Advancing automated security protocol verification

Author(s):
Meier, Simon

Publication Date:
2013

Permanent Link:

https://doi.org/10.3929/ethz-a-009790675 →

DISS. ETH NO. 20742

ADVANCING AUTOMATED SECURITY PROTOCOL VERIFICATION

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

presented by
SIMON MEIER,
Dipl. Ing. Inf., ETH Zürich
born on August 3rd, 1982
citizen of Luzern, Switzerland

accepted on the recommendation of
Prof. Dr. David Basin, examiner
Prof. Dr. Ueli Maurer, co-examiner
Dr. Steve Kremer, co-examiner
Dr. Cas Cremers, co-examiner

2013

Abstract

Security protocols are distributed algorithms for achieving security goals, like secrecy or authentication, even when communicating over an insecure network. They play a critical role in modern network and business infrastructures. This thesis focuses on the automated verification of security protocols in symbolic models of cryptography. The verification of security protocols is particularly important, as their design is error-prone, and errors may lead to the loss of money or even human lives.

The automated verification of security protocols has already been the subject of much research. Automatic verification of all protocols is however impossible to achieve, as protocol security is undecidable in general. The goal of existing research works and this thesis is therefore to extend the scope of automated verification to cover increasingly many practical verification problems. In this context, our development of the theory and implementation of the Tamarin prover represents a significant step forward. In particular, Tamarin is the first tool that supports, without requiring any bounds, the automatic falsification and verification of protocols making use of loops and non-monotonic state, and of protocols that use Diffie-Hellman exponentiation to achieve resilience against strong adversaries. The theory underlying the Tamarin prover is of particular interest because it is derived from a simple, but expressive security protocol model, and because it is constructed in a way that simplifies future extensions.

Apart from extending the scope of automated security protocol verification, this thesis also provides an answer to the question of how to improve the trustworthiness of a result obtained by an automatic security protocol verification tool. We explain a generic approach to construct proof-generating versions of security protocol verification algorithms. We validate this approach by implementing `scyther-proof`, a version of Scyther that generates *machine-checked* proofs. We demonstrate the practical applicability of `scyther-proof` and Scyther on the ISO/IEC 9798 standard for entity authentication, which is used as a core building block in numerous other standards. When analyzing the standard using the Scyther tool, we surprisingly find that the most recent version of this standard still exhibits both known and new weaknesses. We therefore propose fixes and use `scyther-proof` to generate machine-checked proofs of the correctness of our repaired protocols. The ISO/IEC working group responsible for the 9798 standard has released an updated version of the standard based on our proposed fixes.

Zusammenfassung

Sichertheitsprotokolle sind verteilte Algorithmen für den sicheren Datenaustausch über unsichere Netzwerke wie zum Beispiel das Internet. Sichertheitsprotokolle basieren oft auf kryptographischen Techniken und spielen eine kritische Rolle in modernen Netzwerken und Geschäftsprozessen. Diese Dissertation behandelt die automatische Verifikation von Sicherheitsprotokollen in symbolischen Modellen der Kryptographie. Die Verifikation von Sicherheitsprotokollen ist besonders wichtig, da ihr Design fehleranfällig ist, und weil Fehler in Sicherheitsprotokollen grosse Kosten verursachen können.

Die automatische Verifikation von Sicherheitsprotokollen war bereits Gegenstand von vielen Forschungsarbeiten. Es ist allerdings unmöglich eine Methode zu entwickeln, die alle Protokolle vollautomatisch verifizieren kann, da die Sicherheit von Protokollen unentscheidbar ist. Daher verfolgen sowohl die existierenden Arbeiten wie auch diese Dissertation das Ziel, die Anwendbarkeit von automatischen Methoden im Bezug auf praktisch relevante Protokolle zu erhöhen. In diesem Kontext ist unsere Entwicklung der Theorie und der Implementierung des Tamarin Beweisers ein signifikanter Fortschritt. Der Tamarin Beweiser ist das erste Werkzeug, das sowohl Protokolle mit Schleifen und nicht-monotonischem Zustand als auch Diffie-Hellman Protokolle im Bezug auf starke Angreifer vollautomatisch und ohne zusätzliche Beschränkungen verifizieren kann. Die Theorie, welche dem Tamarin Beweiser zugrunde liegt, ist aus zwei Gründen von besonderem Interesse. Erstens basiert sie auf einem einfachen aber sehr ausdrucksstarken Modell von Sicherheitsprotokollen. Und zweitens ist sie so strukturiert, dass zukünftige Erweiterungen einfach darauf aufgebaut werden können.

Abgesehen von der Entwicklung des Tamarin Beweisers, geben wir in dieser Dissertation auch eine Antwort auf die Frage, wie man die Vertrauenswürdigkeit der Resultate von vollautomatischen Verifikationswerkzeugen erhöhen kann. Wir formulieren einen generischen Ansatz, um einen Algorithmus zur Sicherheitsprotokollverifikation so zu modifizieren, dass er automatisch Sicherheitsbeweise generiert. Wir validieren diesen Ansatz indem wir `scyther-proof` implementieren. Dies ist eine Version des Scyther Tools, welche automatisch Sicherheitsbeweise generiert und in dem Isabelle/HOL Theorembeweiser überprüft. Wir demonstrieren die praktische Anwendbarkeit von `scyther-proof` und dem Scyther Tool an dem ISO/IEC 9798 Standard for Entitity Authentication, welcher eine Kernkomponente in weiteren Standards ist. Überraschenderweise stellten wir bei der Analyse der aktuellsten Version des Standards mit dem Scyther Tool fest, dass diese Version immer noch bereits bekannte und auch neue Mängel aufweist. Wir schlugen daher Reparaturen vor und bewiesen die Korrektheit unserer Reparaturen mit Hilfe von `scyther-proof`. Die für den 9798 Standard verantwortliche ISO/IEC Arbeitsgruppe hat daraufhin eine aktualisierte Version des Standards herausgegeben, welche auf unseren Reparaturen basiert.

Acknowledgments

First, I would like to thank David Basin. Without his great lectures that I enjoyed during my diploma studies here at ETH, I would not have started this PhD — and without his generous support and scientific advice, I could not have written this thesis. I am also deeply indebted to Cas Cremers, who has guided me with lots of good advice and humor around all the obstacles that I encountered during my PhD. I could not have wished for a better mentor. I also want to thank Benedikt Schmidt for the time we spent together on developing the Tamarin prover. I really enjoyed our collaboration and I am looking forward to future projects.

Second, I would like to thank all the current and former members of the ETH Institute of Information Security for the pleasant and productive working atmosphere. I especially thank my officemates Patrick Schaller and Benedikt Schmidt for the many enlightening discussions, Achim Brucker and Burkhart Wolff for introducing me to Isabelle/HOL, Barbara Geiser for getting rid of *any* administrative overhead, and Christoph Sprenger, Ralf Sasse, Saša Radomirović, Felix Klaedtke, Srdan Marinovic, Marko Horvat, Michèle Feltz, and Ognjen Maric for their valuable comments on my work. I also thank Martin Schaub and Cedric Staub for the contributions they made to my work as part of their master's and bachelor's theses.

Third and to no lesser extent, I would like to thank all my friends for the many sportive, culinary, entrepreneurial, witty, and otherwise very interesting times and adventures that we enjoyed together in the last five years. They kept me balanced and well-earthed despite the rather abstract, and sometimes very demanding nature of my PhD. I especially thank Yves, Jonas, Christof, Andy, Björn, Chrissy, Corinne, Daniel, Daniela, Erik, Fabian, Marc, Martin, Manuel, Robi, Silvan, Tom C., Tom D., Thorsten, and Varja.

Most importantly, I would like to thank my parents, Roberto and Christina, and in particular Andrea, for their unconditional love and support.

Contents

1. Introduction

This thesis is about improving the state of the art in automated symbolic security protocol verification. We motivate our work and the two research questions that it is based on in Section 1.1. We then detail our approach for solving these two questions in Section 1.2. We highlight the contributions of this thesis in Section 1.3 and give an overview of the thesis in Section 1.4.

1.1. Motivation

Security protocols are distributed algorithms for achieving security goals, like secrecy or authentication, even when communicating over an untrusted network. As witnessed by the large number of flaws in published or deployed security protocols, e.g., [JV96, Low96, BWM99b, LM06, Cre10, Far10, ACC+08, WCW12, MT12], designing security protocols is error-prone. Due to their critical role in modern network and business infrastructures, it seems prudent to require that security protocols are designed using a methodology that not only yields functional specifications, but also formal security proofs.

The first step towards such a methodology is to formally define what it means for a protocol to be secure. At a high level, one can distinguish between security definitions stated in a computational model and security definitions stated in a symbolic model. A computational security definition roughly states that violating the security of a protocol is as difficult as efficiently solving a computational problem that is assumed to be hard, e.g., integer factorization or computing discrete logarithms.

Computational security definitions are generally considered the most faithful definitions because of the following two properties. First, computational security definitions reduce the security of a protocol to explicit, well-known hardness assumptions. Second, computational security definitions are formalized as probabilistic statements that relate the probability of a successful attack to the size of the used keying material. This allows for the construction of security proofs that provide concrete bounds on the probability of a successful attack. These bounds could in theory be used to determine the size of the keying material in a concrete application, e.g., by performing a risk assessment.

Proving that a protocol satisfies a computational security definition is however a difficult, laborious, and usually manual task. One of the difficulties is to prove that the protocol cannot be broken by exploiting only the intended relations between the cryptographic algorithms employed in a protocol, e.g., that decrypting an encrypted message with the right key yields the message itself. Flaws that only exploit these intended interactions are *logical flaws* in the protocol design. A large number of flaws in published or deployed protocols are actually logical flaws. One of the reasons for the prominence of logical flaws is the counterintuitive nature of the execution of security protocols in the context of an active adversary. Such an adversary is typically assumed to have complete control over the untrusted network. He thus learns every message sent

and can block and insert his own messages. Moreover, the adversary may concurrently communicate with an unbounded number of protocol participants. Typically, one further assumes that the adversary can act as an unbounded number of legitimate participants of the protocol. These abilities often allow the adversary to violate the security of a protocol by cleverly recombining the messages sent by different protocol participants.

Symbolic security protocol models formalize the execution of a security protocol in the context of an active adversary who only exploits a fixed set of interactions between the employed cryptographic algorithms. Symbolic models allow to construct symbolic protocol security proofs that prove the absence of logical flaws. The key idea behind symbolic models is to represent the messages exchanged over the network not as bitstrings, but as terms that describe what cryptographic algorithms were used to construct these messages. The interactions between the cryptographic algorithms can then be modeled as equalities between these terms. For example, the decryption of an encrypted message with the same key could be represented as the term $\mathrm{dec}(\mathrm{enc}(m, k), k)$, which would be considered to be equal to the term m when modeling symmetric encryption.

In this thesis, we focus on the automated construction of symbolic protocol security proofs, both for existing and future protocols. We are interested in principled approaches that allow a protocol designer to state and formally prove the absence of logical flaws in his protocols. To allow for a wide adoption of such an approach, we desire proof construction to be automated as much as possible. Unfortunately full automation is unachievable in general, as symbolic security protocol verification is undecidable without bounding the number of considered protocol instances [DLM04, TEB05].

Nevertheless, research in *unbounded* symbolic security protocol verification has made significant progress in the last ten years. There now exist several specialized tools (e.g., ProVerif [Bla01], Scyther [Cre08a], and CPSA [RG09]) that automatically verify secrecy and authentication properties of models of classic protocols like Kerberos V [Bel07] or the TLS handshake [Pau99] in a few seconds for an unbounded number of sessions. The correctness of results obtained using these tools obviously depends the soundness of the proofs justifying their underlying theory and the correctness of their implementations. Because of the complexity of these proofs and implementations, one may rightfully question the correctness of such results. This motivates our first research question.

> **Research Question 1:** How can one improve the trustworthiness of the results obtained using an automatic symbolic security protocol verification tool?

ProVerif is the most widely applied symbolic security protocol verification tool. The core idea underlying ProVerif is to reduce the existence of an attack on a security protocol to the derivability of a fact in a Horn theory, which is a standard problem in logic. This reduction over-approximates the execution of a security protocol and may give rise to false attacks, i.e., derivations that do not correspond to actual attacks. Moreover, derivability in a Horn theory is still undecidable in general. To decrease the chance of non-termination and reporting false attacks, ProVerif uses a carefully engineered reduction and a specialized resolution prover. In practice [Bla09], these techniques enable ProVerif to successfully analyze many protocol verification problems.

ProVerif is very flexible due to the expressivity of Horn theories. To adapt it to a new type of verification problem, it suffices to develop a corresponding reduction. For the

analysis of the resulting Horn theories, one then profits from all the engineering effort that was put into its resolution prover. Some problems might require reductions that result in Horn theories that the prover has difficulty coping with. In this case, further effort has to be spent on improving the prover. The benefit of the approach underlying ProVerif is that the resulting improvements are then available to the whole range of protocol verification problems that can be reduced to Horn theories.

ProVerif's reliance on Horn theories is also its weakest point. Modeling the execution of a protocol as a Horn theory introduces a monotonicity assumption: once an action of a protocol participant is enabled in such a model, it can be repeated unboundedly many times. Hence, one cannot easily use Horn theories to verify a protocol whose security relies on *non-monotonic state*, i.e., protocols where certain actions are enabled in a state, but required to be disabled in a later state.

Recently, there has been a lot of interest in the verification of cryptographic APIs like the Trusted Platform Module using existing security protocol analysis tools [Her06]. The main difficulty of this approach is that it requires faithfully modeling non-monotonic state. Nevertheless, first steps towards reducing the resulting verification problems to Horn theories have been made [Möd10a, DKRS11, ARR11]. Unfortunately, the employed reductions are fragile and rely on manually proving additional protocol-specific properties, e.g., the soundness of protocol-specific abstractions.

In contrast to ProVerif, tools like Scyther and CPSA do not use any abstraction, but still manage to verify classic security protocols for an unbounded number of sessions. Their applicability is however limited by their restricted security protocol model. Both of them represent protocols as sets of roles where each role is a linear sequence of send and receive steps, and both of them hard-code a fixed set of adversary models. Modeling protocols with loops or global state, or modeling user-defined adversaries is for example out of their scope. Nevertheless, their ability to reason about classic security protocols without using abstraction is intriguing. This motivates our second research question.

> **Research Question 2:** How to verify security protocols specified in a formalism that is at least as expressive as Horn theories, but supports the faithful modeling of non-monotonic state.

1.2. Approach Taken

In the following, we describe the approach that we use to solve our two research questions. As shown in [CLN09], the performance of the Scyther tool is close to the one of ProVerif for the analysis of secrecy and authentication properties of classic security protocols like NSL [Low96] or the TLS handshake [Pau99]. Moreover, the algorithm underlying the Scyther tool is well-documented [Cre06, Cre08b] and does not use abstraction. We therefore base our solutions on the Scyther tool.

Research Question 1

We solve our first research question by developing a proof-generating version of the algorithm underlying the Scyther tool. The generated proofs are represented such that they are human readable and can be checked using the theorem prover. In our

implementation of this algorithm, called `scyther-proof`, we check the generated proofs using the Isabelle/HOL theorem prover [NPW02]. This significantly improves the trustworthiness of these proofs because Isabelle/HOL is renowned in the verification community for its careful design to avoid accepting flawed proofs. Moreover, this checking allows us to obtain protocol security proofs whose trustworthiness is *independent* of any implementation errors in `scyther-proof`.

The generated proofs are composed from instances of a small number of theorems. Each theorem formalizes one of the steps that our proof-generating security protocol verification algorithm may take. We ensure the soundness of these theorems by *formally deriving* them in Isabelle/HOL from a straightforward, Dolev-Yao-style [DY81] security protocol execution model. Note that our protocol execution model is untyped, i.e., protocol variables may be instantiated with arbitrary messages. The generated proofs therefore also entail the absence of type-flaw attacks [HLS03].

We validate the practical applicability of the `scyther-proof` tool on standard examples from the literature and on the ISO/IEC 9798 standard for entity authentication [Int10a]. The results are positive. For example, generating proofs of secrecy and non-injective authentication for the TLS handshake takes less than a second on an Intel i7 Quad Core processor. Checking these proofs in Isabelle/HOL takes half a minute.

Research Question 2

Our solution to the second research question is inspired by the insights that we gained from the development of `scyther-proof`. In particular, we found that we could use the theorems justifying the proof steps taken by `scyther-proof` as a verification theory for security protocols that allows the construction of succinct protocol security proofs. This theory highlights the key ideas for constructing, *without using abstraction*, protocol security proofs that hold for an unbounded number of sessions. In the following, we first explain these ideas. Afterwards, we describe how we generalize them to a simple, but expressive security protocol model.

Key ideas underlying `scyther-proof`'s verification theory

The first idea is to extend the protocol execution model such that the message deduction performed by the adversary is made explicit. This allows us to formalize efficient adversaries, which do not perform redundant message deductions, e.g., deducing the same message twice. We then observe the following invariant about message deduction performed by an efficient adversary: every message known to the adversary was either constructed by himself or it was extracted from a message sent by the protocol using a chain of decryptions. In such a chain, the input of each decryption, except the first one, is provided by the output of the preceding decryption. This invariant gives rise to an inference rule, which we call the Chain rule. It formalizes a finite and complete enumeration of the possible origins of messages known to the intruder.

In `scyther-proof`'s verification theory, we use the Chain rule to prove security properties as follows. We start by assuming that there exists an attack on the security property to be proven. Hence, the intruder must know certain messages. We make a case distinction on how the intruder learned these messages using the Chain rule. In

each case, we gain additional assumptions about the structure of a possible attack. If these assumptions lead to a contradiction, then no attack exists. Otherwise, we again apply the Chain rule to iteratively refine the structure of possible attacks. Once we have shown that every case is contradictory, we have proven that the security property holds.

As we can see, this proof strategy is based on backwards-reasoning. This is a common strategy to reason about infinite-state systems. Backwards-reasoning is goal-directed and thereby avoids exploring parts of the state that are irrelevant to the property of interest. To successfully construct a proof using backwards-reasoning one requires strong inference rules, as each backwards step must result in proof obligations that are smaller (according to some measure) than the previous ones. It is easy to see that a naive inference rule that just enumerates the possible message deduction steps that the adversary may have taken to deduce the message of interest is often insufficient to reason about decryption. The problem is that there could always be some larger encrypted message from which the adversary learned the message of interest by decryption. The Chain rule avoids this problem by starting with an arbitrary message sent by the protocol and locally reasoning in a forward-fashion about what messages may be decrypted from it.

This local forward-reasoning about decryption requires one additional idea for the successful proof construction. The problem is that a protocol typically extracts certain parts from a message received from the network and then recombines these parts to messages sent to the network. In a protocol's specification, these message parts are referenced by variables, which are instantiated upon the receipt of a message and then used in the construction of messages sent to the network. As we are working with an untyped model, these message parts can in general be arbitrary messages. We therefore need a way to determine what messages can be decrypted from such message parts. We solve this problem by introducing protocol-specific type assertions.

These type assertions generalize the types naturally occurring in protocol specifications so that they also account for the cases where the adversary constructs the received message. For example, suppose that a protocol specifies that a variable v is of type nonce. We then assert that v is always instantiated with either (1) a nonce or (2) some message that the adversary knew before v was instantiated. The first case accounts for receiving a message sent by another honest protocol participant. The second case accounts for receiving a message from the adversary, who need not follow any typing discipline. We use such type assertions together with the Chain rule to reason about what the adversary can decrypt from the part of a received message referenced by a variable. Note that we do not assume a-priori that a protocol satisfies its type assertions; this must be proven. We construct these proofs using a specialized induction scheme that we derive from our protocol execution model.

Generalizing `scyther-proof`'s verification theory

Summarizing, there are four key ideas underlying `scyther-proof`'s verification theory: goal-directed backwards-reasoning, efficient adversaries, local forward-reasoning, and protocol-specific type assertions. None of these ideas fundamentally relies on restrictions of the security protocol execution model from which we derived this verification theory. To solve our second research question, we adapt these ideas to a model that represents

messages as terms modulo an equational theory, protocols as labeled multiset-rewriting systems, and security properties as two-sorted first-order-logic formulas, which allow quantification over timepoints and messages.

