a uniform and modular way. It is not specialized to any particular algebraic theory and thereby allows users to declare new operators and properties as part of the protocol specification. The framework is based on the use of modular rewriting to formalize a generalized equational deduction problem for the Dolev-Yao intruder, and on bounding the depth of message terms and the analysis operations of the intruder to control the complexity of the equational unification problems that arise. These bounds allow us to give general algorithms for the equational unification and intruder deduction problems. Moreover, under these bounds, our framework is also open to the integration of more efficient algorithms that are specialized to particular algebraic theories (and which usually work without such bounds), e.g. [49, 50, 112]. A contribution on the theoretical side here is that the protocol analysis problem and essential subproblems become undecidable when dropping any of the restrictions that we have introduced, namely bounding the variables of protocol descriptions and the intruder deductions.

Note that OFMC and CL-AtSe represent the only lazy-intruder based approaches with algebraic properties that have been implemented. The advantage of CL-AtSe is that, due to specialized algorithms, it is more efficient than OFMC and does not need any bounds on the depth of messages and deduction steps for algebraic reasoning. The advantage of OFMC is that the user can specify a theory, within a broad class of theories, as part of the analysis problem. This allows us to consider many properties that are not integrated in other tools. Also, as mentioned in Section 17.3 for the SRP case study, the flexibility provided by custom algebraic theories allows us to perform the analysis with different focuses, namely on the properties of the operators or on other aspects like a larger number of sessions.

The idea of providing a general approach for integrating equational properties into security protocol analysis has recently attracted considerable attention. [66] presents an approach based on standard rewriting that supports the specification of properties like the cancelation theories of our framework. However it does not allow for properties like AC, which we handle by finite theories. The approach of [51] has aims similar to ours: to provide a general framework that is open to the integration of existing algorithms. This approach, however, is based on a different idea, namely ordered rewriting, and is therefore applicable to classes of theories that are incomparable to the ones that are supported by our framework. The approaches of [2, 58, 97, 110] are the most closely related to ours as they also employ modular rewriting. They differ from our work in that they are more restrictive in terms of the kinds of modulo theories that can be considered; namely they consider a fixed modulo theory (or, similarly, assume given a unification algorithm for the modulo theory), or they require that the unification problems are finitary. These restrictions, however, allow them to work without the bounds required by our approach.

Our framework is not biased towards a particular analysis method, and thus can be used as a basis for handling algebraic equations when employing different types of formalisms or techniques for protocol analysis. We have shown that we can apply our algorithms for algebraic reasoning for building a semi-decision algorithm for the insafety of IF protocol specifications, and a decision algorithm when restricting ourselves to protocol specifications with bounded variables and sessions and restricting the intruder to bounded deductions. We have also sketched the integration of our framework for algebraic reasoning with the lazy intruder technique in Section 11.4.

Application to Modeling Off-line Guessing Based on the framework for algebraic properties, we have formalized off-line guessing in Chapter 12 as an extension and modification of the Dolev-Yao model. Our formalization is based on the notion of using maps to explicitly represent the intermediate computation steps of an intruder during guessing. We exploit the fact that with algebraic properties we can express that a decryption yields the original clear-text iff the correct key was used, and thereby defining how the intruder can identify the correct guess iff there is something in the clear-text that he can distinguish from some garbage that results out of decryption with a wrong key.

Our formalization is general and uniform in the sense that it is independent of the overall protocol model and of the details of the considered intruder model, i.e. cryptographic primitives,

algebraic properties, and intruder capabilities. Moreover, it allows us to consider multiple guesses simultaneously.

Several other approaches have similar extensions of the Dolev-Yao model with additional intruder capabilities to model a notion of off-line guessing, starting with the work of [100], which inspired [55, 61] and was extended in [104]. These first approaches are somewhat limited (for instance [61, 100] consider only single guesses) and are thus helpful to find protocol attacks, but are not fully appropriate for providing a definition of off-line guessing and verifying that a protocol is invulnerable to such guessing attacks.