As we will demonstrate, this model is both simple and expressive. In particular, this model naturally supports modeling non-monotonic state and comprises Horn-theories as a special case. In the following, we give an overview of the resulting verification theory and explain how we validate its practical applicability.

To formalize efficient adversaries, we introduce the notion of dependency graphs, which can be seen as a generalization of strand spaces [THG99] to multiset rewriting. We characterize the execution of a protocol in the context of an efficient adversary by the set of dependency graphs satisfying certain normal form conditions, e.g., no message being deduced twice. The concrete normal form conditions depend on the equational theory of interest. Their goal is to enable the effective reasoning about message deduction using a generalization of the Chain rule. We provide normal form conditions for reasoning about the combination of an arbitrary subterm-convergent rewriting theory and a theory modeling Diffie-Hellman (DH) exponentiation. We use them for example to reason about protocols that combine DH exponentiation with standard cryptographic primitives like signatures, symmetric and asymmetric encryption, and hashing.

We formalize the goal-directed backwards-reasoning using constraint solving. The solutions of a constraint system are all normal form dependency graphs that represent executions of the protocol of interest *and* satisfy the system's constraints. To prove a security property, we show that the constraint system denoting all counter-examples to this property has no solutions. We show this using a set of constraint-reduction rules, which generalize the inference rules constituting `scyther-proof`'s verification theory, e.g., the Chain rule.

The resulting verification theory also allows the use of protocol-specific invariants, which generalizes the use of type assertions. These invariants enable reasoning about protocols with loops, e.g., the TESLA stream authentication protocol [PCTS00] or the Yubikey security device [KS12]. We prove these invariants using induction over the traces of the protocol of interest. The resulting proof obligations are again discharged using constraint solving.

To validate the practical applicability of our verification theory, we implemented it in a tool, the Tamarin prover [MS12]. Tamarin supports both the falsification and the unbounded verification of security properties of protocols formalized in the expressive model described above. The user interface of Tamarin provides both an interactive and an automatic mode. In the interactive mode, the user selects the next constraint-reduction rule to apply to a constraint system using a GUI. To support the user in his verification task, the intermediate constraint systems are visualized as partial protocol executions. The interactive mode thereby allows a user to gain valuable insights into the workings of a security protocol. The interactive mode also provides valuable insights into non-terminating proof attempts, which are bound to occur because the considered problems are undecidable in general.

In the automatic mode, the Tamarin prover uses a heuristic to select the next constraint-reduction rule to apply to a constraint system. Note that some of our constraint-reduction rules perform case distinctions. In general, a constraint-reduction rule therefore reduces a single constraint system to multiple constraint systems. A

reduction of a constraint system to the empty set of constraint systems therefore certifies that this system has no solutions. In the fully automatic mode, the Tamarin prover terminates once it has reduced the constraint system denoting all counter-examples *either* to the empty set of constraint systems *or* to a solved constraint system from which a concrete counter-example can be extracted immediately. Despite the undecidability of the considered problem, the Tamarin prover often terminates and succeeds in the automatic analysis of a wide range of security protocol verification problems.

In particular, we show that the Tamarin prover succeeds in automatically analyzing several recent Authenticated Key Exchange (AKE) protocols that make use of DH exponentiation to achieve security in strong adversary models such as the eCK security model [LLM07]. We also show that Tamarin succeeds in automatically analyzing several case studies from the works on extending ProVerif with support for non-monotonic state, i.e., [Möd10a, DKRS11, ARR11]. In contrast to these works, we do not require discharging non-trivial manual proof obligations, e.g., the soundness of protocol-specific abstractions. We may require the specification of additional invariants, but their soundness is always proven automatically by the Tamarin prover. We further demonstrate the use of Tamarin to reason about protocols with loops on the basic version of the TESLA protocol [PCTS00], a protocol that was previously out of scope of automatic security protocol verification tools. Finally, we show that the Tamarin prover is similarly efficient as existing state-of-the-art tools for the automatic verification of classic security protocols.

1.3. Contributions

We identify three main contributions in our work.

1. We develop the theory and the implementation of `scyther-proof`, a proof-generating version of the security protocol verification tool Scyther. The generated proofs provide strong correctness guarantees, as they are machine-checked in Isabelle/HOL and all employed inference rules are formally derived from a straightforward, untyped security protocol execution model. Our presentation of the verification theory underlying `scyther-proof` is of independent interest because it highlights the key ideas for constructing unbounded protocol security proofs without using abstraction.

2. We demonstrate both the usefulness and feasibility of the formal verification of an actual security protocol standard. We analyze the ISO/IEC 9798 standard for entity authentication and find that its most recent version still exhibits both known and new weaknesses. We propose fixes and use `scyther-proof` to generate machine-checked proofs of the repaired protocols. The resulting proofs imply the absence of logical errors: the repaired protocols provide strong authentication properties in our Dolev-Yao-style model, even when multiple protocols from the standard are run in parallel using the same key infrastructure. The ISO/IEC working group responsible for the 9798 standard has released an updated version of the standard based on our proposed fixes.

3. We contribute to the development of the theory and the implementation of the Tamarin prover. It supports both interactive and automatic, falsification and unbounded verification of properties of security protocols specified in a very expressive model. In particular, this model generalizes the Horn-theory based model employed by ProVerif with direct support for modeling non-monotonic state. We validate the wide applicability of the Tamarin prover in a number of case studies. In particular, we show that Tamarin allows the automatic analysis of recent Diffie-Hellman based AKE protocols, protocols employing unbounded loops, and protocols whose correctness arguments depend on non-monotonic state. Several of our case studies were previously out of scope for automatic security protocol verification tools.

Note that the theory and implementation of the Tamarin prover was jointly developed with Benedikt Schmidt, building upon `scyther-proof` as outlined in the previous section. We published first results in [SMCB12a]. This publication details the theory that we use to analyze Diffie-Hellman based AKE protocols and the corresponding case studies. This thesis additionally develops the theory of trace induction and validates its usefulness on several new case studies. Schmidt's thesis [Sch12] further extends [SMCB12a] with support for equational theories that model bilinear pairing and multiset union.

The presentation of the theory and implementation of the Tamarin prover given in this thesis is a rewritten and extended version of [SMCB12a]. The theory is developed in several steps and many more examples are provided. The goal of the presentation given in this thesis is to facilitate further extensions of our work by explicating all its assumptions and design decisions.

1.4. Overview

This thesis consists of two parts. The first part details the theory and implementation of `scyther-proof` and the analysis of the ISO/IEC 9798 standard. It is based on the publications [MCB10, BCM12, MCB13, BCM13], which are joint work with Cas Cremers and David Basin. The second part details the theory and implementation of the Tamarin prover. It is based on the publication [SMCB12a], which is joint work with Benedikt Schmidt, Cas Cremers, and David Basin.

We introduce concepts and notation common to both parts in Chapter 2. Jointly for both parts, we present related work in Chapter 10 and conclusions and future work in Chapter 11. Otherwise, the two parts are self-contained. We outline their chapters below.

Part I: Improving the Trustworthiness

In Chapter 3, we define the protocol execution model from which we derive the verification theory underlying `scyther-proof`. This is a straightforward Dolev-Yao-style model similar to the operational semantics of security protocols proposed by Cremers and Mauw [CM05]. More precisely, we model cryptographic messages built from symmetric and asymmetric encryption, signing, and hashing. We use a free term algebra to represent these messages. We assume that both unidirectional and bidirectional symmetric keys

between pairs of agents and all agents' public keys are pre-shared. As is standard, we model an adversary that has complete control over the network. He moreover has access to the pre-shared keying material of an unbounded, but statically determined set of compromised agents. The adversary may therefore actively participate in a protocol. Protocols are modeled as role scripts, i.e., sets of linear sequences of send and receive steps. We use message patterns to specify how messages are composed and decomposed. To faithfully model protocols that blindly forward composed messages (e.g., Kerberos IV [BP98]), we allow variables in message patterns to match any message. We formalize security properties as trace properties specified in a fragment of first-order logic. This fragment covers both secrecy and various forms of injective and non-injective authentication properties, e.g., all the properties described in [Low97, CMdV06].

In Chapter 4, we derive and explain the security protocol verification theory that we derive from our model. We illustrate its workings on several manual, but completely formal security proofs. In Chapter 5, we then automate proof construction in this theory, describe the proof-generation algorithm implemented in `scyther-proof`, and report on experimental results. In Chapter 6, we detail our case study on the ISO/IEC 9798 standard. We provide several additional examples that further illustrate the work presented in this part in Appendix A.

Note that all definitions, rules, and theorems in this part of the thesis are justified by corresponding definitions and machine-checked (soundness) proofs in Isabelle/HOL. The corresponding proof scripts are distributed together with the source-code of the `scyther-proof` tool [Mei12].

Part II: Extending the Scope

In Chapter 7, we explain the protocol execution model used in the second part of thesis. We use terms modulo an equational theory to model cryptographic messages. We specify both the actions of the honest protocol participants and the capabilities of the adversary jointly as a labeled multiset-rewriting system. Security properties are trace properties of these systems. They are specified as formulas in two-sorted first-order logic, which allows quantification over both timepoints and messages. We demonstrate the expressivity of this model on two protocols: we formalize security of the NAXOS protocol in the eCK security model [LLM07] and the security of the basic TESLA stream authentication protocol under the assumption of synchronized clocks [PCTS00].

In Chapter 8, we explain the verification theory underlying the Tamarin prover. We develop this theory in four steps. In Section 8.1, we formalize the notion of dependency graphs, which are an alternative representation of the execution of a labeled multiset rewriting system. In Section 8.2, we introduce our basic constraint-reduction rules. They formalize a goal-directed backwards-search for a dependency graph satisfying a given trace property. They are sufficient to formalize correctness arguments that only rely on the non-looping parts of a security protocol. In Section 8.3, we then formalize the use of trace induction to reason about looping parts of a protocol. Unfortunately, trace induction is not sufficient to reason about message deduction. In Section 8.4, we therefore develop specialized constraint-reduction rules for reasoning about message deduction by generalizing the ideas underlying `scyther-proof`'s verification theory. These constraint-reduction rules are based on the idea of searching for *normal form*

dependency graphs, which formalize the notion of efficient adversaries. The concrete definition of normal form dependency graphs depends on the equational theory of interest. We therefore parametrize our rules on the concrete assumptions about normal form dependency graphs to make them more widely applicable. We explain the need for and use of type assertions in Section 8.4.4. They are proven using trace induction. The proofs justifying our verification theory are either given inline or in Appendix B.

In Chapter 9, we explain how we automate the use of our constraint-reduction rules for the construction of protocol security proofs in the Tamarin prover. We discuss both Tamarin's interactive mode and the results that we obtained when analyzing a wide range of case studies using Tamarin's automatic mode.

2. Background

In this chapter, we introduce the notation that we use throughout this thesis and provide background on term rewriting and the Isabelle/HOL theorem prover.

2.1. Notation

Definitions We mark *a notion being defined* using a slanted font. We moreover write $x \coloneqq e$ to emphasize that x is defined by the expression e.

Sets We write $\mathbb{N}$ for the set of natural numbers and $\mathbb{Q}$ for the set of rational numbers. For a set A, we write $\mathcal{P}(A)$ for its power set and $\mathcal{P}_{\mathit{fin}}(A)$ for the set of finite sets with elements from A. We use standard notation for pairs and define the projections $\pi_i(x_1, x_2) \coloneqq x_i$ for $i \in \{1, 2\}$. For a binary relation R, we denote its domain by $\mathit{dom}(R)$, its range by $\mathit{ran}(R)$, its transitive closure by R^+, and its reflexive transitive closure by R^*. Overloading notation, we sometimes write $f(X)$ for the pointwise application of a function $f : A \to B$ to all elements of a set $X \subseteq A$.

Functions Let f be a function. Overloading notation, we write $\mathit{dom}(f)$ and $\mathit{ran}(f)$ to denote f's domain and range, respectively. We write $f[a \mapsto b]$ to denote the update of f at a, defined as the function f' where $f'(x) = b$ when $x = a$ and $f'(x) = f(x)$ otherwise. Analogously, we use $f[x \mapsto g(x)]_{x \in X}$ to denote the update of f at each $x \in X$ with $g(x)$. We write $f : X \rightharpoonup Y$ to denote a partial function mapping all elements in $\mathit{dom}(f) \subseteq X$ to elements from Y. We model partial functions in a logic of total functions by mapping all elements in $X \setminus \mathit{dom}(f)$ to the undefined value $\bot$, different from all other values.

Multisets For a set A, we write $A^\sharp$ for the set of finite multisets of elements from A. We also use the superscript $^\sharp$ to denote the usual operations on multisets. For example, we write $\varnothing^\sharp$ for the empty multiset, $m_1 \cup^\sharp m_2$ for the union of two multisets m_1 and m_2, and $\{a, a, b\}^\sharp$ for the multiset containing a twice and b once. We write $\mathit{set}(m)$ for the set of all elements of a multiset m.

Sequences For a set S, we write S^* to denote the set of finite sequences of elements from S. For a sequence s, we write s_i for the i-th element, $|s|$ for the length of s, and $\mathit{idx}(s) \coloneqq \{1, \ldots, |s|\}$ for the set of indices of s. We write $\vec{s}$ to emphasize that s is a sequence. We use $[]$ to denote the empty sequence, $[s_1, \ldots, s_k]$ to denote the sequence s consisting of the elements s_1 through s_k. We use $s \cdot s'$ to denote the concatenation of the sequences s and s'.

Overloading notation, we write $\mathit{set}(s)$ for the set of all elements of a sequence s and $\mathit{mset}(s)$ for the corresponding multiset. We moreover define that $x \in s$ iff $x \in \mathit{set}(s)$. We

write $x <_s y$ to denote that x *precedes* y in the sequence s, i.e., $\exists a\, b.\, s = a \cdot b \wedge x \in a \wedge y \in b$. Note that $<_s$ is a strict total order on the elements in s iff s is duplicate-free. We say that a sequence s_1 is a *prefix of a sequence* s iff there exists s_2 such that $s = s_1 \cdot s_2$. We say that a set of sequences S is *prefix closed* iff S contains every prefix of every sequence $s \in S$.

Algebraic datatypes We use algebraic datatypes as is standard in functional programming languages like Haskell and theorem provers like Isabelle/HOL. We specify a datatype ty with constructors $\mathsf{C}_i : \alpha_{i,1} \times \ldots \times \alpha_{i,m} \to ty$, where $1 \le i \le n$, as

$$ty ::= \mathsf{C}_1(\alpha_{1,1} \times \ldots \times \alpha_{1,m}) \mid \ldots \mid \mathsf{C}_n(\alpha_{n,1} \times \ldots \times \alpha_{n,m}) \ .$$

Note that ty may also occur as a constructor argument $\alpha_{i,j}$, i.e., the definition of ty may be recursive. Unless stated otherwise, we consider datatype definitions to be inductive, i.e., we consider ty to be the *smallest* set that is closed under the application of the constructors C_i. For a formal treatment of datatypes in the context of higher-order logic, we refer the reader to [BW99].

2.2. Term Rewriting

We recall standard notions from rewriting, following [ESM12].

Order sorted term algebras An *order-sorted signature* $\mathbf{\Sigma} := (S, \le, \Sigma)$ consists of a set of sorts S, a partial order $\le$ on S, and a set of function symbols Σ associated with sorts such that the following two properties are satisfied. First, for every $s \in S$, the connected component C of s in $(S, \le)$ has a *top sort* denoted $top(s)$ such that $c \le top(s)$ for all $c \in C$. Second, for every $f : s_1 \times \ldots \times s_k \to s$ in Σ with $k \ge 1$, $f : top(s_1) \times \ldots \times top(s_k) \to top(s)$ is in Σ.

We assume that there are pairwise disjoint, countably infinite sets of variables $\mathcal{V}_s$ and constants $\mathcal{C}_s$ for each sort $s \in S$. We define the set of all variables as $\mathcal{V} := \bigcup_{s \in S} \mathcal{V}_s$ and the set of all constants as $\mathcal{C} := \bigcup_{s \in S} \mathcal{C}_s$. We use $x{:}s$ to denote variables from $\mathcal{V}_s$. For an arbitrary set $A \subseteq \mathcal{V} \cup \mathcal{C}$ of variables and constants, $\mathcal{T}_{\mathbf{\Sigma}}(A)$ denotes the set of well-sorted terms constructed over $\Sigma \cup A$.

If there is only one sort in S, we say that $\mathbf{\Sigma}$ is *unsorted* and we identify $\mathbf{\Sigma}$ with its set of function symbols Σ. We write Σ^k for the set of all k-ary function symbols in Σ.

Positions, subterms, and variables of a term A *position* p is a sequence of natural numbers. For a term t, the *subterm of* t *at position* p, written $t|_p$, is defined as

$$t|_p := \begin{cases} t & \text{if } p = [\,] \\ t_i|_{p'} & \text{if } p = [i] \cdot p' \text{ and } t = f(t_1, \ldots, t_k) \text{ and } 1 \le i \le k \\ \bot & \text{otherwise} \end{cases} \ .$$

We say that p is a *valid position in* t iff $t|_p$ is defined. We define the set of *subterms of* t as $St(t) := \{t|_p \mid p \text{ a valid position in } t\}$. A *proper subterm of* t is a subterm $s \in St(t) \setminus \{t\}$. We define the set of *variables of* t as $vars(t) := St(t) \cap \mathcal{V}$. A term is *ground* iff $vars(t) = \varnothing$.

We use $t[s]_p$ to denote the term t where the subterm of t at position p has been replaced with the term s.

Substitutions A *substitution* σ is a well-sorted function from $\mathcal{V}$ to the set of terms $\mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V} \cup \mathcal{C})$ that corresponds to the identity function on all variables except on a finite set of variables. Overloading notation, we call this finite set of variables the *domain* of σ, written $dom(\sigma)$. We use $range(\sigma)$ to denote the image of $dom(\sigma)$ under σ and define $vrange(\sigma) := \bigcup_{t \in range(\sigma)} vars(t)$. We identify σ with its usual extension to an endomorphism on $\mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V} \cup \mathcal{C})$ and use the notation $t\sigma$ for the application of this extension of σ to a term t.

Equations and equational theories An *equation* over the signature $\mathbf{\Sigma}$ is an unordered pair $\{s, t\}$ of terms $t, s \in \mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V})$, written $s \simeq t$. An *equational presentation* is a tuple $\mathcal{E} = (\mathbf{\Sigma}, E)$ of a signature $\mathbf{\Sigma}$ and a set of equations E. The corresponding *equational theory* $=_{\mathcal{E}}$ is the smallest Σ-congruence containing all instances of equations in E. We often leave the signature $\mathbf{\Sigma}$ implicit and identify the equations E with the equational presentation $\mathcal{E}$ and the equational theory $=_E$, i.e., $=_{\mathcal{E}}$.

We say that two terms t and s are *equal modulo* E iff $t =_E s$. We use the subscript $_E$ to denote the usual operations on sets, sequences, and multisets adapted to use equality modulo E instead of syntactic equality. For example, we use $\in_E$ to denote set membership modulo E.

Unification and matching An *E-unifier* of two terms s and t is a substitution σ such that $s\sigma =_E t\sigma$. For $W \subseteq \mathcal{V}$, we use $unify_E^W(s, t)$ to denote an E-unification algorithm that returns a set of E-unifiers of s and t away from W, i.e., $vrange(\sigma) \cap W = \varnothing$ for all $\sigma \in unify_E^W(s, t)$. We say that this unification algorithm is *complete* iff, for all terms s and t and all E-unifiers τ of s and t, there is a $\sigma \in unify_E^W(s, t)$ and a substitution α such that $x\tau =_E (x\sigma)\alpha$ for all $x \in vars(s, t)$. We say that this algorithm is *finitary* iff it terminates for all inputs and returns a finite set of unifiers.

An *E-matcher* matching a term t to a term s is a substitution σ such that $t\sigma =_E s$. The notions of complete and finitary coincide with those for E-unifiers.

Rewriting A *rewrite rule* is an ordered pair of terms (l, r) with $l, r \in \mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V})$, written $l \to r$. A *rewrite system* $\mathcal{R}$ is a set of rewrite rules. The corresponding *rewrite relation* $\to_{\mathcal{R}}$ is defined such that $s \to_{\mathcal{R}} t$ iff there is a position p in s, a rewrite rule $l \to r \in \mathcal{R}$, and a substitution σ such that $s|_p = l\sigma$ and $s[r\sigma]_p = t$. A rewrite system $\mathcal{R}$ is *terminating* iff there is no infinite sequence $(t_i)_{i \in \mathbb{N}}$ of terms with $t_i \to_{\mathcal{R}} t_{i+1}$ for all $i \in \mathbb{N}$. A rewrite system $\mathcal{R}$ is *confluent* iff, for all terms t, s_1, and s_2 with $t \to_{\mathcal{R}}^* s_1$ and $t \to_{\mathcal{R}}^* s_2$, there is a term t' such that $s_1 \to_{\mathcal{R}}^* t'$ and $s_2 \to_{\mathcal{R}}^* t'$. A rewrite system $\mathcal{R}$ is *convergent* iff it is terminating and confluent. In this case, we use $t{\downarrow_{\mathcal{R}}}$ to denote the unique *$\mathcal{R}$-normal form of* t, i.e., $t{\downarrow_{\mathcal{R}}}$ is the unique term t' such that $t \to_{\mathcal{R}}^* t'$ and there is no term t'' with $t' \to_{\mathcal{R}} t''$.

A rewrite system $\mathcal{R}$ is *subterm-convergent* iff it is convergent and, for each rule $l \to r \in \mathcal{R}$, r is either a proper subterm of l or r is ground and in $\mathcal{R}$-normal form.

Rewriting modulo [Vir02] Rewriting is often used to reason about equality modulo an equational theory E. If the equations in E can be oriented to obtain a convergent rewriting system $\mathcal{R}$, then $\mathcal{R}$ can be used to decide equality since $t =_E s$ iff $t{\downarrow_{\mathcal{R}}} = s{\downarrow_{\mathcal{R}}}$.

The notion of $\mathcal{R}$,$\mathcal{AX}$-*rewriting* for a rewrite system $\mathcal{R}$ and an equational theory $\mathcal{AX}$ generalizes this approach to equational theories containing some equations that cannot be oriented, e.g., commutativity. The *rewrite relation* $\rightarrow_{\mathcal{R},\mathcal{AX}}$ is defined such that $s \rightarrow_{\mathcal{R},\mathcal{AX}} t$ iff there is a position p in s, a rewrite rule $l \rightarrow r \in \mathcal{R}$, and a substitution σ such that $s|_p =_{\mathcal{AX}} l\sigma$ and $s[r\sigma]_p = t$. Note that the relation $\rightarrow_{\mathcal{R},\mathcal{AX}}$ is decidable, if $\mathcal{AX}$-matching is decidable. Analogous to rewrite systems, we say that $\mathcal{R}$ is $\mathcal{AX}$-*convergent* iff the relation $\rightarrow_{\mathcal{R},\mathcal{AX}}$ is terminating and confluent. In this case, we use $t{\downarrow}^{\mathcal{R}}_{\mathcal{AX}}$ to denote the corresponding unique $\mathcal{R}$,$\mathcal{AX}$-*normal form of* t. We say that $\mathcal{R}$ is $\mathcal{AX}$-*coherent* iff, for all terms t_1, t_2, and t_3, the assumptions $t_1 \rightarrow_{\mathcal{R},\mathcal{AX}} t_2$ and $t_1 =_{\mathcal{AX}} t_3$ imply that there are terms t_4 and t_5 with $t_4 =_{\mathcal{AX}} t_5$ such that $t_2 \rightarrow^*_{\mathcal{R},\mathcal{AX}} t_4$ and $t_3 \rightarrow^+_{\mathcal{R},\mathcal{AX}} t_5$. We define $\mathcal{R}^{\simeq} := \{l \simeq r \mid l \rightarrow r \in \mathcal{R}\}$. If $(\mathbf{\Sigma}, \mathcal{R}^{\simeq} \cup \mathcal{AX})$ is an equational presentation of $=_E$ and $\mathcal{R}$ is both $\mathcal{AX}$-convergent and $\mathcal{AX}$-coherent, then $t =_E s$ iff $t{\downarrow}^{\mathcal{R}}_{\mathcal{AX}} =_{\mathcal{AX}} s{\downarrow}^{\mathcal{R}}_{\mathcal{AX}}$.

2.3. Isabelle/HOL

Isabelle is a generic, tactic-based theorem prover. We use Isabelle's implementation of higher-order logic, Isabelle/HOL [NPW02], for the development of the work presented in Part I of this thesis.

We formalize the protocol semantics underlying Part I as a conservative definitional extension of higher-order logic (HOL). This guarantees the consistency of our formalization, provided that HOL itself is consistent. Based on our semantics, we formalize several domain-specific predicates that serve as the basic building blocks for expressing security properties as HOL formulas. Such a formalization is sometimes called a *shallow embedding* [Gor89] in the verification community, as we do not introduce a separate concrete syntax for formalizing security properties, but work directly with the formulas formalizing their semantics. The benefit of a shallow embedding is that we can reuse the existing reasoning infrastructure for HOL and only need to extend it to reason about our domain-specific predicates. We extend the existing infrastructure by deriving inference rules (HOL theorems) from our semantics that encode reasoning principles for constructing protocol security proofs.

We emphasize that all definitions, rules, and theorems in the first part of this thesis are justified by corresponding definitions and machine-checked (soundness) proofs in Isabelle/HOL. The corresponding proof scripts are distributed together with the source code of the `scyther-proof` tool [Mei12].

Part I.

Improving the Trustworthiness

3. Security Protocol Model

In this chapter, we define our security protocol model. It consists of three parts: (1) a *protocol specification language* based on role scripts (sequences of send and receive steps) and pattern matching, (2) an operational semantics defining *protocol execution* in the presence of an active adversary, and (3) a collection of predicates for formalizing *security properties* like secrecy and authentication.

3.1. Protocol Specification

We model security protocols as sets of roles where each role is given by a role script specified by a sequence of role steps. A role step sends or receives messages matching given message patterns. We first describe the elements of our specification language and then provide an example.

Let *Const* be a set of *constants*, *Fresh* be a set of messages to be freshly generated (nonces, coin flips, etc.), and *Var* be a set of *variables*. We assume that *Const*, *Fresh*, and *Var* are pairwise-disjoint. We further assume that the set of variables *Var* is partitioned into two sets, *AVar* and *MVar*, denoting *agent variables* and *message variables*. Agent variables are placeholders for agent names, which are chosen when creating a new role instance, and message variables are placeholders for messages (which may also be agent names) received during the execution of a role instance. We define the set *Pat* of *message patterns* as

$$\begin{aligned} Pat ::= {} & Const \mid Fresh \mid Var \mid \mathsf{h}(Pat) \mid \langle Pat, Pat \rangle \\ & \mid \{\!|\, Pat \,|\!\}_{Pat} \mid Pat^{-1} \mid \mathsf{k}(Pat, Pat) \mid \mathsf{k}\{Pat, Pat\} \mid \mathsf{pk}(Pat) \mid \mathsf{sk}(Pat) \ . \end{aligned}$$

We define $vars(pt)$ as the set of all variables in the pattern pt.

Intuitively, message patterns denote instructions on how to compose and decompose messages, which are sent to and received from the network. Note that this choice of using the same constructors to specify both message composition and decomposition is common in literature, but unnecessarily complicates the interpretation of message patterns. In Part II, we use a simpler and more direct model that represents each call to a cryptographic algorithm using *exactly* one constructor. In this model, message decomposition can be made explicit by specifying all the required calls to algorithms like decryption and projection.

The execution of the instructions denoted by a message pattern may fail and we assume that the execution of a role stops upon such a failure. This is the reason for the partiality of the *inst* function, which is given in Section 3.2.3 and defines the formal semantics of message patterns. Informally, the semantics of message patterns is as follows.

A pattern $c \in \mathit{Const}$ denotes a constant bitstring. A pattern $n \in \mathit{FN}$ denotes the result of a call to a random number generator. An agent variable $v \in \mathit{AVar}$ denotes an agent name, chosen by the protocol participant executing a role. A message variable $v \in \mathit{MVar}$ denotes a part of a message, which was extracted from a message received from the network. A pattern $\mathsf{h}(pt)$ denotes a call to a hash function using the bitstring denoted by pt as the argument. Analogously, the pattern $\langle pt_1, pt_2 \rangle$ denotes the tuple of the bitstrings denoted by pt_1 and pt_2. We use the pattern constructor $\{\!|\_|\!\}_\_$ to model the use of algorithms for public-key and symmetric encryption, private-key and symmetric decryption, signing, and signature verification. The pattern $\{\!|p|\!\}_k$ denotes public-key encryption of p when $k = \mathsf{pk}(pt)$, signing p when $k = \mathsf{sk}(pt)$, and symmetric encryption of p otherwise. Note that this allows for a composed message such as the hash of some keying material to be used as a symmetric key.

In setups where symmetric long-term keys are used, it is common that if Alice wants to communicate with Bob, she will use their shared long-term key, which is the same key that Bob would use to communicate with Alice. Such keys are called *bidirectional*. Alternatively one can use *unidirectional* keys where each pair of agents shares two symmetric keys, one for each direction. We use the pattern $\mathsf{k}(a, b)$ to denote the unidirectional long-term symmetric key used by the agent denoted by a to communicate with the agent denoted by b. We use the pattern $\mathsf{k}\{a, b\}$ to denote the bidirectional long-term symmetric key shared between a and b. For $a \in \mathit{AVar}$, $\mathsf{pk}(a)$ denotes a's long-term public key and $\mathsf{sk}(a)$ denotes a's long-term private key. We allow for freshly generated asymmetric keypairs by letting $n \in \mathit{Fresh}$ be the random seed, and using $\mathsf{pk}(n)$ and $\mathsf{sk}(n)$ to denote the corresponding freshly generated public and private keys. The pattern pt^{-1} denotes the inverse of the key denoted by the pattern pt.

Intuitively, the patterns for long-term keys denote to the use of key-lookup algorithms. For example, one possible implementation of the pattern $\mathsf{pk}(a)$ is to lookup the public key associated to the bitstring denoted by a in the table of pre-shared keys. For a bidirectional key like $\mathsf{k}\{a, b\}$, this lookup would be performed using the *sorted* tuple of the bitstrings denoted by a and b as the lookup-key, whereas for an unidirectional key one would use the standard, unsorted tuple as the lookup-key.

Let *Label* be a set of labels. We define the set *RoleStep* of *role steps* as

$$\mathit{RoleStep} ::= \mathsf{Send}_{\mathit{Label}}(\mathit{Pat}) \mid \mathsf{Recv}_{\mathit{Label}}(\mathit{Pat}) \ .$$

A *send role-step* $\mathsf{Send}_l(pt)$ denotes sending the message corresponding to the instantiated pattern pt. A *receive role-step* $\mathsf{Recv}_l(pt)$ denotes receiving a message matching the pattern pt. The labels have no operational meaning: they serve just to distinguish different send (or receive) steps that contain the same message pattern. A *role* is a duplicate-free, finite sequence R of role steps such that

$$\begin{aligned} &\forall\ \mathsf{Send}_l(pt) \in R.\ \forall v \in \mathit{vars}(pt) \cap \mathit{MVar}. \\ &\quad \exists l', pt'.\ \mathsf{Recv}_{l'}(pt') <_R \mathsf{Send}_l(pt) \wedge v \in \mathit{vars}(pt') \ . \end{aligned}$$

This states that every message variable in a role must be instantiated in a receive step before its contents can be used in a send step. Note the we only restrict the use of message variables. Agent variables and freshly generated messages can be used freely in send steps. We denote the set of all roles by *Role*.

A *protocol* is a set of roles. We denote the set of all protocols by *Protocol*. We illustrate protocol specifications with a simple challenge-response protocol.

Example 1 (*CR* Protocol)**.** Let $s \in AVar$, $k \in Fresh$, and $v \in MVar$. We define $CR := \{C, S\}$, where

$$\begin{aligned} C &:= [\, \mathsf{Send}_1(\{\!|\, k \,|\!\}_{\mathsf{pk}(s)}),\quad \mathsf{Recv}_2(\mathsf{h}(k)) \,] \\ S &:= [\, \mathsf{Recv}_1(\{\!|\, v \,|\!\}_{\mathsf{pk}(s)}),\quad \mathsf{Send}_2(\mathsf{h}(v)) \,] \,. \end{aligned}$$

In this protocol, a client, modeled by the C role, chooses a fresh session key k and sends it encrypted with the public key $\mathsf{pk}(s)$ of the server with whom he wants to share k. The server, modeled by the S role, confirms the receipt of k by returning its hash. We use this protocol as a running example. Hence, in subsequent examples, the expressions s, k, v, C, S, and CR refer to those introduced here. ♠

3.2. Protocol Execution

During the execution of a protocol P, agents may execute any number of instances of P's roles in parallel. We call each role instance a *thread*. Threads may generate fresh messages, send messages to the network, and receive messages from the network as specified by the role script they execute. We assume that the network is completely controlled by an active Dolev-Yao style adversary. In particular, the adversary learns every message sent and can block and insert messages. Moreover, the adversary can access the long-term keys of arbitrarily many compromised agents.

We provide an operational semantics for protocol execution in the presence of the adversary, expressed as a state transition system, along the lines of [CM05]. The ingredients of the operational semantics are messages, the system state, agent threads, the adversary's knowledge, and the transition system. We discuss each of these in turn.

3.2.1. Messages

We assume an infinite set *TID* of thread identifiers. We use the thread identifiers to distinguish between fresh messages generated by different threads. For a thread identifier i and a message $n \in Fresh$ to be freshly generated, we write $n\sharp i$ to denote the fresh message generated by the thread i for n. We overload notation and for A a set, we write $A\sharp\, TID$ to denote the set $\{a\sharp i \mid a \in A, i \in TID\}$.

We assume given a set *Agent* of agent names disjoint from *Const*. We define the set *Msg* of *messages* as

$$\begin{aligned} Msg ::= {} & Const \mid Fresh\sharp\, TID \mid Agent \mid \mathsf{h}(Msg) \mid (Msg, Msg) \\ & \mid \{\!|\, Msg \,|\!\}_{Msg} \mid \mathsf{k_{un}}(Agent, Agent) \mid \mathsf{k_{bi}}(\mathcal{P}(Agent)) \mid \mathsf{pk}(Msg) \mid \mathsf{sk}(Msg) \,. \end{aligned}$$

We assume the existence of an inverse function on messages, where k^{-1} denotes the inverse key of k. We have $\mathsf{pk}(x)^{-1} = \mathsf{sk}(x)$ and $\mathsf{sk}(x)^{-1} = \mathsf{pk}(x)$ for every message x, and $m^{-1} = m$ for all other messages m. Thus, depending on the key k, the message $\{\!|\, m \,|\!\}_k$ denotes the result of signing, public-key encryption, or symmetric encryption. We model a bidirectional key as a special case of a key shared between a set of agents; i.e., we use $\mathsf{k_{bi}}(\{a, b\})$ to denote the result of looking up the bidirectional pre-shared

symmetric key of the two agents a and b. We use $\mathsf{k_{un}}(a, b)$ to denote the result of looking up the unidirectional pre-shared symmetric key of the two agents a and b. Note that $\mathsf{k_{bi}}(A) \neq \mathsf{k_{un}}(a, b)$ for all $A \subseteq \mathit{Agent}$ and $a, b \in \mathit{Agent}$; i.e., we assume that all pre-shared bidirectional keys are different from the pre-shared unidirectional keys.

3.2.2. System State

The system state of our operational semantics is a triple (tr, th, σ). It consists of (1) a trace tr recording the history of the executed role steps and the events when messages are learned (i.e., become known for the first time) by the adversary, (2) a thread pool th that stores for every thread the role it executes and the role steps still to be executed, and (3) a variable store σ recording the variable contents of all threads. We define these parts below.

The set of all *trace events* is defined as

$$\mathit{TraceEvent} ::= \mathsf{St}(\mathit{TID}, \mathit{RoleStep}) \mid \mathsf{Ln}(\mathcal{P}(\mathit{Msg})) \ .$$

A *step trace event* $\mathsf{St}(i, s)$ denotes that the thread i executed the role step s. A *learn trace event* $\mathsf{Ln}(M)$ denotes that the adversary learned the set of messages M. We use learn trace events to explicitly record the adversary's message deduction steps in the trace. We will see why we use sets of messages rather than single messages later in the definition of our operational semantics. The *trace* tr is a sequence of *trace events*. The *thread pool* th is a partial function

$$th : \mathit{TID} \nrightarrow (\mathit{Role} \times \mathit{RoleStep}^*) \ ,$$

where $\mathit{dom}(th)$ denotes the identifiers of all threads in the system. Where unambiguous, we identify threads with their corresponding thread identifiers. For $i \in \mathit{dom}(th)$ and $th(i) = (R, \mathit{todo})$, R is the role executed by thread i and todo is a suffix of R denoting the role steps still to be executed by thread i. The *variable store* σ is a function

$$\sigma : \mathit{Var} \times \mathit{TID} \to \mathit{Msg}$$

that stores for each variable v and thread identifier i the contents $\sigma(v, i)$ assigned to v by thread i. We define *Trace* as the set of all traces, *ThreadPool* as the set of all thread pools, and *Store* as the set of all variable stores. Hence, a *system state* is a triple

$$(tr, th, \sigma) \in \mathit{Trace} \times \mathit{ThreadPool} \times \mathit{Store} \ .$$

Note that we explicitly record the history of the protocol execution and the message deduction of the adversary in the trace. As we will see later, this is crucial for formulating our invariants. It allows us, for example, to formulate the statement that if the adversary learns a message m by decrypting an encryption $\{\!|\, m \,|\!\}_k$, then he must have learned the inverse key k^{-1} before m.

3.2.3. Agent Threads

Let (tr, th, σ) be a system state and $i \in \mathit{dom}(th)$ be a thread in the system. The *role of thread* i is defined as $\mathit{role}_{th}(i) := \pi_1(th(i))$. Moreover, the thread i *instantiates* a

$$inst_{\sigma,i}(pt) := \begin{cases} pt & \text{if } pt \in Const \\ pt\sharp i & \text{if } pt \in Fresh \\ \sigma(pt,i) & \text{if } pt \in Var \\ \mathsf{h}(inst_{\sigma,i}(x)) & \text{if } pt = \mathsf{h}(x) \\ (inst_{\sigma,i}(x), inst_{\sigma,i}(y)) & \text{if } pt = (x,y) \\ \{\!|\, inst_{\sigma,i}(x) \,|\!\}_{(inst_{\sigma,i}(k))} & \text{if } pt = \{\!|\, x \,|\!\}_k \\ (inst_{\sigma,i}(x))^{-1} & \text{if } pt = x^{-1} \\ \mathsf{pk}(inst_{\sigma,i}(a)) & \text{if } pt = \mathsf{pk}(a) \\ \mathsf{sk}(inst_{\sigma,i}(a)) & \text{if } pt = \mathsf{sk}(a) \\ \mathsf{k_{un}}(inst_{\sigma,i}(a)\,,\, inst_{\sigma,i}(b)) & \text{if } pt = \mathsf{k}(a,b) \\ & \quad \text{and } inst_{\sigma,i}(a), inst_{\sigma,i}(b) \in Agent \\ \mathsf{k_{bi}}(\{inst_{\sigma,i}(a)\,,\, inst_{\sigma,i}(b)\}) & \text{if } pt = \mathsf{k}\{a,b\} \\ & \quad \text{and } inst_{\sigma,i}(a), inst_{\sigma,i}(b) \in Agent \\ \bot & \text{otherwise} \end{cases}$$

Figure 3.1.: Definition of the partial message pattern instantiation function $inst_{\sigma,i}(\_)$

message pattern pt occurring in its role to the message $inst_{\sigma,i}(pt)$ as defined in Figure 3.1. During instantiation fresh message patterns are replaced with the actual fresh messages and all variables are replaced with the content assigned to them in thread i. Moreover, the key-inverse constructor on patterns is replaced with applications of the key-inverse function on messages. This is well-defined because messages are ground by definition. Note that looking up a symmetric long-term key may fail, which is why *inst* is a partial function.