The more advanced approach of [67] presents a theory of off-line guessing that overcomes such limitations and also includes an analysis of its complexity and a basis for an efficient implementation using a symbolic intruder model. Like previous approaches, it is, however, specialized to the standard Dolev-Yao intruder under a free term algebra, and it is based on involved syntactic concepts like normalized intruder derivations to ensure that the guessing extension is not "too powerful" (in the sense that the intruder could trivially derive everything guessable without restrictions). The approach of [59] is the closest to ours as it also employs cancelation equations rather than explicit decryption rules. Moreover, like our approach, it is not specialized to a particular intruder model, although it is based on an explicit formalization of protocols using the applied pi calculus of [3] where security properties are defined using a notion of indistinguishability of processes.

It is however not easy, at least in our opinion, to estimate whether the approaches of [59, 67], as well as the previous ones, indeed faithfully formalize the notion of off-line guessing, as it is not completely clear what notion they capture. In essence, these works formally define derivation systems but give little intuition about how their respective systems relate to a common-sense understanding of off-line guessing.

The major contributions of our formalization of guessing is that it describes and formalizes such a common-sense intuition of off-line guessing in a precise way and thus provides an appropriate bridge between the intuitions and a formal model of them. A detailed investigation of our intuition led us to the notion of maps on which our formalization is based. In fact, one might say that maps provide a semantical basis for our rules as they make explicit the intermediate computations of an intruder during an off-line guessing attack. Our formalization is thus not only uniform and more general, but also conceptually simpler and easier to understand than previous approaches. Note, however, that our approach is based on algebraic equations, and is thus more difficult to implement within a protocol analysis tool than the less expressive approaches. We have here placed the emphasis on the theoretical foundations underlying our model of guessing; an implementation has not yet been worked out, but is planned for future work on the tool OFMC.

19.4 Abstraction-Based Analysis

The work presented in Part IV is concerned with the questions of modeling protocols and comparing such models rather than particular techniques that are used to automatically verify protocols based on these models. However, there exists a wide variety of empirically successful analysis techniques based on the models presented in this part, e.g. [33, 36, 41, 42, 53, 80, 130].

We have formally proved equivalence and over-approximation relationships between several protocol models. As a representative of standard protocol models, we have introduced the set rewriting model $\mathbb{R}$, closely related to IF. We have shown that it is—in a certain sense—over-approximated by the message trace model $\mathbb{T}$ [116], which has inspired many models based on the exchange of messages. For a more precise account of the relationship, we have defined a persistent variant $\mathbb{P}$ of the set rewriting model; showing that this variant lies strictly between $\mathbb{R}$ and $\mathbb{T}$ demonstrates that there are in fact two orthogonal kinds of over-approximation. Persistent facts express the first kind of over-approximation: each reached local state of an honest agent remains present in the successor states, thereby allowing agents to continue any partial execution of the protocol an unbounded number of times. The second kind over-approximation results from the fact that the message trace model has no notion of local states of honest agents whatsoever, with

the result that each honest agent may act upon any subset of the messages it has sent and received so far—as if it was part of the same session. This leads to the fact that the over-approximation in general is not sound for authentication goals, as demonstrated by Example 16.1: there are protocols that suffer from an authentication flaw in the original model, but not in the over-approximation. We have shown how to correct this problem by introducing special messages similar to the goal facts in Section 4.5.

We further considered a related kind of control abstraction, basically turning from reachable states (traces) to reachable facts $\mathbb{F}$ (events $\mathbb{E}$). The latter kind of models is often used in approaches to verify protocols for an unbounded number of sessions [33, 36, 41, 42, 53, 80, 130]. There, the control abstraction is usually combined with a data abstraction similar to abstract interpretation ideas [63]. One then usually obtains a finite fixed-point, at least when using a typed model (or, more generally, in some way restricting the depth of messages that the intruder can generate).

We have formally proved for all models that for appropriate adaptations of the goals, we can check authentication and secrecy goals, i.e. every violation of the goals in the original model can also be detected in the over-approximated models.

We see the main contribution of this part as giving a precise account of several widely used models and their relationships. This formalization gives us the ability to recognize such subtle problems as the problem with authentication goals, and to give a provenly correct reformulation.

Besides a better understanding of the employed models, these results also pave the way for combining methods based on these models, in particular, connecting automated verification algorithms with a formalization in the theorem prover Isabelle in the style of [116]. Such a connection is motivated by the fact that an automated protocol verification tool is not unlikely to eventually verify a flawed protocol because of a bug in that tool, while the chance that a wrong security proof is accepted by Isabelle is much smaller. (Left aside mistakes and abstractions in the formalization itself.) The results of this work give an idea how such a connection may be possible, namely that the proof object generated by an automated verification tool would be based on an invariant on the set of messages that can ever be sent by an honest agent, and what messages the intruder can derive. We plan to investigate the details of such a connection as future work.

19.5 Experiments and Case Studies

The methods we have described in this thesis do not exist only on paper, but have been fully implemented in the tool OFMC (except, for now, off-line guessing). The experiments in Part V demonstrate that OFMC is feasible for the analysis of a wide range of protocols. First, OFMC supports the necessary concepts like algebraic reasoning and the concepts provided by IF (e.g. handling of sets and negative conditions). Second, OFMC incorporates efficient techniques to address the challenges of protocol analysis, namely the lazy intruder for taming the prolific Dolev-Yao intruder, constraint differentiation for addressing the interleaving problem for a larger number of sessions, and symbolic sessions to handle the session-instantiation problem.

The case study of the H.530 protocol demonstrates the practical value of OFMC in protocol design, namely to quickly detect non-trivial attacks on industrial-scale protocols and to verify the corrections. The case study on the ASW contract-signing protocol showed that our approach can also handle protocols with non-standard goals (like fair exchange) and assumptions (the intruder cannot block communication indefinitely) as well as complex structures (several subprotocols that cannot be considered in isolation, because the security properties rely on the interaction of the subprotocols). The case study on the SRP protocol shows that OFMC is capable of handling protocols that require a complex set of algebraic equations to be executable. In particular, OFMC allows us to model operations like explicit destructors that are performed by the honest agents on the messages, rather than introducing restrictive assumptions that have no counter-part in reality.

Although concrete running times like in Table 18.1 should not be over-interpreted, one can say that a quick response time of the tool is very helpful for practical work, as the user may "play" with several variants of the protocol, goals, or assumptions, which is at least not so easily possible when the user has to wait a day or more for the results. Also, the performance results in Table 18.2

and Table 18.3 give a representative picture of the situation: OFMC is among the most successful available tools in protocol analysis, both in terms of scope and performance. Nonetheless, other tools like SATMC and TA4SP based on completely different methods have other strengths, like the verification for a larger or even unbounded number of sessions. The trade-off, especially in the case of an unbounded number of sessions, is the set of assumptions on which these techniques rely, which explains the relatively small scope of such tools.

We can thus get the maximal benefit for protocol analysis in praxis when using architectures like the one of the AVISPA tool for connecting analysis tools with complementary strengths by a common input language.

19.6 Future Work

We conclude with an outlook of planned future work in the area of automated protocol analysis. One field that we wish to consider more in the future is the area of abstraction-based techniques (the implementation within OFMC is still in a prototypical state). One line of work that we envision is a bridge between an abstraction-based verification algorithm and a semi-automated theorem prover such as Isabelle. The idea is that the verification algorithm automatically produces proofs that can be checked with Isabelle. Assuming that Isabelle does not accept an incorrect proof, possible mistakes in the automated verification algorithm then do not matter anymore as the algorithm merely serves as a heuristics (that may fail) to find a proof.

Abstraction-based methods are helpful not only for considering an unbounded number of sessions, but also for other kinds of problems related to unbounded parameters of the problem. Consider the following examples:

- Group protocols may have an unbounded number of members.
- Several low-level protocols are more closely related to the architecture of the network; the set of possible topologies is also often unbounded.
- In protocols related to mobility, we have a similar problem: the concrete set of locations is infinite.