3.2.4. Adversary Knowledge

We assume that the set of all agents *Agent* is partitioned into a set *Compr* of compromised agents whose long-term keys are known to the adversary and a set of uncompromised agents. Thus, the adversary can impersonate any agent $c \in Compr$ and act as a legitimate protocol participant. The *initial knowledge of the adversary* AK_0 is therefore defined as

$$AK_0 := Const \cup Agent \cup \bigcup_{a \in Agent, c \in Compr} \{\mathsf{pk}(a), \mathsf{sk}(c), \mathsf{k_{un}}(a,c), \mathsf{k_{un}}(c,a), \mathsf{k_{bi}}(\{a,c\})\} \ .$$

Note that we will define our operational semantics such that every trace starts with the learn trace event $\mathsf{Ln}(AK_0)$; i.e., the adversary always learns his complete initial knowledge before any other event happens. Our semantics also forces the adversary to explicitly perform all message deduction steps. Hence, a trace tr records all messages learned by the adversary. The *adversary's knowledge* corresponding to tr is therefore

$$knows(tr) := \bigcup_{\mathsf{Ln}(M) \in tr} M \ .$$

Our operational semantics also ensures that whenever the adversary learns a pair of messages $\langle m_1, m_2 \rangle$, then he also learns its projections m_1 and m_2, provided he does not

know them already. Therefore, the adversary's knowledge is closed under the projection of pairs. This simplifies the reasoning in our security proofs as it allows keeping the construction and projection of pairs implicit. We use the function $split : Msg \to \mathcal{P}(Msg)$ to formalize this closure.

$$split(m) := \begin{cases} \{m\} \cup split(x) \cup split(y) & \text{if } m = \langle x, y \rangle \\ \{m\} & \text{otherwise} \end{cases}$$

We define the set of all messages *learned* by the adversary from a message m in the context of a trace tr as

$$learns_{tr}(m) := split(m) \setminus knows(tr) \ .$$

This states that the adversary learns all messages in $split(m)$ that he did not already know.

Example 2 (System state of the *CR* protocol)**.** Assume that some agent $a \in Agent$ executes the C role in thread $i \in TID$ and has sent his first message $\{\!|\, k \sharp i \,|\!\}_{\mathsf{pk}(b)}$ to establish the fresh session key $k \sharp i$ with an agent $b \in Agent$. Also, assume that agent b executed the $\mathsf{Recv}_1(\{\!|\, v \,|\!\}_{\mathsf{pk}(s)})$ step of the S role in the thread $j \in TID$ and received the first message that thread i sent. If i and j are the only threads running, then the system state is of the form (tr, th, σ), for

$$\begin{aligned} th &:= \{i \mapsto (C, [C_2]),\ j \mapsto (S, [S_2])\} \\ \sigma &:= \sigma'[(s, i) \mapsto b][(s, j) \mapsto b][(v, j) \mapsto k \sharp i] \\ tr &:= [\mathsf{Ln}(AK_0),\ \mathsf{St}(i, C_1),\ \mathsf{Ln}(\{\{\!|\, k \sharp i \,|\!\}_{\mathsf{pk}(b)}\}),\ \mathsf{St}(j, S_1)] \end{aligned}$$

and some $\sigma' \in Store$. ♠

3.2.5. Transition System

The *state transition relation* $\longrightarrow$ formalizing our operational semantics is defined by the transition rules in Figure 3.2. We explain each rule in turn.

A SEND transition is enabled whenever the next step of a thread i is $\mathsf{Send}_l(pt)$ for some label l and message pattern pt whose instantiation does not fail. The trace tr is extended with two trace events. The trace event $\mathsf{St}(i, \mathsf{Send}_l(pt))$ records that the send step has happened. The trace event $\mathsf{Ln}(learns_{tr}(inst_{\sigma,i}(pt)))$ records the set of messages learned by the adversary from the sent message $inst_{\sigma,i}(pt)$. We see here the benefit of being able to record that the adversary learns a set of messages at once. Namely, we can close the adversary's knowledge under projection of pairs without explicitly ordering the events denoting the learning of the pairs' components. Note that $\mathsf{Ln}(learns_{tr}(inst_{\sigma,i}(pt)))$ is equal to $\mathsf{Ln}(\varnothing)$ if and only if the adversary has already learned the sent message $inst_{\sigma,i}(pt)$ before this SEND transition.

A RECV transition is enabled whenever the next step of a thread i is $\mathsf{Recv}_l(pt)$, for some label l and some message pattern pt, and the adversary knows a message matching pt under the variable store σ. The trace tr is extended with the trace event $\mathsf{St}(i, \mathsf{Recv}_l(pt))$, recording that this receive step has happened.

$$\frac{th(i) = (R, [\mathsf{Send}_l(pt)] \cdot todo) \qquad inst_{\sigma,i}(pt) \neq \bot}{(tr, th, \sigma) \longrightarrow (tr \cdot [\, \mathsf{St}(i, \mathsf{Send}_l(pt)),\, \mathsf{Ln}(learns_{tr}(inst_{\sigma,i}(pt)))\,],\; th[i \mapsto (R, todo)],\; \sigma)} \textsc{Send}$$

$$\frac{th(i) = (R, [\mathsf{Recv}_l(pt)] \cdot todo) \qquad inst_{\sigma,i}(pt) \in knows(tr) \qquad inst_{\sigma,i}(pt) \neq \bot}{(tr, th, \sigma) \longrightarrow (tr \cdot [\, \mathsf{St}(i, \mathsf{Recv}_l(pt))\,],\; th[i \mapsto (R, todo)],\; \sigma)} \textsc{Recv}$$

$$\frac{x, y \in knows(tr) \qquad \langle x, y \rangle \notin knows(tr)}{(tr, th, \sigma) \longrightarrow (tr \cdot [\, \mathsf{Ln}(\{\langle x, y \rangle\})\,], th, \sigma)} \textsc{Pair}$$

$$\frac{m \in knows(tr) \qquad \mathsf{h}(m) \notin knows(tr)}{(tr, th, \sigma) \longrightarrow (tr \cdot [\, \mathsf{Ln}(\{\mathsf{h}(m)\})\,], th, \sigma)} \textsc{Hash}$$

$$\frac{m, k \in knows(tr) \qquad \{\!|\, m \,|\!\}_k \notin knows(tr)}{(tr, th, \sigma) \longrightarrow (tr \cdot [\, \mathsf{Ln}(\{\{\!|\, m \,|\!\}_k\})\,], th, \sigma)} \textsc{Encr}$$

$$\frac{\{\!|\, m \,|\!\}_k \in knows(tr) \qquad k^{-1} \in knows(tr)}{(tr, th, \sigma) \longrightarrow (tr \cdot [\, \mathsf{Ln}(learns_{tr}(m))\,], th, \sigma)} \textsc{Decr}$$

Figure 3.2.: Transition rules of the execution model

A Pair, Hash, or Encr transition models the adversary learning respectively a pair, a hash, or an encryption by constructing it by himself. He can do this provided he does not yet know the constructed message. Because the adversary's knowledge is closed under *split*, no projection transition is needed.

A Decr transition models the decryption of an encrypted message with the decryption key and learning all new messages accessible from the encrypted message using projection of pairs.

There is no explicit transition rule for creating new threads. Instead, for each thread pool that can occur in an execution, we construct a separate initial state. For a protocol P, all possible thread pools with threads executing roles of P are represented in the set of *initial states* $Q_0(P)$ of our system.

$$Q_0(P) := \{\, ([\mathsf{Ln}(AK_0)],\; th,\; \sigma) \mid (\forall v \in AVar,\, i \in TID.\; \sigma(v, i) \in Agent) \wedge (\forall i \in dom(th).\; \exists R \in P.\; th(i) = (R, R)) \,\}$$

For each initial state $(tr, th, \sigma) \in Q_0(P)$, the variable store σ is defined such that every agent variable is instantiated with an agent name and each message variable is instantiated with an arbitrary message. Thus, we model the set of possible executions by instantiating all variables non-deterministically at the beginning of a thread. The thread pool th is defined such that every thread $i \in dom(th)$ instantiates a role of P and has not executed any step yet.

For a protocol P, we define the set of *reachable states* as

$$reachable(P) := \{q \mid \exists q_0 \in Q_0(P).\; q_0 \longrightarrow^* q\}\ .$$

3.3. Security Properties

We focus on security properties that are state properties. We say that a security protocol P *satisfies* a security property φ if and only if φ holds for every reachable state of P. Because we include the execution trace in the system state, many security properties from literature can be expressed as state properties. We illustrate this on examples later in this section.

Note that when reasoning about security protocols and their properties, we are interested in when individual messages were learned and not in what messages were learned at the same time. Therefore, we introduce the set of *(proper) events*.

$$\mathit{Event} ::= \mathsf{ST}(\mathit{TID} \times \mathit{RoleStep}) \mid \mathsf{LN}(\mathit{Msg})$$

Analogous to step trace events, a *step event* $\mathsf{ST}(i, s)$ denotes that thread i has executed the role step s. In contrast to learn trace events, a *learn event* $\mathsf{LN}(m)$ denotes that the adversary learned the *single* message m.

In the remainder of this paper, we simplify our notation and definitions by identifying every tuple $(i, s) \in \mathit{TID} \times \mathit{RoleStep}$ with the step event $\mathsf{ST}(i, s)$ and every message $m \in \mathit{Msg}$ with the learn event $\mathsf{LN}(m)$. The previously defined function $\mathit{knows} : \mathit{Trace} \to \mathcal{P}(\mathit{Msg})$ can therefore be used to project a trace tr to the set of all learn events occurring in tr.

The projection of a trace tr to the set of all step events occurring in tr is

$$\mathit{steps}(tr) := \{(i, s) \mid \mathsf{St}(i, s) \in tr\} \ .$$

The *event order* relation $(\prec_{tr}) \subseteq \mathit{Event} \times \mathit{Event}$ denotes the order of events induced by the trace events in tr.

$$\begin{aligned} x \prec_{tr} y := \exists tr_1\ tr_2.\ tr = tr_1 \cdot tr_2 \wedge (x \in \mathit{knows}(tr_1) \vee x \in \mathit{steps}(tr_1)) \\ \wedge (y \in \mathit{knows}(tr_2) \vee y \in \mathit{steps}(tr_2)) \end{aligned}$$

Note that $\prec_{tr}$ is a strict partial order on Event for every trace tr of a reachable state $(tr, th, \sigma) \in \mathit{reachable}(P)$. We define $\preceq_{tr}$ as the reflexive closure of $\prec_{tr}$.

The event order allows us, for example, to formalize the statement "both the encryption $\{\!|\, m \,|\!\}_k$ and the inverse key k^{-1} must have been learned before the adversary learned m" as the proposition

$$\{\!|\, m \,|\!\}_k \prec_{tr} m \ \wedge\ k^{-1} \prec_{tr} m \ .$$

Note that the event order also relates learn events with protocol step events. We exploit this, for example, when stating and verifying temporal secrecy properties, i.e., properties stating that a message is secret as long as a certain protocol step, which leaks this message, has not been executed.

Security properties are formalized as logical formulas built using the previously defined functions and relations. We illustrate this in the following example. Afterwards, we explain how standard secrecy and authentication properties from literature are formalized in our model.

Example 3 (Security properties of the *CR* protocol)**.** For a client who completes its role with an uncompromised server, the *CR* protocol guarantees (1) the secrecy of the

session key k and (2) the existence of a server that answered this client and agrees on all exchanged data. We formalize property (1) by the formula φ_{sec}.

$$\varphi_{\text{sec}}(tr, th, \sigma) := \forall i \in TID.\ role_{th}(i) = C \wedge \sigma(s, i) \notin Compr \Rightarrow k \sharp i \notin knows(tr)$$

Recall that C, s, and k, as well as S and v, were defined in Example 1.

Property (2) is an authentication property, which characterizes the guarantees provided to a client after completing its role with an uncompromised server. Inspired by non-injective synchronization [CMdV06] a stronger version of non-injective agreement [Low97], we formalize it as follows.

$$\begin{aligned}\varphi_{\text{auth}}(tr, th, \sigma) := \forall i \in TID.\ & \\ & role_{th}(i) = C \wedge \sigma(s, i) \notin Compr \wedge (i, C_2) \in steps(tr) \\ & \Rightarrow (\exists j \in TID.\ role_{th}(j) = S \wedge \sigma(s, i) = \sigma(s, j) \wedge k \sharp i = \sigma(v, j) \wedge \\ & \qquad (i, C_1) \prec_{tr} (j, S_1) \wedge (j, S_2) \prec_{tr} (i, C_2))\end{aligned}$$

Recall that a role is a sequence of role-steps. Hence, C_2 denotes the second step of the C role. ♠

3.3.1. Secrecy Properties

Secrecy properties are statements that specify under what conditions a message is guaranteed to be unknown to the adversary. In general, we formalize them as closed formulas of the form

$$\forall q \in reachable(P).\ \forall i \in TID.\ \forall m \in Msg.\ claim_q(i, m) \Rightarrow m \notin knows(tr)\ .$$

Here, $claim_q(i, m)$ is a predicate formalizing for every state q when a thread i claims that a message m is secret.

Note that we cannot formalize perfect forward secrecy [BC10] in our model, as we only model a set of statically compromised agents. The protocol model presented in Part II does not suffer from this limitation. Moreover in [Sch11], Schaub extends the model presented here with support for dynamically compromising adversaries [BC10] and demonstrates that the proof construction technique explained in the following chapter also works in this extended model.

3.3.2. Authentication Properties

Authentication properties are statements about the (apparent) partners of a protocol participant who completed his role. Lowe's hierarchy of authentication properties [Low97] provides a good overview of the properties one typically expects a security protocol to achieve. We recall them below. Afterwards, we explain how we formalize them in our model.

> **Aliveness** [Low97]: A protocol guarantees to an initiator A *aliveness* of another agent B if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol.

Aliveness is the weakest of the properties defined by Lowe. It does not even require the responder to be aware that A is trying to authenticate him. It also does not guarantee that the participants agree on the data exchanged during the run of the protocol. This additional requirement is captured by non-injective agreement, which implies aliveness.

> **Non-injective agreement** [Low97]: A protocol guarantees to an initiator A *non-injective agreement* with a responder B on a set of data items ds (where ds is a set of terms appearing in the protocol description) if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the terms in ds.

The general principle behind aliveness and non-injective agreement is that of a *non-injective two-party authentication* property. We formalize such authentication properties as closed formulas of the form

$$\forall q \in \mathit{reachable}(P).\ \forall i \in \mathit{TID}.\ \mathit{claim}_q(i) \Rightarrow \exists j \in \mathit{TID}.\ \mathit{partner}_q(i,j)\ .$$

Here, $\mathit{claim}_q(i)$ is a predicate formalizing for every state q what threads claim that there exists a partner thread and $\mathit{partner}_q(i,j)$ formalizes for a state q when a thread j is a partner of a thread i.

Non-injective agreement is a strong form of authentication. However, it does not exclude replay attacks, where an attacker replays messages from a previous session to successfully complete the protocol any number of times with A. To prevent such attacks, non-injective agreement is strengthened to injective agreement, which additionally requires that if A successfully completes the protocol n times with B, then B has been running the protocol at least n times.

> **Injective agreement** [Low97]: A protocol guarantees to an initiator A *agreement* with a responder B on a set of data items ds if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the terms in ds, and each such run of A corresponds to a *unique* run of B.

We generalize the idea of injective agreement as follows. Given a function f and a set $A \subseteq \mathit{dom}(f)$, we say that f is *injective on* A, written $\mathit{inj}_A(f)$, if and only if $\forall x, y \in A.\ f(x) = f(y) \Rightarrow x = y$. The *injective two-party authentication* property corresponding to a non-injective two-party authentication property is then

$$\forall q \in \mathit{reachable}(P).\ \exists f.\ \mathit{inj}_{\{i \in \mathit{TID} \mid \mathit{claim}_q(i)\}}(f) \wedge \forall i \in \mathit{TID}.\ \mathit{claim}_q(i) \Rightarrow \mathit{partner}_q(i, f(i))\ .$$

Here, as for a non-injective two-party authentication property, $\mathit{claim}_q(i)$ is a predicate formalizing for every state q what threads claim that there exists a partner thread and $\mathit{partner}_q(i,j)$ formalizes for a state q when a thread j is a partner of a thread i. Note that we only require f to be injective on all thread identifiers i for which $\mathit{claim}_q(i)$ holds because f is only evaluated on such thread identifiers.

Proving Injective Authentication Properties

In practice, many protocols rely on a challenge-response mechanism to achieve injectivity. For such protocols, we provide a general theorem that allows us to reduce proving an injective two-party authentication property to proving the corresponding non-injective authentication property. We explain how to prove non-injective authentication properties in the next chapter.

The employed challenge-response mechanisms typically work such that a thread i generates a nonce (the challenge) and sends it to its intended partner, which returns the nonce in a later message (the response). This response will only be accepted by thread i, as any other threads are waiting for responses containing a different challenge. The challenge-response mechanism thus binds every responding thread to a unique challenging thread.

In our formalization of authentication properties, the set of challenging threads of a state q is characterized by the *claim* predicate, while the associated responding threads are characterized by the *partner* predicate. The *partner* predicate uniquely binds responding threads to challenging threads if

$$\forall i_1, i_2, j \in \mathit{TID}.\ \mathit{partner}_q(i_1, j) \land \mathit{partner}_q(i_2, j) \Rightarrow i_1 = i_2\ .$$

We call this the *injectivity property*. For a *partner* predicate satisfying this property, non-injective authentication implies injective authentication. We formalize this by the following theorem, which we have proven in Isabelle/HOL.

Theorem 1 (Non-Injective to Injective Authentication)**.** *Assume that a protocol P satisfies the non-injective two-party authentication property*

$$\forall q \in \mathit{reachable}(P).\ \forall i \in \mathit{TID}.\ \mathit{claim}_q(i) \Rightarrow \exists j \in \mathit{TID}.\ \mathit{partner}_q(i, j)$$

for some definition of the predicates claim and partner. Then the corresponding injective two-party authentication property

$$\forall q \in \mathit{reachable}(P).\ \exists f.\ \mathit{inj}_{\{i \in \mathit{TID}.\ \mathit{claim}_q(i)\}}(f) \land \forall i \in \mathit{TID}.\ \mathit{claim}_q(i) \Rightarrow \mathit{partner}_q(i, f(i))\ .$$

also holds, provided that the partner predicate satisfies the injectivity property.