In all these cases, abstraction can be helpful, replacing the unbounded number of instantiations of the search parameter with a finite one, e.g. for group protocols distinguishing only the cases that an agent is (a) honest or dishonest and (b) is a member of the group or not a member of the group. We expect to obtain similar results for many such reductions from an infinite-state problem to an abstracted finite-state ones as in Part IV: if the original problem has a flaw, then also the abstract one has, so that safety of the abstract problem implies safety of the original one. An obstacle in abstraction is, however, that this process often introduces false positives that prevent the safety proof for the abstract problem (while the original problem may be fine). The challenge is thus to identify abstractions that work for a large class of cases, i.e. without introducing false positives.

Finally, there is an idea to use the lazy intruder techniques for questions of mobile code and trusted platforms: what can an intruder achieve with malicious code that will run on a certain platform? The point is that we can see malicious code as an intruder-generated “message”. With an appropriate extension of the lazy intruder technique, it may then be possible to lazily explore the possibilities of an intruder to write programs that run on a particular platform. Rather than exploring the infinite number of programs that the intruder could conceive, the programs are “written lazily”, based on the actions the program can perform in a certain state of the platform and on what messages the program can generate from all data it has access to. Thus, the lazy intruder intruder technique may offer feasible solutions also in the area of mobile code and trusted platforms.

As this list of ideas for future work shows, due to the growing number of security-sensitive electronic systems, we will not run of out of challenging questions in the near future.

Bibliography

[1] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. In *Proceedings of ICALP'2004*, LNCS 3142, pages 46–58. Springer-Verlag, 2004.

[2] M. Abadi and V. Cortier. Deciding knowledge in security protocols under (many more) equational theories. In *Proceedings of CSFW'05*, pages 62–76. IEEE Computer Society Press, 2005.

[3] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM, January 2001.

[4] M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 332(3):427–476, 2004.

[5] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. of Cryptology*, 15(2):103–127, 2002.

[6] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In C. Palamidessi, editor, *Proceedings of Concur'00*, LNCS 1877, pages 380–394. Springer-Verlag, 2002.

[7] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, 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 *Proceedings of CAV'05*. Springer-Verlag, 2005.

[8] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In *Proceedings of CAV'02*, LNCS 2404, pages 349–354. Springer-Verlag, 2002. URL of the AVISS and AVISPA projects: `www.avispa-project.org`.

[9] A. Armando and L. Compagna. Sat-based model-checking for security protocols. To appear in *International Journal of Information Security*.

[10] A. Armando and L. Compagna. Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In *Proceedings of FORTE 2002*, LNCS 2529, pages 210–225. Springer-Verlag, 2002.

[11] A. Armando, L. Compagna, and P. Ganty. SAT-based Model-Checking of Security Protocols using Planning Graph Analysis. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *Proceedings of the 12th International Symposium of Formal Methods Europe (FME)*, LNCS 2805, pages 875–893. Springer-Verlag, 2003.

[12] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, 1998.

[13] AVISPA. The AVISPA library. `http://www.avispa-project.org/library`.

[14] AVISPA. Deliverable 2.3: The Intermediate Format. Available at `http://www.avispa-project.org/publications.html`, 2003.

[15] AVISPA. Deliverable 6.1: List of selected problems. Available at `http://www.avispa-project.org`, 2003.

[16] AVISPA. Deliverable 7.4: Assessment of the AVISPA tool v.3. Available at `http://www.avispa-project.org`, 2005.

[17] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[18] F. Baader and K. U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computation*, 21:211–243, 1996.

[19] F. Baader and W. Snyder. Unification theory. In *Handbook of Automated Reasoning*, volume I, pages 445–532. Elsevier Science, 2001.

[20] M. Backes, S. Mödersheim, B. Pfitzmann, and L. Viganò. Symbolic and Cryptographic Analysis of the Secure WS-ReliableMessaging Scenario. In *Proceedings of FOSSACS 2006*, volume 3921 of *LNCS*, pages 428–445. Springer-Verlag, 2006.

[21] M. Backes and B. Pfitzmann. A Cryptographically Sound Security Proof of the Needham-Schroeder-Lowe Public-Key Protocol. *FST TCS: Foundations of Software Technology and Theoretical Computer Science*, 23, 2003.

[22] D. Basin. Lazy infinite-state analysis of security protocols. In *Proceedings of CQRE'99*, LNCS 1740, pages 30–42. Springer-Verlag, 1999.

[23] D. Basin and G. Denker. Maude versus Haskell: an Experimental Comparison in Security Protocol Analysis. In *ENTCS 36*. Elsevier, 2001.

[24] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In E. Snekkenes and D. Gollmann, editors, *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, 2003.

[25] D. Basin, S. Mödersheim, and L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols. In V. Atluri and P. Liu, editors, *Proceedings of CCS'03*, pages 335–344. ACM Press, 2003.

[26] D. Basin, S. Mödersheim, and L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols (Extended Abstract). In *Proceedings of SPV'03*. 2003.

[27] D. Basin, S. Mödersheim, and L. Viganò. Algebraic intruder deductions. In G. Sutcliffe and A. Voronkov, editors, *LPAR 2005*, volume 3835 of *LNAI*, pages 549–564. Springer-Verlag, December 2005.

[28] 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.

[29] G. Bella, F. Massacci, and L. C. Paulson. Verifying the set registration protocols. *IEEE Journal on Selected Areas in Communications*, 21(1), 2003.

[30] G. Bella, F. Massacci, and L. C. Paulson. Verifying the SET Purchase Protocols. *J. Automated Reasoning*, 2005.

[31] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 72–84, May 1992.

[32] G. Birkhoff. On the structure of abstract algebras. In *Proceedings Cambridge Phil. Soc.*, volume 31, 1935.

[33] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of CSFW'01*, pages 82–96. IEEE Computer Society Press, 2001.

[34] B. Blanchet. Automatic verification of cryptographic protocols: A logic programming approach (invited talk). In *Proceedings of PPDP'03*, pages 1–3. ACM Press, 2003.

[35] B. Blanchet. Security protocols: from linear to classical logic by abstract interpretation. *Information Processing Letters*, 95(5), 2005.

[36] Y. Boichut, P.-C. Héam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay technique to automatically verify security protocols. In *AVIS'04*, 2004.

[37] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP'01*, LNCS 2076, pages 667–681. Springer-Verlag, 2001.

[38] M. Boreale and M. G. Buscemi. A framework for the analysis of security protocols. In *Proceedings of CONCUR 2002*, LNCS 2421, pages 483–498. Springer, 2002.

[39] M. Bouallagui and H. Jain. Automatic Session Generation. AVISPA report, LORIA-INRIA-Lorraine, Nancy, 2003.

[40] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment.* Springer-Verlag, 2003.

[41] L. Bozga, Y. Lakhnech, and M. Perin. Pattern-based abstraction for verifying secrecy in protocols. In *Proceedings of TACAS 2003*, LNCS 2619. 2003.

[42] P. Broadfoot, G. Lowe, and A. Roscoe. Automating data independence. In *Proceedings of Esorics 2000*, LNCS 1895, pages 175–190. Springer-Verlag, 2000.

[43] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[44] C. Caleiro, L. Viganò, and D. Basin. Deconstructing Alice and Bob. *Electronic Notes in Theoretical Computer Science 135(1):3–22*, 2005.

[45] I. Cervesato, N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A Comparison between Strand Spaces and Multiset Rewriting for Security Protocol Analysis. In *Proceedings of ISSS 2002*, LNCS 2609, pages 356–383. Springer-Verlag, 2003.

[46] R. Chadha, M. Kanovich, and A. Scedrov. Inductive methods and contract-signing protocols. In *P. Samarati, editor, Proceedings, 8th ACM Conference on Computer and Communications Security*, pages 176–185, New York, November 2001. ACM Press.