Example 4 (Injective authentication)**.** We prove that the C role of the CR protocol injectively synchronizes with the S role [CMdV06]. We define

$$\begin{aligned}\mathit{claim}_{(tr,th,\sigma)}(i) &:= \mathit{role}_{th}(i) = C \land \sigma(s, i) \notin \mathit{Compr} \land (i, C_2) \in \mathit{steps}(tr)\\ \mathit{partner}_{(tr,th,\sigma)}(i, j) &:= \mathit{role}_{th}(j) = S \land \sigma(s, i) = \sigma(s, j) \land k \sharp i = \sigma(v, j)\\ &\quad \land (i, C_1) \prec_{tr} (j, S_1) \land (j, S_2) \prec_{tr} (i, C_2)\ .\end{aligned}$$

The corresponding non-injective authentication property is exactly φ_{auth} from Example 3. Note that φ_{auth} holds for all reachable states of the CR protocol. We prove this later in Example 7 on page 33. Moreover, the *partner* predicate satisfies the injecitivity property, as $\mathit{partner}_{(tr,th,\sigma)}(i, j)$ implies the equality $k \sharp i = \sigma(v, j)$, i.e., threads i and j agree on

the nonce $k\sharp i$ generated by thread i. This equality implies the injectivity property, as $k\sharp i_1 = \sigma(v, j)$ and $k\sharp i_2 = \sigma(v, j)$ imply $k\sharp i_1 = \sigma(v, j) = k\sharp i_2$, which in turn implies $i_1 = i_2$. Using Theorem 1, we thus conclude that in the *CR* protocol the *C* role injectively synchronizes with the *S* role. ♠

4. Security Proofs Based on Decryption Chains

We now present our theory for proving security properties with respect to the semantics described in Chapter 3. As we already stated in the introduction, our theory consists of two key elements: the Chain rule and invariants constructed from protocol-specific type assertions.

We present the Chain rule together with the rest of the core inference rules in Section 4.1. We explain the intuition behind the Chain rule in Section 4.2 by describing our strategy for proving secrecy and authentication properties and by illustrating this strategy on the security properties of the *CR* protocol given in Example 3. Note that the *CR* protocol is one of the few protocols where our proof strategy works without auxiliary type assertions. The reasons for this will become clear when we explain our use of type assertions in Section 4.3. We conclude with a discussion of our proof construction method in Section 4.4.

4.1. Core Inference Rules

Our core inference rules are given in Figure 4.1. Each of these rules denotes a statement about the states in $\mathit{reachable}(P)$. The free variables of a rule are implicitly universally quantified. The rule Role for example denotes the statement

$$\forall (tr, th, \sigma) \in \mathit{reachable}(P).$$
$$\forall i\ R\ s'\ s.\ (\mathit{role}_{th}(i) = R \wedge s' <_R s \wedge (i, s) \in \mathit{steps}(tr)) \Rightarrow (i, s') \prec_{tr} (i, s)\ .$$

Note that we have formally proven every statement corresponding to one of our inference rules in Isabelle/HOL.

The rules Kn_1 and Kn_2 state that if the adversary knows a pair of messages $\langle m_1, m_2 \rangle$, then he also knows m_1 and m_2. Similarly, the rules Ord_1 and Ord_2 state that if a pair of messages $\langle m_1, m_2 \rangle$ was learned before the event e happened, then both m_1 and m_2 were also learned before e happened. These four rules allow us to reduce statements about the knowledge of pairs to the knowledge of the contained nonces, hashes, and encryptions. These rules are sound because the adversary's knowledge is closed under the projection of pairs.

The rules Known and Exec follow trivially from the definitions of $\prec_{tr}$, *knows*, and *steps*. They formalize the intuition that, if an event e happened before some other event, then e has happened.

The rules Irr and Trans formalize that $\prec_{tr}$ is a strict partial order. These rules are sound because roles are duplicate-free and our execution model therefore guarantees that all executed steps are unique and the adversary never learns the same message twice.

$$\frac{\langle m_1, m_2\rangle \in \mathit{knows}(tr)}{m_1 \in \mathit{knows}(tr)}\ \textsc{Kn}_1 \qquad \frac{\langle m_1, m_2\rangle \in \mathit{knows}(tr)}{m_2 \in \mathit{knows}(tr)}\ \textsc{Kn}_2$$

$$\frac{\langle m_1, m_2\rangle \prec_{tr} e}{m_1 \prec_{tr} e}\ \textsc{Ord}_1 \qquad \frac{\langle m_1, m_2\rangle \prec_{tr} e}{m_2 \prec_{tr} e}\ \textsc{Ord}_2$$

$$\frac{m \prec_{tr} e}{m \in \mathit{knows}(tr)}\ \textsc{Known} \qquad \frac{(i, s) \prec_{tr} e}{(i, s) \in \mathit{steps}(tr)}\ \textsc{Exec}$$

$$\frac{e \prec_{tr} e}{\mathit{false}}\ \textsc{Irr} \qquad \frac{e_1 \prec_{tr} e_2 \qquad e_2 \prec_{tr} e_3}{e_1 \prec_{tr} e_3}\ \textsc{Trans}$$

$$\frac{\mathit{role}_{th}(i) = R \qquad s' \prec_R s \qquad (i, s) \in \mathit{steps}(tr)}{(i, s') \prec_{tr} (i, s)}\ \textsc{Role}$$

$$\frac{(i, \mathsf{Recv}_l(pt)) \in \mathit{steps}(tr)}{\mathit{inst}_{\sigma,i}(pt) \prec_{tr} (i, \mathsf{Recv}_l(pt))}\ \textsc{Input}$$

$$\frac{m \in \mathit{knows}(tr)}{\begin{array}{l}(m \in AK_0) \vee \\ (\exists x.\ \ m = \mathsf{h}(x) \ \ \wedge x \prec_{tr} \mathsf{h}(x) \qquad\qquad\qquad\quad) \vee \\ (\exists x\ k.\ m = \{\!|\, x \,|\!\}_k \ \wedge x \prec_{tr} \{\!|\, x \,|\!\}_k \ \wedge k \prec_{tr} \{\!|\, x \,|\!\}_k \ ) \vee \\ (\exists x\ y.\ m = \langle x, y\rangle \ \wedge x \prec_{tr} \langle x, y\rangle \ \wedge y \prec_{tr} \langle x, y\rangle \ ) \vee \\ (\exists R \in P.\ \exists\, \mathsf{Send}_l(pt) \in R.\ \exists i.\ \mathit{role}_{th}(i) = R \wedge \\ \qquad\qquad\qquad\qquad \mathit{chain}_{tr}(\{(i, \mathsf{Send}_l(pt))\},\ \mathit{inst}_{\sigma,i}(pt),\ m)\ )\end{array}}\ \textsc{Chain}$$

Figure 4.1.: Core inference rules for decryption-chain reasoning, which hold under the assumption that $(tr, th, \sigma) \in \mathit{reachable}(P)$.

$$\begin{array}{l}\mathit{chain}_{tr}(E, m', m) := \\ \quad (\ \ m' = m \ \ \wedge (\forall e \in E.\ e \prec_{tr} m \ \ ) \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad) \vee \\ \quad (\exists x\, k.\, m' = \{\!|\, x \,|\!\}_k \wedge (\forall e \in E.\ e \prec_{tr} \{\!|\, x \,|\!\}_k) \wedge \mathit{chain}_{tr}(\{\{\!|\, x \,|\!\}_k, k^{-1}\}, x, m) \qquad) \vee \\ \quad (\exists x\, y.\, m' = \langle x, y\rangle \qquad\qquad\qquad\quad \wedge (\mathit{chain}_{tr}(E, x, m) \vee \mathit{chain}_{tr}(E, y, m)))\end{array}$$

Figure 4.2.: Definition of the *chain* predicate using recursion over the message m'.

The rule ROLE states that if a thread i that is an instance of role R has executed role step s, then all the role steps $s' <_R s$ have been executed before s by the thread i. This rule is sound because both the SEND and the RECV transitions execute role steps in the order specified by the roles.

The rule INPUT states that if a thread i has executed a receive step $\mathsf{Recv}_l(pt)$, then the instantiated pattern pt was learned before $\mathsf{Recv}_l(pt)$ was executed by the thread i. This rule is sound because the RECV transition ensures that the adversary knows the received message.

The rule CHAIN states that there are precisely five ways that an adversary can learn a message m.

1. He knew m from the start.
2. m is a hash $\mathsf{h}(x)$ of the known message x and the adversary built $\mathsf{h}(x)$ himself using the HASH transition.
3. m is an encryption $\{\!|\, x \,|\!\}_k$ of a known message x with a known key k and the adversary built $\{\!|\, x \,|\!\}_k$ himself using the ENCR transition.
4. m is a pair $\langle x, y \rangle$ of two known messages x and y and the adversary built $\langle x, y \rangle$ himself using the PAIR transition.
5. There was some step $\mathsf{Send}_l(pt)$ executed by some thread i such that the adversary learned the sent message $inst_{\sigma,i}(pt)$ and from this message he learned m using zero or more decryptions and projections.

We prove the soundness of this case distinction by induction over the reachable states $(tr, th, \sigma) \in reachable(P)$. The key idea behind Case (5), called the *decryption-chain case*, is that the adversary can only learn a message by decrypting an encryption that *he did not build himself*. Analogously, the adversary can only learn a message by projecting a pair that he did not build himself. Thus, whenever the adversary learns a message m by decrypting an encryption $\{\!|\, x \,|\!\}_k$, then he must have learned $\{\!|\, x \,|\!\}_k$ from a send step or by decrypting an encryption or projecting a pair containing $\{\!|\, x \,|\!\}_k$. As every message is of finite size, any such chain of repeated decryptions and projections is of finite length.

We formalize the notion of *decryption chains* using the *chain* predicate defined in Figure 4.2. For a set $E \subseteq Event$, the expression $chain_{tr}(E, m', m)$ formalizes that the adversary learned the message m using zero or more decryptions and projections after he learned some message in $split(m')$, which he learned after the events in E happened. The definition of *chain* distinguishes between three cases.

1. m' is the message m and the adversary learned m' after the events in E, or
2. m' is an encryption $\{\!|\, x \,|\!\}_k$ and the adversary learned m after he used the inverse key k^{-1} to decrypt $m' = \{\!|\, x \,|\!\}_k$, which he learned after the events in E, or
3. m' is a pair $\langle x, y \rangle$ and the adversary learned m from a chain starting from x or from a chain starting from y. The set E is unchanged in this case because, in our protocol semantics, the messages x and y are learned at the same time or before the pair $\langle x, y \rangle$.

4.2. Proof Strategy

Suppose we want to prove that a protocol P satisfies a security property φ. Our strategy for proving this consists of two main steps. First, we simplify the use of the Chain rule by instantiating it for the protocol P. This entails specializing the Chain rule for different outermost message constructors in its premise, enumerating P's roles and their send steps in the conclusion, and completely unfolding all occurrences of the *chain* predicate. Second, we prove that φ holds for every reachable state of the protocol P by repeatedly applying the simplified Chain rule to the messages that the adversary is supposed to know (e.g., received messages). Combined with straightforward logical reasoning formalized in HOL, this suffices to complete the proof in many cases.

The first step is completely mechanical. It allows us to share work between different security proofs of the same protocol. Moreover, it yields a compact description of the adversary's message derivation capabilities in the context of a given security protocol. We illustrate this first step on the *CR* protocol.

Example 5. Figure 4.3 shows the simplified instances of the Chain rule for the *CR* protocol. These rules capture the adversary's message deduction capabilities in the context of the *CR* protocol. For example, the *CR* protocol does not send private keys and long-term symmetric keys in an accessible position. Intuitively, the adversary can therefore learn such keys only from his initial knowledge. The rules SKChain$_{CR}$ and KChain$_{CR}$, which were derived mechanically from the Chain rule, justify this intuition. Analogously, the rule NChain$_{CR}$ shows that there is exactly one way for the adversary to learn a nonce $n\sharp i$. This nonce must be the freshly generated key $k\sharp i$ that the client thread i sent encrypted in its first step. ♠

In the second step of our proof strategy, the Chain rule is used to prove the security property of interest, φ. In general, there will be multiple premises of the form $m \in knows(tr)$ to which the Chain rule can be applied. Hence, a choice must be made. A rule application cannot render a previously provable security property unprovable. However, unnecessary applications of the Chain rule would needlessly increase the size of the resulting proof. To find short, direct proofs, we use the following strategy to prove both secrecy and authentication properties.

To prove *secrecy properties*, we use the Chain rule both for the message m to be proven secret as well as for the keys that must be kept secret if m is not to be decrypted. If the secrecy of some message depends on its authenticity (e.g., a key that is received), we use the proof strategy for authentication properties outlined below.

To prove *authentication properties*, we use the Chain rule on the received message m to justify why its receipt implies the existence of a partner thread that sent m. If the authenticity of a message depends on the secrecy of another message (e.g., the key used for a MAC), then we use the strategy for secrecy properties outlined above.

We also factor out repeated subproofs, such as proofs about secrecy properties. We do this by introducing additional lemmas that generalize the properties proven by the repeated subproofs.

Note that we may switch multiple times between proving secrecy properties and proving authentication properties. In general, this does not result in cyclic dependencies between proofs because these proofs concern different messages. Some cases where

$$\frac{\mathsf{sk}(a) \in \mathit{knows}(tr)}{\mathsf{sk}(a) \in AK_0}\ \textsc{SKChain}_{CR} \qquad \frac{\mathsf{k}(a,b) \in \mathit{knows}(tr)}{\mathsf{k}(a,b) \in AK_0}\ \textsc{KChain}_{CR}$$

$$\frac{n \sharp i \in \mathit{knows}(tr)}{\begin{array}{l}\mathit{role}_{th}(i) = C \ \wedge\ (i, C_1) \prec_{tr} \{\!|\, k \sharp i \,|\!\}_{\mathsf{pk}(\sigma(s,i))} \prec_{tr} k \sharp i \\ \qquad \wedge\ \mathsf{sk}(\sigma(s,i)) \prec_{tr} k \sharp i \ \wedge\ k = n\end{array}}\ \textsc{NChain}_{CR}$$

$$\frac{\mathsf{h}(x) \in \mathit{knows}(tr)}{\begin{array}{l}(x \prec_{tr} \mathsf{h}(x)) \vee \\ (\exists j.\ \mathit{role}_{th}(j) = S \wedge (j, S_2) \prec_{tr} \mathsf{h}(\sigma(v,j)) = \mathsf{h}(x))\end{array}}\ \textsc{HChain}_{CR}$$

$$\frac{\{\!|\, m \,|\!\}_x \in \mathit{knows}(tr)}{\begin{array}{l}(m \prec_{tr} \{\!|\, m \,|\!\}_x \wedge x \prec_{tr} \{\!|\, m \,|\!\}_x) \vee \\ (\exists j.\ \mathit{role}_{th}(j) = C \wedge (j, C_1) \prec_{tr} \{\!|\, k \sharp j \,|\!\}_{\mathsf{pk}(\sigma(s,j))} = \{\!|\, m \,|\!\}_x)\end{array}}\ \textsc{EChain}_{CR}$$

Figure 4.3.: Simplified instances of the Chain rule for the *CR* protocol

decryption-chain reasoning fails could however be interpreted as a cyclic dependency of a proof on itself. We discuss this later in Section 4.4.

We illustrate the second step of our proof strategy on the two security properties of the *CR* protocol described in Example 3. We first prove a secrecy property. Afterwards, we prove an authentication property and show how we can use the already proven secrecy property to prove two secrecy subproofs.

Example 6 (Proof of session-key secrecy)**.** We prove $\forall q \in \mathit{reachable}(CR).\ \varphi_{\text{sec}}(q)$, for φ_{sec} from Example 3.

Proof. Suppose the secrecy property φ_{sec} does not hold for some state $(tr, th, \sigma) \in \mathit{reachable}(CR)$. Then there is a thread i such that $\mathit{role}_{th}(i) = C$, $\sigma(s,i) \notin \mathit{Compr}$, and $k \sharp i \in \mathit{knows}(tr)$. Hence, the adversary must have learned $k \sharp i$.

There is only one premise, $k \sharp i \in \mathit{knows}(tr)$, to which we can apply the Chain rule. Instead of applying the Chain rule directly, we apply the corresponding simplified instance, $\textsc{NChain}_{CR}$. From its conclusion, we see that the adversary can learn $k \sharp i$ only by decrypting the message $\{\!|\, k \sharp i \,|\!\}_{\mathsf{pk}(\sigma(s,i))}$, which implies that $\mathsf{sk}(\sigma(s,i)) \prec_{tr} k \sharp i$.

Using Known, we have that $\mathsf{sk}(\sigma(s,i)) \in \mathit{knows}(tr)$ and, from $\textsc{SKChain}_{CR}$, we conclude $\mathsf{sk}(\sigma(s,i)) \in AK_0$. Given the definition of AK_0, we have $\sigma(s,i) \in \mathit{Compr}$, which contradicts our assumptions.

We therefore conclude that φ_{sec} holds for all reachable states of the *CR* protocol. ♠

Example 7 (Proof of non-injective synchronization)**.**
We prove that $\forall q \in \mathit{reachable}(CR).\ \varphi_{\text{auth}}(q)$, where φ_{auth} is defined in Example 3.

Proof. We must show that for every state $(tr, th, \sigma) \in \mathit{reachable}(CR)$ and every thread i such that $\mathit{role}_{th}(i) = C$, $\sigma(s,i) \notin \mathit{Compr}$, and $(i, C_2) \in \mathit{steps}(tr)$, there is a thread j such

that *syncWith(j)* holds.

$$\begin{aligned} \mathit{syncWith(j)} := \mathit{role}_{th}(j) = S \wedge \sigma(s,i) = \sigma(s,j) \wedge k\sharp i = \sigma(v,j) \\ \wedge (i, C_1) \prec_{tr} (j, S_1) \wedge (j, S_2) \prec_{tr} (i, C_2) \end{aligned}$$

We prove this by applying the CHAIN rule to the received messages.

From $(i, C_2) \in tr$, we have that $\mathsf{h}(k\sharp i) \prec_{tr} (i, C_2)$ using the INPUT rule and $\mathsf{h}(k\sharp i) \in \mathit{knows}(tr)$ using the KNOWN rule. Applying the HCHAIN_{CR} rule yields the following conclusion, whose disjuncts we have numbered.

$$\begin{aligned} &\textbf{(1)} \quad (k\sharp i \prec_{tr} \mathsf{h}(k\sharp i)) \\ &\textbf{(2)} \quad \vee (\exists j \in \mathit{TID}.\ \mathit{role}_{th}(j) = S \wedge (j, S_2) \prec_{tr} \mathsf{h}(\sigma(v,j)) \wedge \mathsf{h}(\sigma(v,j)) = \mathsf{h}(k\sharp i)) \ . \end{aligned}$$

Case **(1)** is where the adversary builds the received message by himself. Using the KNOWN rule, we have that $k\sharp i \in \mathit{knows}(tr)$. This contradicts the secrecy property we proved in Example 6.

Case **(2)** implies that there is a server thread j, $\mathit{role}_{th}(j) = S$, that sent the message that the client thread i received. We show that client i synchronizes with server j.

From $\mathsf{h}(k\sharp i) = \mathsf{h}(\sigma(v,j))$ and the injectivity of $\mathsf{h}(\_)$, it follows that $k\sharp i = \sigma(v,j)$. From $(j, S_2) \prec_{tr} \mathsf{h}(\sigma(v,j))$, $\mathsf{h}(\sigma(v,j)) = \mathsf{h}(k\sharp i)$, and $\mathsf{h}(k\sharp i) \prec_{tr} (i, C_2)$, it follows that $(j, S_2) \prec_{tr} (i, C_2)$. To establish *syncWith(j)*, it remains to be shown that the first message of client i was received in the first step of server j; i.e., $\sigma(s,i) = \sigma(s,j)$ and $(i, C_1) \prec_{tr} (j, S_1)$.

From $(j, S_2) \prec_{tr} (i, C_2)$, we have $(j, S_1) \prec_{tr} (j, S_2)$ using the rules EXEC and ROLE. Hence, $\{\!|\, k\sharp i \,|\!\}_{\mathsf{pk}(\sigma(s,j))} \prec_{tr} (j, S_1)$ using the rules EXEC and INPUT and the fact $k\sharp i = \sigma(v,j)$. From the KNOWN rule, we have $\{\!|\, k\sharp i \,|\!\}_{\mathsf{pk}(\sigma(s,j))} \in \mathit{knows}(tr)$. Applying the ECHAIN_{CR} rule yields

$$\begin{aligned} &\textbf{(2.1)} \quad (\, k\sharp i \prec_{tr} \{\!|\, k\sharp i \,|\!\}_{\mathsf{pk}(\sigma(s,j))} \wedge \mathsf{pk}(\sigma(s,j)) \prec_{tr} \{\!|\, k\sharp i \,|\!\}_{\mathsf{pk}(\sigma(s,j))}) \\ &\textbf{(2.2)} \quad \vee (\, \exists i'.\ \mathit{role}_{th}(i') = C \wedge (i', C_1) \prec_{tr} \{\!|\, k\sharp i' \,|\!\}_{\mathsf{pk}(\sigma(s,i'))} \wedge \\ &\qquad\qquad \{\!|\, k\sharp i' \,|\!\}_{\mathsf{pk}(\sigma(s,i'))} = \{\!|\, k\sharp i \,|\!\}_{\mathsf{pk}(\sigma(s,j))} \qquad\qquad) \ . \end{aligned}$$

Case **(2.1)** states that the adversary fakes the message, which again contradicts the secrecy property proven in Example 6 due to $k\sharp i \prec_{tr} \{\!|\, k\sharp i \,|\!\}_{\mathsf{pk}(\sigma(s,j))}$ and the rule KNOWN.

Case **(2.2)** implies $i' = i$ since $k\sharp i' = k\sharp i$. Hence, we have

$$(i, C_1) \prec_{tr} \{\!|\, k\sharp i \,|\!\}_{\mathsf{pk}(\sigma(s,i))} = \{\!|\, k\sharp i \,|\!\}_{\mathsf{pk}(\sigma(s,j))} \prec_{tr} (j, S_1) \ .$$

This implies that $\sigma(s,i) = \sigma(s,j)$ and $(i, C_1) \prec_{tr} (j, S_1)$, which concludes the proof. ♠

4.3. Type Assertions

The proof strategy outlined in the previous subsection relies on our ability to instantiate the CHAIN rule and *completely* unfold its conclusion. This is straightforward in our *CR* example because it does not have a send step with a variable in an *accessible position*, i.e., a position that is neither below a hash nor below an encryption-key position.

However, most protocols do have send steps with variables in accessible positions and this results in expressions of the form $chain_{tr}(E, \sigma(v,i), m)$ in the conclusions of the simplified CHAIN rule instances. In general, $\sigma(v,i)$ can be an arbitrary message; hence, there *may* be a decryption chain starting from $\sigma(v,i)$ and resulting in m. However, for a concrete protocol, the assumption that $\sigma(v,i)$ is an arbitrary message is too pessimistic because the set of possible instantiations of protocol variables is restricted by both the protocol specification and the operational semantics. What we are missing to completely unfold these $chain_{tr}(E, \sigma(v,i), m)$ expressions is a formalization of these restrictions. We capture these using invariants that are constructed from protocol-specific type assertions.

We explain our use of type assertions in three steps. First, we explain how we formalize type assertions. Then, we explain how we use them to completely unfold the conclusion of CHAIN rule instances. Finally, we give a theorem for proving the soundness of type assertions using decryption-chain reasoning.

4.3.1. Syntax and Semantics of Type Assertions

A *type* is a term constructed according to the following grammar.

$$\begin{aligned} \mathit{Type} ::= {} & \mathit{Const} \mid \mathsf{Ag} \mid \mathit{Role.Fresh} \mid \langle \mathit{Type}, \mathit{Type} \rangle \mid \mathsf{h}(\mathit{Type}) \mid \{\!|\, \mathit{Type} \,|\!\}_{\mathit{Type}} \\ & \mid \mathsf{pk}(\mathit{Type}) \mid \mathsf{sk}(\mathit{Type}) \mid \mathsf{k_{un}} \mid \mathsf{k_{bi}} \mid \mathit{Type} \cup \mathit{Type} \mid \mathsf{kn}(\mathit{RoleStep}) \end{aligned}$$

Intuitively, types denote sets of messages. For every global constant $\gamma \in \mathit{Const}$, the type γ denotes the set $\{\gamma\}$. The type Ag denotes the set of all agent names. The type $R.f$ denotes all fresh messages named f that were generated by some thread executing role R. The type constructors $\langle \_, \_ \rangle$, $\mathsf{h}(\_)$, $\{\!|\, \_ \,|\!\}_\_$, $\mathsf{pk}(\_)$, and $\mathsf{sk}(\_)$ reflect message constructors to the type level. The type $\mathsf{k_{un}}$ denotes all pre-shared unidirectional keys and the type $\mathsf{k_{bi}}$ denotes all pre-shared bidirectional keys. The type $\alpha \cup \beta$ denotes the sum of the two types α and β. We use type-sums to type variables that are instantiated with messages of different types.

The denotation of a type $\mathsf{kn}(s)$ is context-dependent. It depends on the system state and the thread i whose variables' instantiation we are specifying. The denotation of $\mathsf{kn}(s)$ is the set of all messages known to the adversary before the role step s was executed by the thread i. We use $\mathsf{kn}(s)$ types to account for the interaction with the active adversary, as we illustrate in the following example.

Example 8 (Type assertion)**.** The variable v of the server role S of the *CR* protocol is always instantiated with messages of type $C.k \cup \mathsf{kn}(S_1)$.

Note that the type $C.k$ does not cover all possible instantiations of v. It covers all instantiations that occur when receiving messages sent by client threads. The type-sum with $\mathsf{kn}(S_1)$ also covers those instantiations that are the result of receiving messages constructed by the adversary. For such an instantiation σ, the variable v of a thread i executing the S role is obviously not guaranteed to be instantiated with some message $k \sharp j$ that was freshly generated by some thread j executing the C role. However, it is guaranteed that the adversary knew $\sigma(v,i)$ before (i, S_1) was executed, as he constructed the received message himself.

In general, we construct the type of a variable v as follows. We use type constructors other than $\mathsf{kn}(\_)$ to specify the shape of v's instantiations which result from receiving messages sent by other threads that execute protocol roles. We use the $\mathsf{kn}(s)$ type to account for message parts that could be injected by the adversary, where the role step s is the step where v is instantiated. This construction works in all our case studies. Moreover, we can even automatically infer the types of almost all variables in our case studies using a simple heuristic. ♠

Formally, the meaning of a type is given by the *type interpretation function* $[\![\_]\!]$, which associates to every type γ the set of messages $[\![\gamma]\!]^i_q$ denoted by the type γ in the context of a thread $i \in \mathit{TID}$ and a system state $q = (\mathit{tr}, \mathit{th}, \sigma)$.

$$[\![\gamma]\!]^i_q := \begin{cases} \{\gamma\} & \text{if } \gamma \in \mathit{Const} \\ \mathit{Agent} & \text{if } \gamma = \mathsf{Ag} \\ \{n\sharp j \mid \mathit{role}_{\mathit{th}}(j) = R\} & \text{if } \gamma = R.n \\ \{\mathsf{h}(x) \mid x \in [\![\alpha]\!]^i_q\} & \text{if } \gamma = \mathsf{h}(\alpha) \\ \{(x,y) \mid x \in [\![\alpha]\!]^i_q \wedge y \in [\![\beta]\!]^i_q\} & \text{if } \gamma = \langle\alpha,\beta\rangle \\ \{\{\!|\, x \,|\!\}_k \mid x \in [\![\alpha]\!]^i_q \wedge k \in [\![\beta]\!]^i_q\} & \text{if } \gamma = \{\!|\, \alpha \,|\!\}_\beta \\ \{\mathsf{pk}(x) \mid x \in [\![\alpha]\!]^i_q\} & \text{if } \gamma = \mathsf{pk}(\alpha) \\ \{\mathsf{sk}(x) \mid x \in [\![\alpha]\!]^i_q\} & \text{if } \gamma = \mathsf{sk}(\alpha) \\ \{\mathsf{k_{un}}(a,b) \mid a,b \in \mathit{Agent}\} & \text{if } \gamma = \mathsf{k_{un}} \\ \{\mathsf{k_{bi}}(A) \mid A \subseteq \mathit{Agent}\} & \text{if } \gamma = \mathsf{k_{bi}} \\ [\![\alpha]\!]^i_q \cup [\![\beta]\!]^i_q & \text{if } \gamma = \alpha \cup \beta \\ \{m \mid m \prec_{\mathit{tr}} (i,s)\} & \text{if } \gamma = \mathsf{kn}(s) \end{cases}$$

We specify all type assertions for a single protocol together as a *typing*, which is a partial function $\mathit{ty} : (\mathit{Role} \times \mathit{Var}) \nrightarrow \mathit{Type}$. A state is *well-typed* with respect to a typing ty if and only if every variable is instantiated with a message denoted by its type; that is

$$\mathit{well\text{-}typed}_{\mathit{ty}}(\mathit{tr}, \mathit{th}, \sigma) := \forall (i,s) \in \mathit{steps}(\mathit{tr}).\ \forall v \in \mathit{vars}(s).\ \sigma(v,i) \in [\![\mathit{ty}(\mathit{role}_{\mathit{th}}(i), v)]\!]^i_{(\mathit{tr},\mathit{th},\sigma)} \ .$$

We assume that *vars* is extended to role steps such that $\mathit{vars}(s)$ denotes the variables of the message pattern of the role step s.

A protocol P is *well-typed* with respect to a typing ty if and only if all its reachable states are well-typed with respect to ty; that is

$$\mathit{well\text{-}typed}_{\mathit{ty}}(P) := \forall q \in \mathit{reachable}(P).\ \mathit{well\text{-}typed}_{\mathit{ty}}(q) \ .$$

Conversely, we say that a typing ty is *sound* with respect to a protocol P if and only if P is well-typed with respect to ty.

We illustrate the use of type assertions on the CR' protocol, which is identical to the *CR* protocol from Example 1 except that in the first message the client identity is also sent and the server uses public key encryption to return the session key.

$$\frac{\mathsf{h}(x) \in \mathit{knows}(tr)}{\begin{array}{l}(x \prec_{tr} \mathsf{h}(x)) \\ \vee\ (\exists j.\ \mathit{role}_{th}(j) = C' \wedge (j, C'_1) \prec_{tr} \{\!|\, \sigma(c,j), k \sharp j \,|\!\}_{\mathsf{pk}(\sigma(s,j))} \\ \qquad\qquad \wedge\ \mathit{chain}_{tr}(\{\{\!|\, \sigma(c,j), k \sharp j \,|\!\}_{\mathsf{pk}(\sigma(s,j))}, \mathsf{sk}(\sigma(s,j))\}, \sigma(c,j), \mathsf{h}(x))) \\ \vee\ (\exists j.\ \mathit{role}_{th}(j) = S' \wedge (j, S'_2) \prec_{tr} \{\!|\, \sigma(v,j) \,|\!\}_{\mathsf{pk}(\sigma(c,j))} \\ \qquad\qquad \wedge\ \mathit{chain}_{tr}(\{\{\!|\, \sigma(v,j) \,|\!\}_{\mathsf{pk}(\sigma(c,j))}, \mathsf{sk}(\sigma(c,j))\}, \sigma(v,j), \mathsf{h}(x)) \qquad)\end{array}} \text{HCHAIN}_{CR'}$$

Figure 4.4.: Incomplete unfolding of the $\text{HCHAIN}_{CR'}$ rule

Example 9 (CR' Protocol)**.** We define $CR' := \{C', S'\}$, where C' and S' are defined as follows for $c, s \in \mathit{AVar}$, $k \in \mathit{Fresh}$, and $v \in \mathit{MVar}$.

$$\begin{aligned} C' &:= [\ \mathsf{Send}_1(\{\!|\, c, k \,|\!\}_{\mathsf{pk}(s)}),\quad \mathsf{Recv}_1(\{\!|\, k \,|\!\}_{\mathsf{pk}(c)})\ ] \\ S' &:= [\ \mathsf{Recv}_1(\{\!|\, c, v \,|\!\}_{\mathsf{pk}(s)}),\quad \mathsf{Send}_2(\{\!|\, v \,|\!\}_{\mathsf{pk}(c)})\ ] \end{aligned}$$

Note that we write $\{\!|\, c, k \,|\!\}_{\mathsf{pk}(s)}$ as an abbreviation of $\{\!|\, \langle c, k \rangle \,|\!\}_{\mathsf{pk}(s)}$. The CR' protocol is well-typed with respect to the typing

$$\begin{aligned} CR'_{ty} := \{\ &(C', c) \mapsto \mathsf{Ag}, (C', s) \mapsto \mathsf{Ag}, \\ &(S', c) \mapsto \mathsf{Ag}, (S', s) \mapsto \mathsf{Ag}, (S', v) \mapsto (C'.k \cup \mathsf{kn}(S'_1))\ \}\ . \end{aligned}$$

We provide a formal proof of this claim in Example 11 after we have shown how to exploit type assertions and how to prove their soundness. ♠

4.3.2. Exploiting Type Assertions

The following example illustrates the problem that we will solve using type assertions.

Example 10. As the CR' protocol does not send any hashes, one may expect its CHAIN rule for hashes to be

$$\frac{\mathsf{h}(x) \in \mathit{knows}(tr)}{x \prec_{tr} \mathsf{h}(x)}\ \text{HCHAIN}_{CR'}\ .$$

However, as we can see in Figure 4.4, there are also two decryption-chain cases, subsequently named (a) and (b), that arise when instantiating and unfolding the CHAIN rule. Case (a) states that it may be possible to learn $\mathsf{h}(x)$ from the content $\sigma(c,j)$ of the agent variable c that the client sends in its first message. Case (b) states that it may be possible to learn $\mathsf{h}(x)$ from the content $\sigma(v,j)$ of the variable v that the server sends in its second message.

We can remove both of these cases, as they are always false. This is obvious for Case (a), as agent variables contain agent names and agent names are never equal to hashes. For Case (b), the intuition is that the adversary must have faked the message received in step (j, S'_1) for the variable $\sigma(v,j)$ to contain the hash $\mathsf{h}(x)$. Hence, the adversary must have known $\mathsf{h}(x)$ already before (j, S'_1) was executed. This contradicts the statement in Case (b) that the adversary did *not* know $\mathsf{h}(x)$ before step (j, S'_2).

Formally, the above arguments exploit the fact that the CR' protocol is well-typed with respect to the type assertion CR'_{ty}. From this and $(tr, th, \sigma) \in \mathit{reachable}(CR')$, we have $\mathit{well\text{-}typed}_{CR'_{ty}}(tr, th, \sigma)$, which we use as follows to show that the cases (a) and (b) are contradictory.

In Case (a), we use $\mathit{well\text{-}typed}_{CR'_{ty}}(tr, th, \sigma)$ to derive

$$\sigma(c, j) \in [\![CR'_{ty}(C', c)]\!]^{j}_{(tr,th,\sigma)} \ ,$$

which is equivalent to $\sigma(c, j) \in [\![\mathsf{Ag}]\!]^{j}_{(tr,th,\sigma)}$ and hence also to $\sigma(c, j) \in \mathit{Agent}$. Therefore, only the first case of the *chain* predicate applies, which implies $\sigma(c, j) = \mathsf{h}(x)$ and hence $\mathsf{h}(x) \in \mathit{Agent}$, which is false. Thus we can remove the first decryption-chain case in Figure 4.4.

In Case (b), we use $\mathit{well\text{-}typed}_{CR'_{ty}}(tr, th, \sigma)$ to show that $\sigma(v, j) \in [\![C'.k \cup \mathsf{kn}(S'_1)]\!]^{j}_{(tr,th,\sigma)}$, which is equivalent to

$$\begin{array}{ll} \textbf{(1)} & (\exists i.\ \mathit{role}_{th}(i) = C' \wedge \sigma(v, j) = k \sharp i) \vee \\ \textbf{(2)} & (\sigma(v, j) \prec_{tr} (j, S'_1)) \ . \end{array}$$

For subcase **(1)**, we can proceed as for the agent variable above. We unfold the *chain* predicate and conclude $k \sharp i = \mathsf{h}(x)$, which is a contradiction. For subcase **(2)**, we conclude

$$\sigma(v, j) \prec_{tr} (j, S'_1) \prec_{tr} (j, S'_2) \prec_{tr} \{\!|\, \sigma(v, j) \,|\!\}_{\mathsf{pk}(\sigma(c,j))} \wedge \mathit{chain}_{tr}(\{\{\!|\, \sigma(v, j) \,|\!\}_{\mathsf{pk}(\sigma(c,j))}, \mathsf{sk}(\sigma(c, j))\}, \sigma(v, j), \mathsf{h}(x))$$

by combining all facts and additionally using the rules Exec and Role to derive $(j, S'_1) \prec_{tr} (j, S'_2)$. Note that $\mathit{chain}_{tr}(E, m, m')$ implies that there exists an $x \in \mathit{split}(m)$ such that $e \prec_{tr} x$ for every $e \in E$. Hence there exists an $x \in \mathit{split}(\sigma(v, j))$ such that $x \prec_{tr} (j, S'_1) \prec_{tr} \{\!|\, \sigma(v, j) \,|\!\}_{\mathsf{pk}(\sigma(c,j))} \prec_{tr} x$. This contradicts the irreflexivity of $\prec_{tr}$ and we can also remove the second decryption-chain case in Figure 4.4. ♠

In general, we exploit type assertions by instantiating the rule TypChain given in Figure 4.5 instead of the Chain rule in the first step of our proof strategy. The TypChain

$$\frac{m \in \mathit{knows}(tr) \qquad \mathit{well\text{-}typed}_{ty}(tr, th, \sigma)}{\begin{array}{l} (m \in AK_0) \vee \\ (\exists x.\ \ m = \mathsf{h}(x) \ \ \wedge x \prec_{tr} \mathsf{h}(x) \qquad\qquad\qquad\quad) \vee \\ (\exists x\ k.\ m = \{\!|\, x \,|\!\}_k \ \wedge x \prec_{tr} \{\!|\, x \,|\!\}_k \ \wedge k \prec_{tr} \{\!|\, x \,|\!\}_k \ ) \vee \\ (\exists x\ y.\ m = \langle x, y \rangle \ \wedge x \prec_{tr} \langle x, y \rangle \ \wedge y \prec_{tr} \langle x, y \rangle \ ) \vee \\ (\exists R \in P.\ \exists\, \mathsf{Send}_l(pt) \in R.\ \exists j.\ \mathit{role}_{th}(j) = R \wedge \\ \qquad\qquad (\forall v \in \mathit{vars}(pt).\ \sigma(v, j) \in [\![\mathit{ty}(R, v)]\!]^{j}_{(tr,th,\sigma)}) \wedge \\ \qquad\qquad \mathit{chain}_{tr}(\{(j, \mathsf{Send}_l(pt))\},\ \mathit{inst}_{\sigma,j}(pt),\ m) \qquad) \end{array}} \text{TypChain}$$

Figure 4.5.: The TypChain rule, a typed version of the Chain rule.

$$\frac{m \prec_{tr} e \qquad e \in E \qquad \mathit{chain}_{tr}(E, m, x)}{\mathit{false}}\ \textsc{ChainIrr}$$

Figure 4.6.: The ChainIrr rule, which holds for $(tr, th, \sigma) \in \mathit{reachable}(P)$.

rule is a version of the Chain rule that additionally states that all variables contained in send steps are instantiated according to their type. Hence, whenever we encounter an expression of the form $\mathit{chain}_{tr}(E, \sigma(v, j), m)$ in a decryption-chain case, we can proceed by instantiating $\sigma(v, j)$ with an arbitrary message m corresponding to v's type and simplifying the resulting cases as follows.

The type constructors corresponding to the message constructors constrain the structure of m sufficiently that we can unfold the *chain* predicate. A union-type $\alpha \cup \beta$ results in an additional case split. For a $\mathsf{kn}(s)$ type, we can reduce the case to false, provided that s is the role step where v is instantiated. We ensure this side condition by constructing our type assertions accordingly. To justify the reduction to false, we use ChainIrr rule given in Figure 4.6, which generalizes the argument used in the example above.