[47] R. Chadha, J. C. Mitchell, A. Scedrov, and V. Shmatikov. Contract signing, optimism, and advantage. In *CONCUR: 14th International Conference on Concurrency Theory.* LNCS, Springer-Verlag, 2003.

[48] Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, J. Mantovani, S. Mödersheim, and L. Vigneron. *A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols*, volume 180 of *Automated Software Engineering*, pages 193–205. Austrian Computer Society, Austria, September 2004.

[49] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *Proceedings of LICS'03*, pages 261–270. IEEE Computer Society Press, 2003.

[50] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *Proceedings of FST TCS'03*, LNCS 2914, pages 124–135. Springer, 2003.

[51] Y. Chevalier and M. Rusinowitch. Combining Intruder Theories. In *Proceedings of ICALP 2005*, LNCS 3580, pages 639–651, 2005.

[52] Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *Proceedings of ASE'01*. IEEE Computer Society Press, 2001.

[53] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In *Proceedings of CAV'02*, LNCS 2404, pages 324–337. 2002.

[54] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997. URL: `www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`.

[55] E. Cohen. Proving protocols safe from guessing. In *Proceedings of Foundations of Computer Security 2002*, pages 85–92, 2002.

[56] H. Comon and V. Shmatikov. Is It Possible to Decide Whether a Cryptographic Protocol Is Secure Or Not? *Journal of Telecommunications and Information Technology*, 4:5–15, 2002.

[57] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *Proceedings of ESOP'2003*, LNCS 2618, pages 99–113. Springer-Verlag, 2003.

[58] H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In *Proceedings of RTA'05*, LNCS 3467, pages 294–307. Springer, 2005.

[59] R. Corin, J. Doumen, and S. Etalle. Analysing password protocol security against offline dictionary attacks. In *Proceedings of 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP)*. Electronic Notes in Theoretical Computer Science, 2004.

[60] R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proceedings of SAS 2002*, LNCS 2477, pages 326–341. Springer-Verlag, 2002.

[61] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? here is a new tool that finds some new guessing attacks (extended abstract). In R. Gorrieri and R. Lucchi, editors, *Proceedings of IFIP WG 1.7 and ACM SIGPLAN Workshop on Issues in the Theory of Security (WITS)*, pages 62–71, 2003.

[62] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, (1):1–43, 2006.

[63] P. Cousot. Abstract interpretation. *Symposium on Models of Programming Languages and Computation, ACM Computing Surveys*, 28(2):324–328, June 1996.

[64] C. Cremers. The scyther tool: Automatic verification of security protocols.

[65] H. B. Curry, J. R. Hindley, and J. P. Seldin. *Combinatory Logic, Vol. II.* North Holland, 1972.

[66] S. Delaune and F. Jacquemard. A decision procedure for the verification of security protocols with explicit destructors. In *Proceedings of CCS'04*, pages 278–287. ACM Press, 2004.

[67] S. Delaune and F. Jacquemard. A theory of guessing attacks and its complexity. In *Proceedings of CSFW'04*, pages 2–15, 2004.

[68] G. Denker and J. Millen. CAPSL Intermediate Language. In *Formal Methods and Security Protocols*, 1999. FLOC'99 Workshop.

[69] G. Denker, J. Millen, A. Grau, and J. Filipe. Optimizing protocol rewrite rules of CIL specifications. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW '00)*, pages 52–63, Washington - Brussels - Tokyo, July 2000. IEEE.

[70] G. Denker, J. Millen, and H. Rueß. The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, Menlo Park, CA, October 2000.

[71] W. Diffie, P. v. Oorschot, and M. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[72] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[73] B. Donovan, P. Norris, and G. Lowe. Analyzing a Library of Security Protocols using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[74] N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[75] S. Even and O. Goldreich. On the security of multi-party ping pong protocols. In *Proceedings of 24th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, 1983.

[76] F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.