4.3.3. Proving the Soundness of Type Assertions

The core inference rules together with the rules for exploiting type assertions allows one to prove security properties for a protocol P under the assumption that P is well-typed for some fixed type assertion *ty*. Such a proof can be seen as a proof in a typed model, which ensures *by definition* that variables are only instantiated according to their specified type. However, such a proof does not exclude the existence of *type-flaw attacks*, which are reachable states that are not well-typed and violate the security property. To exclude the existence of type-flaw attacks, we must prove the soundness of the type assertion *ty*; i.e., we must prove that

$$\forall q \in \mathit{reachable}(P).\ \mathit{well\text{-}typed}_{ty}(q)\ .$$

We prove this by induction over the reachable states of the protocol P. The only non-trivial case of this induction stems from the Recv transition. It is the only transition where a thread i could add a role step s that instantiates a variable $v \in \mathit{vars}(s)$ to the trace. If v is an agent variable, then it suffices to show that it is mapped to the Ag type, as agent variables are always instantiated with agent names. If v is a message variable, then we exploit that the adversary must know the message m received by the role step s and that, from the induction hypothesis, the current state is well-typed with respect to the type assertion *ty*. Hence, we can use the TypChain rule to establish the possible origins of the received message m. If the type assertion *ty* is sound, then we expect the kn part of v's type to cover the case where the adversary fakes m and the rest of v's type to cover the case where the message is (part of) a message sent by another thread. The following theorem, proven in Isabelle/HOL, formalizes this argument.

Theorem 2 (Soundness of Type Assertions)**.** *A protocol P is well-typed with respect to the typing ty provided the following two propositions hold.*

1. *For every* $R \in P$, $s \in R$, *and* $v \in vars(s) \cap AVar$, *it holds that* $ty(R, v) = \mathsf{Ag}$.
2. *For every* $R \in P$, $s \in R$, *and* $v \in vars(s) \cap MVar$, *every state* $(tr, th, \sigma) \in reachable(P)$, *every thread* $i \in dom(th)$, *and all sequences of role steps* done *and* rest, *the assumptions*
   - $inst_{\sigma,i}(pt) \in knows(tr)$
   - $th(i) = (R, [\mathsf{Recv}_l(pt)] \cdot \mathrm{rest})$
   - $R = \mathrm{done} \cdot [\mathsf{Recv}_l(pt)] \cdot \mathrm{rest}$
   - $v \in (vars(pt) \cap MVar) \setminus \bigcup_{s \in \mathrm{done}} vars(s)$
   - $well\text{-}typed_{ty}(tr, th, \sigma)$

   imply
$$\sigma(v, i) \in [\![ty(R, v)]\!]^i_{(tr \cdot [\, \mathsf{St}(i, \mathsf{Recv}_l(pt)) \,],\, th,\, \sigma)} \ .$$

We illustrate the use of this theorem in the following example.

Example 11 (Soundness of the CR'_{ty} assertion)**.** Recall the typing

$$\begin{aligned} CR'_{ty} := \{\, &(C', c) \mapsto \mathsf{Ag}, (C', s) \mapsto \mathsf{Ag}, \\ &(S', c) \mapsto \mathsf{Ag}, (S', s) \mapsto \mathsf{Ag},\ (S', v) \mapsto (C'.k \cup \mathsf{kn}(S'_1)) \,\} \ . \end{aligned}$$

We prove that the protocol CR' is well-typed with respect to this typing.

Proof. Applying Theorem 2 yields five cases. The first four cases deal with the agent variables of the roles C' and S' and are trivial. The fifth case deals with the instantiation of variable v in step (i, S'_1) of some thread i executing the S' role.

We use the assumption $inst_{\sigma,i}(\{\!|\, c, v \,|\!\}_{\mathsf{pk}(s)}) \in knows(tr)$ provided by Theorem 2 to show that

$$\sigma(v, i) \in [\![C'.k \cup \mathsf{kn}(S'_1)]\!]^i_{(tr \cdot [\, \mathsf{St}(i, S'_1) \,],\, th,\, \sigma)}$$

holds for every state $(tr, th, \sigma) \in reachable(CR')$ satisfying $well\text{-}typed_{CR'_{ty}}(tr, th, \sigma)$.

After applying the $\textsc{EChain}_{CR'}$ rule given in Figure 4.7 to

$$\{\!|\, \sigma(c, i), \sigma(v, i) \,|\!\}_{\mathsf{pk}(\sigma(s,i))} \in knows(tr)$$

and dropping some conjuncts that are not required in the remainder of the proof, we are left with the following conclusion.

$$\begin{aligned} &\textbf{(1)} \quad (\sigma(c, i), \sigma(v, i)) \prec_{tr} \{\!|\, \sigma(c, i), \sigma(v, i) \,|\!\}_{\mathsf{pk}(\sigma(s,i))} &&) \vee \\ &\textbf{(2)} \quad (\exists j.\ role_{th}(j) = C' \wedge \{\!|\, \sigma(c, j), k \sharp j \,|\!\}_{\mathsf{pk}(\sigma(s,j))} = \{\!|\, \sigma(c, i), \sigma(v, i) \,|\!\}_{\mathsf{pk}(\sigma(s,i))})) \vee \\ &\textbf{(3)} \quad (\exists j.\ role_{th}(j) = S' \wedge \sigma(v, j) \in [\![C'.k \cup \mathsf{kn}(S'_1)]\!]^j_{(tr, th, \sigma)} \\ &\qquad\qquad \wedge \{\!|\, \sigma(v, j) \,|\!\}_{\mathsf{pk}(\sigma(c,j))} = \{\!|\, \sigma(c, i), \sigma(v, i) \,|\!\}_{\mathsf{pk}(\sigma(s,i))} &&) \end{aligned}$$

Case **(1)** states the adversary could have faked the received message. This case is covered by the $\mathsf{kn}(S'_1)$ part of v's type, as we have $\sigma(v, i) \in knows(tr)$ using rules $\textsc{Ord}_2$ and $\textsc{Known}$. Hence, $\sigma(v, i) \in [\![\mathsf{kn}(S'_1)]\!]^i_{(tr \cdot [\, \mathsf{St}(i, S'_1) \,],\, th,\, \sigma)}$, as

$$\sigma(v, i) \prec_{tr \cdot [\, \mathsf{St}(i, S'_1) \,]} (i, S'_1) \iff \sigma(v, i) \in knows(tr) \ .$$

$$\frac{\{\!|\, m \,|\!\}_x \in \mathit{knows}(tr) \qquad \mathit{well\text{-}typed}_{CR'_{ty}}(tr, th, \sigma)}{\begin{array}{l}(m \prec_{tr} \{\!|\, m \,|\!\}_x \wedge x \prec_{tr} \{\!|\, m \,|\!\}_x) \vee \\ (\exists j.\ \mathit{role}_{th}(j) = C' \wedge (j, C'_1) \prec_{tr} \{\!|\, \sigma(c,j), k \sharp j \,|\!\}_{\mathsf{pk}(\sigma(s,j))} = \{\!|\, m \,|\!\}_x \\ \qquad\qquad \wedge \sigma(c,j) \in \mathit{Agent} \wedge \sigma(s,j) \in \mathit{Agent} \qquad\qquad) \vee \\ (\exists j.\ \mathit{role}_{th}(j) = S' \wedge (j, S'_2) \prec_{tr} \{\!|\, \sigma(v,j) \,|\!\}_{\mathsf{pk}(\sigma(c,j))} = \{\!|\, m \,|\!\}_x \\ \qquad\qquad \wedge \sigma(c,j) \in \mathit{Agent} \wedge \sigma(v,j) \in [\![\, C'.k \cup \mathsf{kn}(S'_1) \,]\!]^i_{(tr,th,\sigma)})\end{array}} \text{ECHAIN}_{CR'}$$

Figure 4.7.: The $\text{ECHAIN}_{CR'}$ rule derived by instantiating the TYPCHAIN rule with an arbitrary encryption, the CR' protocol, and the CR'_{ty} typing

Case **(2)** states that the received message could have been sent by a client thread j. This case is covered by the $C'.k$ part of v's type, as we have

$$\mathit{role}_{th}(j) = C' \ \wedge\ \sigma(v,i) = k \sharp j$$

due to the injectivity of $\{\!|\, \_ \,|\!\}_\_$ and pairing.

Case **(3)** states that a server thread j could have sent the message matching the received message in his second step. In a typed model, this case would be impossible, as the types of the sent and received messages do not match. In our untyped model, a similar argument applies. However, it involves an additional case, as the types may not match due to the interaction with the adversary.

Formally, we have $\sigma(v,j) = (\sigma(c,i), \sigma(v,i))$ due to the injectivity of $\{\!|\, \_ \,|\!\}_\_$. Given this equality, we have

$$(\sigma(c,i), \sigma(v,i)) \in [\![\, C'.k \cup \mathsf{kn}(S'_1) \,]\!]^j_{(tr,th,\sigma)} \ .$$

We have $(\sigma(c,i), \sigma(v,i)) \in [\![\mathsf{kn}(S'_1)]\!]^j_{(tr,th,\sigma)}$ because $[\![\, C'.k \,]\!]^j_{(tr,th,\sigma)}$ does not contain any pairs. Unfolding the definition of $[\![\_]\!]$ yields

$$(\sigma(c,i), \sigma(v,i)) \prec_{tr} (j, S'_1) \ ,$$

which implies $\sigma(v,i) \in \mathit{knows}(tr)$ due to the rules ORD_2 and KNOWN. Hence, $\sigma(v,i) \in [\![\mathsf{kn}(S'_1)]\!]^i_{(tr[\,\mathsf{St}(i,S'_1)\,],\, th,\, \sigma)}$, which concludes the proof of Case **(3)**.

Thus, the CR' protocol is well-typed with respect to the typing CR'_{ty}. ♠

4.4. Discussion of Decryption-Chain Reasoning

We call the proof strategy that we described in this chapter *decryption-chain reasoning*. Decryption-chain reasoning suffices for verifying many security protocols, we give examples in Section 5.2.5 and in Chapter 6.

In contrast to other security protocol verification methods, decryption-chain reasoning does not require a typed model. Nevertheless, the notion of types plays an important

role in decryption-chain reasoning. Types yield a uniform construction of protocol-specific invariants (i.e., well-typedness with respect to a protocol-specific typing) that are strong enough to reason about messages of unbounded size. Said differently, decryption-chain reasoning illustrates that we can shift the notion of types from being an a-priori assumption on the semantics of security protocol execution to serving as a powerful tool for constructing security proofs.

Nevertheless, decryption-chain reasoning is not a silver bullet. It may fail in two ways. (1) A protocol may not be typeable; i.e., there does not exist a typing with respect to which the protocol is well-typed. (2) The case distinctions provided by the TypChain rule are too weak to prove the security property of interest.

Problem (1) is inherent to all approaches based on types. It must be solved per protocol by extending the set of types such that all reachable states are described. Afterwards, our approach of using the $\mathsf{kn}(\_)$ type constructor to describe the adversary interaction can be applied again.

Problem (2) exists as, even for typeable protocols, decryption-chain reasoning is necessarily incomplete. If it were complete, we could then decide the secrecy problem for typeable protocols because we could enumerate both proofs as well as attacks. This would contradict the undecidability of the secrecy problem with unbounded sessions and nonces [DLM04], as the proof also applies to typeable protocols.

The following example illustrates the incompleteness of decryption-chain reasoning for a typeable protocol.

Example 12 (No progress)**.** Consider the (artificial) protocol $P := \{I, R\}$, where

$$I := [\mathsf{Send}_1(\{\!|\, n, n \,|\!\}_{\mathsf{k}(a,b)})] \qquad R := [\mathsf{Recv}_1(\{\!|\, v, w \,|\!\}_{\mathsf{k}(a,b)}), \mathsf{Send}_2(\{\!|\, v, n' \,|\!\}_{\mathsf{k}(a,b)})]$$

and $n, n' \in \mathit{Fresh}$, $v \in \mathit{MVar}$, and $a, b \in \mathit{AVar}$. We can show that a and b have type Ag, v has type $I.n \cup \mathsf{kn}(R_1)$, and w has type $I.n \cup R.n' \cup \mathsf{kn}(R_1)$. Moreover, the contents of variable v in role R are obviously secret, provided that both a and b are uncompromised. However, we cannot prove the above secrecy property using decryption-chain reasoning.

The problem in this example is that the message sent in step R_2 can be received in step R_1. Hence, there can be an *unbounded* chain of threads executing the R role where each thread receives the message that the previous thread sent. This unbounded chain of threads manifests itself during proof construction as follows. At some point in the proof construction, the only inference step left is to apply the TypChain rule to determine the possible origin of the message m that a thread i executing the R role receives in its first step. After this application, we are left with a case stating that m was sent in the second step of some other thread j also executing the R role. In this case, our proof makes no progress, as we know as much about thread j as we already knew about thread i before the case distinction. Said differently, the proof of the authenticity of the message m that is sent in the second step of some role R depends on itself, when considering the limited perspective of decryption-chain reasoning. ♠

Note that we can resort to induction over the reachable states to reason about protocols like the one above. We can then use decryption-chain reasoning for the individual induction steps. Moreover, the above problem is of a theoretical nature. Many practical protocols have an intended message flow that is acyclic and ensure that this intended message flow is achieved, e.g., using tagging.

Example 13 (Ensuring an acyclic message flow)**.** We can ensure an acyclic message flow for protocol P from Example 12 by redefining $P := \{I, R\}$ as

$$I := [\mathsf{Send}_1(\{\!|\, 1, n, n \,|\!\}_{\mathsf{k}(a,b)})] \qquad R := [\mathsf{Recv}_1(\{\!|\, 1, v, w \,|\!\}_{\mathsf{k}(a,b)}), \mathsf{Send}_2(\{\!|\, 2, v, n' \,|\!\}_{\mathsf{k}(a,b)})]$$

for $1, 2 \in \mathit{Const}$. After inserting these tags, decryption-chain reasoning works without resorting to induction over the reachable states. ♠

5. Machine-Checked Security Proofs

In this chapter, we present two approaches for constructing machine-checked security proofs. The first approach uses Isabelle/HOL to interactively construct the corresponding proof script using our verification theory. The second approach uses an algorithm to automatically generate the corresponding Isabelle/HOL proof script.

Where applicable we use the automatic approach to generate the security proofs, otherwise we resort to interactively constructing them. The key benefit of the interactive approach is its versatility. Martin Schaub for example uses our interactive approach in his master's thesis [Sch11] to reason about security properties in the context of dynamically compromising adversaries [BC10]. We explain the interactive approach in Section 5.1 and the automatic approach in Section 5.2.

5.1. Interactive Proof Construction

To simplify the interactive construction of security proofs, we extended Isabelle's proof language [WPN08] with commands to define roles, protocols, and type assertions as well as a tactic that automates the application of the TypChain rule.

The commands to define roles and protocols introduce corresponding constants and set up Isabelle's automation infrastructure to simplify reasoning about the steps and roles of the protocol. The command to define a type assertion automatically derives the simplified instances of the TypChain rule under the assumption that this type assertion is sound for the given protocol.[1] Provided that this soundness assumption holds, it can be discharged in an interactive proof by applying Theorem 2 and using decryption-chain reasoning for the resulting proof obligations, i.e., the theorem's premises.

Case distinctions on the possible origins of a message $m \in knows(tr)$ are automated using the tactic "sources". A call "sources m" selects the simplified TypChain rule instance corresponding to the outermost constructor of the message m and uses it to enumerate the possible origins of m. The resulting cases are discharged automatically using Isabelle's built-in tools, if possible. Otherwise, they are named and presented to the user for further processing.

The following example and the proofs in our case studies [Mei12], show that our mechanization of decryption-chain reasoning allows for succinct, machine-checkable security proofs.

Example 14. The session-key secrecy proof given in Example 6 corresponds to the proof script given in Figure 5.1, which is checked by Isabelle in under a second. Note that we have taken minor liberties in pretty-printing to improve readability.

[1] The type assertion soundness assumptions and the simplified TypChain rule instances are managed using Isabelle's *locale* infrastructure [Bal06].

```
1:  lemma (in CR-state) client-k-secrecy:
2:    assumes
3:      "role_th(i) = C"
4:      "σ(AV(''s''), i) ∉ Compr"
5:      "NO(''k'', i) ∈ knows(tr)"
6:    shows "False"
7:  proof(sources "NO(''k'', i)")
8:    case C_1-k thus "False"
9:    proof(sources "SK (σ(AV(''s''), i))")
10:     case ik0 thus "False" by auto
11:   qed
12: qed
```

Figure 5.1.: Proof script of session-key secrecy for the *CR* protocol.

Line 1 begins the lemma, named "client-k-secrecy". The statement "(**in** CR-state)" expresses that this lemma is proven under the assumption that (tr, th, σ) is a reachable state of the *CR* protocol. Lines 2–6 state the secrecy property. The expression NO(''k'', i) denotes the freshly generated message $k\sharp i$. The expression AV(''s'') denotes the agent variable s.

Lines 7–12 give the proof, which has the same structure as the pen-and-paper proof from Example 6. Isabelle supports its interactive construction as follows. After parsing the lemma's statement, Isabelle prints all assumptions and the goal that we must prove. There is only one assumption to which we can apply the TypChain rule[2], which we do in Line 7. Isabelle computes the resulting non-trivial cases and prints them together with their additional assumptions (i.e., what agents are compromised, equalities between messages, what events happened, the order between events, and the roles of the involved threads). Only one non-trivial case, C_1-k, results from this application of the TypChain rule. We select it in Line 8 and state that it is contradictory; i.e., we can prove "False" from its assumptions. We prove "False" by applying the TypChain rule to "SK (σ(AV(''s''), i))", the private key of the server that thread i is communicating with. The only non-trivial case is "ik0", which states that the server "σ(AV(''s''), i)" was compromised. This case contradicts the assumption in Line 4. We use Isabelle's built-in tactic "auto" to prove this. ♠

In Appendix A, we also provide Isabelle formalizations of the authentication proof from Example 7 and the type assertion soundness proof from Example 11. In both cases, our extension of Isabelle's proof language allows for succinct formalizations highlighting the main argument underlying these proofs. This succinctness is one of the key properties of decryption-chain reasoning. Other examples of this are the security proofs (available from [Mei12]) that we interactively constructed for the Yahalom [Pau01], the Kerberos V [Bel07], and the TLS handshake [Pau99] protocols based on the models developed using the Inductive Method [Pau98]. Modeling each protocol took under an hour. Proving the security properties took 1.5 hours for Yahalom, 2 hours for Kerberos V,

[2] We can use the TypChain rule because we prove after the definition of the *CR* protocol that it is well-typed with respect to the typing $\{(C, s) \mapsto \mathsf{Ag}, (S, s) \mapsto \mathsf{Ag}, (S, v) \mapsto Ck \cup \mathsf{kn}(S_1)\}$.

and 2.5 hours for the TLS handshake protocol. These times represent roughly a two orders of magnitude improvement over the times reported by Paulson using his Inductive Method [Pau98, Pau99, Pau01].

5.2. Automatic Proof Generation

We now describe an algorithm for automatically generating Isabelle/HOL proof scripts for secrecy and authentication properties of security protocols. The input of the algorithm is a protocol $P \in \mathit{Protocol}$, a set of type assertions ty, and a security property φ. If the algorithm succeeds in proving the soundness of the type assertions ty and the validity of φ, then it outputs an Isabelle proof script. The script contains the specification of the protocol P, the type assertions ty and its soundness proof, and a lemma stating φ and its proof.

Our algorithm uses symbolic messages and variables for thread identifiers as part of its proof state representation. Symbolic messages are built from the message constructors given in Section 3.2.1, the uninterpreted function $\sigma : \mathit{Var} \times \mathit{TID} \to \mathit{Msg}$ denoting a variable store, and the function $\_^{-1} : \mathit{Msg} \to \mathit{Msg}$ denoting symbolic key inversion. In the following sections, we use i and j (possibly primed) to denote thread identifier variables, a to denote symbolic agent variables, m to denote symbolic messages, s to denote role-steps, R to denote roles, and e to denote events built from symbolic messages. Our algorithm handles security properties that can be represented as closed formulas of the form

$$\forall (\mathit{tr}, \mathit{th}, \sigma) \in \mathit{reachable}(P).\ \forall i_1 \ldots i_l.\ (\textstyle\bigwedge_{A \in \Gamma} A) \Rightarrow \exists i_{l+1} \ldots i_n.\ (\textstyle\bigwedge_{B \in \Delta} B)$$

where $0 \le l \le n$ and Γ, Δ are sets of *atoms* of the following form.