[77] M. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proceedings of CSFW'01*. IEEE Computer Society Press, 2001.

[78] N. Furgerson and B. Schneier. A Cryptographic Evaluation of IPsec. 2000. Available at `http://www.counterpane.com/ipsec.pdf`.

[79] J. A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Proc. 19th International Advances in Cryptology Conference – CRYPTO '99*, pages 449–466, 1999.

[80] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proceedings of CADE'00*, LNCS 1831, pages 271–290. Springer-Verlag, 2000.

[81] A. Gordon and M. Abadi. A calculus for cryptographic protocols: The spi calculus. In *4th ACM Conference on Computer and Communications Security*, 1997.

[82] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn. RFC 2637: Point-to-Point Tunneling Protocol, July 1999.

[83] P. Hankes Drielsma and S. Mödersheim. The ASW Protocol Revisited: A Unified View. In *Proceedings of the Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA 2004)*, pages 141–156. Electronic Notes in Theoretical Computer Science 125 (Elsevier Science Direct), July 2005.

[84] P. Hankes Drielsma, S. Mödersheim, and L. Viganò. A formalization of off-line guessing for security protocol analysis. In F. Baader and A. Voronkov, editors, *Proceedings of LPAR'04*, volume 3452 of *LNAI*, pages 363–379. Springer, 2005.

[85] P. Hankes Drielsma, S. Mödersheim, L. Viganò, and D. Basin. Formalizing and analyzing sender invariance. In *Formal Aspects of Security and Trust*, 2006. To appear. An intermediate version of this paper is available as ETH technical report no. 528.

[86] D. Harkins and D. Carrel. RFC 2409: The Internet Key Exchange (IKE). 1998.

[87] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of The 13th Computer Security Foundations Workshop (CSFW'00)*. IEEE Computer Society Press, 2000.

[88] N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22(1):16–30, 1996.

[89] C. A. R. Hoare. CSP — Communicating Sequential Processes, 1985. Available at `http://www.usingcsp.com`.

[90] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[91] A. Huima. Efficient infinite-state analysis of security protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[92] ITU-T Recommendation H.530: Symmetric Security Procedures for H.510 (Mobility for H.323 Multimedia Systems and Services), 2002.

[93] ITU-T Recommendation H.530, Corrigendum 1. 2003. Corrected version of [92].

[94] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Proceedings of LPAR 2000*, LNCS 1955, pages 131–160. Springer-Verlag, 2000.

[95] D. Kapur, P. Narendran, and L. Wang. An E-unification algorithm for analyzing protocols that use modular exponentiation. In *Proceedings of RTA 2003*, LNCS 2706, pages 165–179. Springer, 2003.

[96] C. Kirchner and H. Kirchner. *Rewriting, Solving, Proving*. A preliminary version of a book available at `http://www.loria.fr/~ckirchne/rewriting.html`.

[97] P. Lafourcade, D. Lugiez, and R. Treinen. Intruder deduction for AC-like equational theories with homomorphisms. In *Proceedings of RTA'05*, LNCS 3467, pages 308–322. Springer, 2005.

[98] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[99] G. Lowe. Breaking and Fixing the Needham-Shroeder Public-Key Protocol Using FDR. In T. Margaria and B. Steffen, editors, *Proceedings of TACAS'96*, LNCS 1055, pages 147–166. Springer-Verlag, 1996.

[100] G. Lowe. Some new attacks upon security protocols. In *Proceedings of The 9th Computer Security Foundations Workshop (CSFW'96)*. IEEE Computer Society Press, 1996.

[101] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of CSFW'97*, pages 31–43. IEEE Computer Society Press, 1997.

[102] G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

[103] G. Lowe. Towards a completeness result for model checking of security protocols. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW'98)*, pages 96–105. IEEE Computer Society Press, 1998.

[104] G. Lowe. Analysing protocols subject to guessing attacks. *Journal of Computer Security*, 12(1), 2004.

[105] The Maude System. URL: `http://maude.csl.sri.com`.

[106] L. Mazaré. Satisfiability of dolev-yao constraints. In *Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA)*, 2004.

[107] C. Meadows. Analyzing the Needham-Schroeder Public-Key Protocol: A Comparison of Two Approaches. In *ESORICS: European Symposium on Research in Computer Security*. LNCS, Springer-Verlag, 1996.

[108] C. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[109] C. Meadows. Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1999.

[110] J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. *Journal of Higher-Order and Symbolic Computation*, 2005.

[111] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of CCS'01*, pages 166–175. ACM Press, 2001.

[112] J. K. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proceedings of CSFW'03*, pages 47–61. IEEE Computer Society Press, 2003.

[113] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. Technical Report CSL-78-4, Xerox Palo Alto Research Center, Palo Alto, CA, USA, 1978. Reprinted June 1982.

[114] F. Oehl, G. Gécé, O. Kouchnarenko, and D. Sinclair. Automatic approximation for the verification of cryptographic protocols. In *FASec'02*, 2002.

[115] L. C. Paulson. *Isabelle: a Generic Theorem Prover*. LNCS 828. Springer-Verlag, 1994.

[116] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1):85–128, 1998.

[117] D. Peled. Ten Years of Partial Order Reduction. In *Proceedings of CAV 1998*, LNCS 1427, pages 17–28. Springer-Verlag, 1998.

[118] A. Perrig and D. Song. Looking for diamonds in the desert — extending automatic protocol generation to three-party authentication and key agreement protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW '00)*. IEEE Computer Society Press, 1999.

[119] S. Peyton Jones et al. Haskell 98 language and libraries. 2002. `www.haskell.org`.

[120] A. Roscoe and P. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7:147–190, 1999.

[121] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In *Proceedings of CSFW'01*. IEEE Computer Society Press, 2001.

[122] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Educations, 2nd edition, 2003.

[123] B. Schneier, Mudge, and D. Wagner. Cryptanalysis of microsoft's PPTP authentication extensions (MS-CHAPv2). In *CQRE: International Exhibition and Congress on Secure Networking – CQRE [Secure]*, 1999.

[124] V. Shmatikov and J. C. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science*, 283(2):419–450, June 2002.

[125] J. Siekmann and P. Szabó. The undecidability of the D_A unification problem. *Journal of Symbolic Computation*, 54(2):402–414, 1989.

[126] D. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9:47–74, 2001.

[127] G. Steele. Deduction with XOR Constraints in Security API Modelling. In *CADE 20*, 2005.

[128] M. Turuani. *Sécurité des Protocoles Cryptographiques: Décidabilité et Complexité.* Phd-thesis, Université Henri Poincaré, Nancy, December 2003.

[129] V. Vanackère. History-dependent scheduling for cryptographic processes. In *Verification, Model Checking, and Abstract Interpretation (VMCAI 2004)*, LNCS, pages 16–29, 2004.

[130] C. Weidenbach. Towards an automatic analysis of security protocols. In H. Ganzinger, editor, *Proceedings of CADE'99*, LNCS 1632, pages 378–382. Springer-Verlag, Berlin, 1999.

[131] T. Wu. The Secure Remote Password Protocol. In *Proc. of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1998.

[132] T. Wu. The SRP Authentication and Key Exchange System, 2000.

[133] J. Zhou. Further Analysis of the Internet Key Exchange Protocol. *Computer Communications*, 23(17):1606–1612, 2000.

[134] G. Zorn. RFC 2759: Microsoft PPP CHAP Extensions, Version 2, Jan. 2000.

Curriculum Vitae

Since 2003 Researcher and doctoral student in Prof. Basin's information security group at the Swiss Federal Institute of Technology (ETH Zürich).

2002 Visiting researcher in a 3-months project at Siemens AG, Munich, Germany.

2001–2002 Researcher in Prof. Basin's software engineering group at the Albert-Ludwigs-Universität Freiburg, Germany.

1995–2001 Student of computer science with second subject psychology at the University of Freiburg. Diploma in computer science with grade *very good*.