$i = i'$	$m = m'$	$\mathit{role}_{\mathit{th}}(i) = R$	$\sigma(a, i) \in \mathit{Compr}$	false
$e \prec_{\mathit{tr}} e'$	$(i, s) \in \mathit{steps}(\mathit{tr})$	$m \in \mathit{knows}(\mathit{tr})$	$\sigma(a, i) \notin \mathit{Compr}$	

We abbreviate such a security property as a *judgment* $\Gamma \vdash_P \Delta$, where implicitly all thread identifier variables in Γ are universally quantified and all thread identifier variables in Δ that do not occur in Γ are existentially quantified.

Example 15. The secrecy property φ_{sec} from Example 3 is represented as

$$\mathit{role}_{\mathit{th}}(i) = C,\ \sigma(s, i) \notin \mathit{Compr},\ k \sharp i \in \mathit{knows}(\mathit{tr}) \vdash_{\mathit{CR}} \mathit{false}.$$

The authentication property φ_{auth} from the same example is represented as

$$\begin{array}{r} \mathit{role}_{\mathit{th}}(i) = C, \\ \sigma(s, i) \notin \mathit{Compr}, \\ (i, C_2) \in \mathit{steps}(\mathit{tr}) \end{array} \quad \vdash_{\mathit{CR}} \quad \begin{array}{l} \mathit{role}_{\mathit{th}}(j) = S,\ \sigma(s, i) = \sigma(s, j),\ k \sharp i = \sigma(v, j), \\ (i, C_1) \prec_{\mathit{tr}} (j, S_1),\ (j, S_2) \prec_{\mathit{tr}} (i, C_2). \end{array}$$

♠

We present our algorithm in four steps. First, we describe the core proof generation algorithm, which mechanizes the use of the TypChain rule assuming the soundness of the given type assertions. Second, we describe an algorithm for proving the soundness of

```
1: procedure CorePrfGen(Γ ⊢_P Δ, ty)
2:     solve all equality premises of Γ ⊢_P Δ
3:     saturate Γ under all rules except TypChain
4:     if Γ ⊢_P Δ is trivially valid then
5:         print "by auto"
6:     else
7:         if there is no new (m ∈ knows(tr)) ∈ Γ then
8:             fail (possible attack found: Γ ⊢_P Δ)
9:         else
10:            select new (m ∈ knows(tr)) ∈ Γ
11:            print "proof(sources m)"
12:            J ← apply TypChain to m ∈ knows(tr) and well-typed_ty(tr, th, σ)
13:            for each J ∈ J do
14:                print "case nameOf(J)"
15:                CorePrfGen(J, ty)
16:            end for
17:            print "qed"
18:        end if
19:    end if
20: end procedure
```

Figure 5.2.: The core proof generation algorithm CorePrfGen

type assertions based on Theorem 2 and the core proof generation algorithm. We then show how to combine these two algorithms in our proof generation algorithm. Third, we explain how we extend this algorithm to prove injective two-party authentication properties using Theorem 1. Fourth, we describe two further extensions that increase the scope and efficiency of our proof generation algorithm as well as the readability of the generated proofs. Afterwards, we present experimental results that we obtained using our `scyther-proof` tool, which implements our algorithm and all its extensions.

5.2.1. Core Algorithm

The core proof-generation algorithm CorePrfGen is given in Figure 5.2. Given a judgment $\Gamma \vdash_P \Delta$ and a set of type assertions ty, this algorithm tries to prove the validity of $\Gamma \vdash_P \Delta$ under the assumption that the type assertions ty are sound with respect to the protocol P as follows.

First, the equalities between symbolic messages, thread identifiers, and roles in Γ are solved using equational unification. We exploit that all symbolic messages with an outermost $\mathsf{k_{bi}}(\_)$ constructor are of the form $\mathsf{k_{bi}}(\{a, b\})$, as we use this constructor only to model bidirectional long-term symmetric keys. The equational theory that we consider respects the equivalence $(\mathsf{k_{bi}}(\{a, b\}) = \mathsf{k_{bi}}(\{c, d\})) \Leftrightarrow (\{a, b\} = \{c, d\})$ and the definition of key inversion given in Section 3.2.1. It moreover regards $\sigma : \mathit{Var} \times \mathit{TID} \to \mathit{Msg}$ as an uninterpreted function and $\sharp : \mathit{Fresh} \times \mathit{TID} \to \mathit{Msg}$ as a free constructor for symbolic messages.

For example, the symbolic equation $x^{-1} = \mathsf{h}(y)$ is solved with the substitution

$\{x \mapsto \mathsf{h}(y)\}$, the equation $((x^{-1})^{-1})^{-1} = \mathsf{sk}(a)$ is solved with $\{x \mapsto \mathsf{pk}(a)\}$, and the equation $x \sharp i = x' \sharp i'$ is solved with $\{x \mapsto x', i \mapsto i'\}$. The equation $\mathsf{k_{bi}}(\{a, b\}) = \mathsf{k_{bi}}(\{c, d\})$ is solved using a case distinction to distinguish between its two solutions $\{a \mapsto c, b \mapsto d\}$ and $\{a \mapsto d, b \mapsto c\}$. The equation $x \sharp i = \sigma(x', i')$ is solved with the substitution $\{\sigma(x', i') \mapsto x \sharp i\}$. We require such non-standard substitutions because we regard σ as an uninterpreted function, whose function values are therefore regarded as unknowns during unification.

Note that our non-standard construction of symbolic messages stems from our decision to formalize them in Isabelle/HOL using a shallow embedding; i.e., we define $^{-1}$, $\sharp$, and σ over ground messages only and use HOL terms and variables to represent symbolic messages. This simplifies the formalization of symbolic messages in Isabelle/HOL, but complicates the corresponding meta-theoretic results. In retrospect, we prefer the much more straightforward formalization of symbolic messages used in Part II of this thesis, which is based on standard notions from equational term rewriting.

If unification fails, then $\Gamma \vdash_P \Delta$ is trivially valid because there are contradicting equalities in Γ. This case is handled by the trivial validity check in Line 4. If unification succeeds, the resulting substitution of thread identifiers and messages is applied to $\Gamma \vdash_P \Delta$. Second, the premises Γ are saturated by extending them with all atoms derivable using inference rules other than the TypChain rule. Third, the resulting judgment $\Gamma \vdash_P \Delta$ is checked for *trivial validity*; namely, whether one of the following holds:

1. $\mathit{false} \in \Gamma$,
2. $e \prec_{tr} e \in \Gamma$,
3. $(x \in \mathit{Compr}) \in \Gamma$ and $(x \notin \mathit{Compr}) \in \Gamma$, or
4. there exists a substitution τ of the existentially quantified thread identifiers in Δ such that $\tau(\Delta) \setminus \Gamma$ consists of reflexive equalities only.

If the judgment $\Gamma \vdash_P \Delta$ is trivially valid, then Isabelle can prove the validity of $\Gamma \vdash_P \Delta$ using its built-in tactic "auto". Otherwise, the algorithm checks whether there exists an atom $(m \in \mathit{knows}(tr)) \in \Gamma$ that has not yet been selected. If no such atom exists, proof-generation fails and the corresponding proof state indicates a possible attack, as we explain in the next paragraph. Otherwise, we pick the next atom and apply the TypChain rule to it. In our case studies, we use a simple heuristic that first picks atoms with messages containing long-term keys, then atoms with messages containing nonces supposedly known to the adversary, and finally atoms that have been in the proof state for the longest time. The application of the TypChain rule to the selected atom $m \in \mathit{knows}(tr)$ results in a case distinction on how the adversary learned m. Each case is represented again as a judgment J of the form $\Gamma \cup \Sigma \vdash_P \Delta$, where Σ are the new assumptions introduced by the case that J represents. For each case, we output the information necessary for Isabelle to know which case is being proven and generate the corresponding proof script by recursively calling CorePrfGen.

Despite the undecidability of protocol security, the CorePrfGen algorithm terminates for many practical protocols, including all the case studies from Table 5.1 on page 52. Moreover, analogous to the algorithm underlying Scyther [Cre08b], the failure to generate a proof often indicates an attack. In particular, for secrecy properties written

```
1: procedure TypePrfGen(P, ty)
2:     J ← proof obligations of Theorem 2 showing that
3:             the type assertion ty is sound for P
4:     for each J ∈ J do
5:         CorePrfGen(J, ty)
6:     end for
7: end procedure
```

Figure 5.3.: The TypePrfGen algorithm for generating type assertion soundness proofs

```
1: procedure PrfGen(Γ ⊢_P Δ, ty)
2:     TypePrfGen(P, ty)
3:     CorePrfGen(Γ ⊢_P Δ, ty)
4: end procedure
```

Figure 5.4.: The proof generation algorithm PrfGen

as a judgment $\Gamma \vdash_P \mathit{false}$ that only mentions roles, role-steps, nonces, and variables of the protocol P, a failure to generate a proof always indicates an attack. An attack on such a property is a reachable state that satisfies the premises Γ. The only non-trivial constraints on such a state stem from the $m \in \mathit{knows}(tr)$ premises in Γ. Each case of an application of the TypChain rule provides an explanation of how the adversary learned m. Once we reach a non-contradictory set of premises Γ' without any remaining unsolved $m \in \mathit{knows}(tr)$ premises, we therefore know how to construct a reachable state satisfying Γ'; i.e., we construct such a state by choosing some instantiation of all variables in Γ' and executing the protocol steps and message deductions in Γ' according to their dependencies in Γ'. Analogously, we can construct attacks from failed proofs of authentication properties, provided they can be written as a judgment $\Gamma \vdash_P \Delta$ that only mentions roles, role-steps, nonces, and variables of the protocol P.

5.2.2. Generating Type Assertion Soundness Proofs

The TypePrfGen algorithm given in Figure 5.3 generates soundness proofs for type assertions. It uses an extended version of the CorePrfGen algorithm that also allows judgments of the form

$$\Gamma \vdash_P \sigma(v, i) \in [\![\alpha]\!]^i_{(tr\cdot[\,\mathsf{St}(i,\mathsf{Recv}_l(pt))\,],\, th,\, \sigma)} \ ,$$

where $\alpha \in \mathit{Type}$ and Γ is a set of atoms as before. These judgments are sufficient to represent the proof obligations stemming from applications of Theorem 2 given on page 39. The $[\![\alpha]\!]^i_{(\ldots)}$ expressions are handled by unfolding the definition of $[\![\_]\!]$ for the given type α and the trivial validity of judgments is redefined accordingly. Note that Theorem 2 provides $\mathit{well\text{-}typed}_{ty}(tr, th, \sigma)$ as an induction hypothesis. This justifies our use of CorePrfGen in Line 5, which assumes $\mathit{well\text{-}typed}_{ty}(tr, th, \sigma)$.

The PrfGen algorithm given in Figure 5.4 combines the previous two algorithms. Analogous to CorePrfGen, a failure of the PrfGen algorithm provides useful information about the protocol under investigation. If the type assertion soundness proof

fails, then there are other variable instantiations possible than the ones specified by the type assertions. These instantiations might be exploited in a type-flaw attack. If proving the judgment $\Gamma \vdash_P \Delta$ fails, then there is a possible attack on the security property being verified, as explained in the previous section.

5.2.3. Proving Injective Two-Party Authentication Properties

We extend our algorithm with support for proving injective two-party authentication properties as follows. We first observe that every judgment $\Gamma \vdash_P \Delta$ where i is the only free variable in Γ and i and j are the only free variables in Δ formalizes a non-injective two-party authentication property, i.e.,

$$\forall q \in \mathit{reachable}(P).\ \forall i \in \mathit{TID}.\ (\bigwedge_{A \in \Gamma} A) \Rightarrow \exists j \in \mathit{TID}.\ (\bigwedge_{B \in \Delta} B)\ .$$

The extended algorithm uses Theorem 1 given on page 27 to prove the corresponding injective two-party authentication property. It uses the PrfGen algorithm to prove the corresponding non-injective property denoted by $\Gamma \vdash_P \Delta$ and verifies the injectivity property by checking whether

$$(\bigwedge_{B \in \Delta} B\{i_1/i\}) \wedge (\bigwedge_{B \in \Delta} B\{i_2/i\}) \quad \text{implies} \quad i_1 = i_2\ ,$$

where $B\{y/x\}$ denotes the substitution of all occurrences of x in B with y.

5.2.4. Lemma Instantiation and Minimal Proof Generation

We now describe two further extensions of our proof generation algorithm: lemma instantiation and minimal proof generation. The aim of these two extensions is to reduce the size of the generated proofs, which in turn improves proof checking time as well as human readability. Moreover, lemma instantiation also extends the scope of our algorithm, as we explain below.

Lemma instantiation shortens proofs by referring to already proven security properties. The idea is to instantiate the universally quantified thread variables of a security property modeled as a judgment such that the judgment's premises are satisfied in the current proof state and its conclusions can therefore be added to the proof state. Lemma instantiation is crucial for sharing subproofs between different cases of a proof. We can see this in the authentication proof from Example 7, where we instantiate the previously proven secrecy property twice. Without lemma instantiation, we are forced to prove the secrecy of the exchanged session key twice. Indeed it is not difficult to construct protocols where lemma instantiation even results in exponentially smaller proofs.

For some security protocols, we also use lemma instantiation to model additional assumptions (axioms) about a protocol's execution. A typical example is the assumption that a certificate authority's long-term private key is never compromised. We model such an assumption by assuming the corresponding security property instead of proving it. We can then exploit this property in subsequent proofs using lemma instantiation. This construction extends the scope of our algorithm. It allows us to verify properties of the form

$$\left(\bigwedge_{1 \le i \le n} (\Gamma_i \vdash_P \Delta_i)\right) \Rightarrow (\Gamma \vdash_P \Delta)$$

	Protocol	generation	checking
1	Amended Needham Schroeder Symmetric Key	0.25	25
2	Lowe's fixed Needham-Schroeder Public Key	0.14	16
3	Paulson's strengthened version of Yahalom	0.11	20
4	Lowe's modified Denning-Sacco Shared Key	0.12	23
5	BAN modified version of CCITT X.509 (3)	0.56	12
6	Lowe's modified BAN Concrete Andrew Secure RPC	0.01	10
7	Woo and Lam Pi 3	0.02	9
8	Kerberos V	1.67	101
9	Kerberos IV	8.76	175
10	TLS Handshake	0.84	33

Table 5.1.: Timings in seconds for generating and checking minimal proofs.

where the judgments $\Gamma_i \vdash_P \Delta_i$ model n additional assumptions about the execution of the security protocol P.

We implement lemma instantiation in the CorePrfGen algorithm using resolution of judgments. Assume given a lemma $\Gamma_1 \vdash_P \Delta$ with no existentially quantified thread identifiers in Δ. When proving a judgment of the form $\Gamma_1 \cup \Gamma_2 \vdash_P \Pi$, we can exploit this lemma using the following resolution rule.

$$\frac{\Gamma_1 \vdash_P \Delta \qquad \Delta \cup \Gamma_1 \cup \Gamma_2 \vdash_P \Pi}{\Gamma_1 \cup \Gamma_2 \vdash_P \Pi}$$

The thread identifier variables in $\Gamma_1 \vdash_P \Delta$ may need to be renamed for this rule to apply, which can be done as these variables are all universally quantified. Note that lemma instantiation is especially useful when $\Delta = \mathit{false}$; e.g., secrecy properties have this form. In this case, the second premise of the resolution rule becomes trivial, as $\mathit{false} \cup \Gamma_1 \cup \Gamma_2 \vdash_P \Pi$ always holds.

We also implement the generation of *minimal proofs*, which are proofs with a minimal number of TypChain rule applications. In a minimal proof every case distinction is required. This makes minimal proofs especially well-suited for human understanding, as every case distinction conveys information about how the protocol achieves the security property. We generate minimal proofs using a branch-and-bound strategy to minimize the number of case distinctions in the generated proofs. Moreover, we instantiate lemmas eagerly to ensure that their consequences are available before making a further case distinction. We remove superfluous lemma instantiations after a minimal proof is found, thereby further improving its readability. See Appendix A for two examples of minimal proofs generated using our algorithm.

5.2.5. Experimental Results

Table 5.1 shows part of the experimental results we obtained using our `scyther-proof` tool, which implements the PrfGen algorithm and its extensions. In Chapter 6, we also use the `scyther-proof` tool to construct machine-checked proofs of our repaired

versions of all 17 protocols of the ISO/IEC 9798 standard for entity authentication. All our protocol models are distributed together with the source code of the tool [Mei12].

Protocols 1-8 from Table 5.1 are modeled based on to their description in the SPORE [SPO05] security protocol library. Protocols 9 and 10 are based on the corresponding models that were verified by Bella and Paulson using the Inductive Method [BP98, Pau99]. The times for proof generation and checking were measured using Isabelle2009-1 on an Intel i7 Quad Core laptop with 4GB RAM. For all protocols, we prove non-injective agreement of all shared data for all roles of the protocol where possible and secrecy for the freshly generated keys and payloads, if they are present. For protocols 8 and 10, we additionally prove non-injective synchronization [CMdV06], which is a strengthened variant of non-injective agreement.

Our case studies demonstrate that our implementation of decryption-chain reasoning efficiently constructs machine-checked security proofs for complex protocols like Kerberos or the TLS handshake. On average 2.5 applications of the TypChain rule are required to prove a security property in our case studies. The comparatively high generation time for Kerberos IV is due to generating minimal proofs. It suggests that our minimization-strategy can be improved. The comparatively high checking times for the Kerberos protocols are due to the number of authentication proofs (one for each of the four roles) and the complexity of these proofs (each proof needs several nested case distinctions to determine the ordering of the individual steps). However, these times are definitely fast enough for machine-checking the security proofs once a protocol design is finished.

Most variables of the protocols in our case studies have a simple type: either Ag, $\mathsf{kn}(s)$, or $\mathsf{kn}(s) \cup R.n$, for the receive role-step s where the variable is instantiated and the nonce n that is sent in the corresponding send role-step in role R. We use a simple heuristic to infer such types by matching the receive role-steps with their corresponding send role-steps. Our heuristic suffices to infer the types of 115 out of the 118 variables of the protocols from Table 5.1. It fails only for the three variables inside an encryption that are instantiated with composed terms in Protocol 1 (Amended NS) and Protocol 9 (Kerberos IV). We specified their type by hand. For example, the client of the Kerberos IV protocol receives the authentication ticket sent by the Authentication server inside an encryption. The type of the corresponding variable is

$$\mathsf{kn}(C_2) \cup \{\!|\, \mathsf{kn}(C_2), \mathsf{Ag}, A.\mathit{AuthKey}, A.\mathit{Ta} \,|\!\}_{\mathsf{k_{un}}} \ ,$$

where C_2 is the second step of the client role, which receives the authentication ticket, A is the Authentication server role, $\mathit{AuthKey}$ is the freshly generated authentication key, and Ta is a nonce that we use to model the timestamp of the authentication ticket. The innermost $\mathsf{kn}(C_2)$ type stems from a variable containing the client's name, which is received by the Authentication server as plaintext and could therefore be any message known to the adversary.

6. Provably Repairing the ISO/IEC 9798 Standard for Entity Authentication

In this chapter, we illustrate the use of the Scyther and `scyther-proof` tools to analyze and provably repair the ISO/IEC 9798 Standard for Entity Authentication. Our work demonstrates that the formal analysis of an actual security protocol standard is not only feasible, but also desirable. Based on our analysis, the ISO/IEC working group responsible for the 9798 standard will release an updated version of the standard, incorporating our proposed fixes.

This chapter is organized as follows. In Section 6.1, we describe the standard. In Section 6.2, we model the protocols and analyze them using the Scyther tool, discovering numerous weaknesses. In Section 6.3, we analyze the sources of these weaknesses and present two design principles that eliminate them. In Section 6.4, we use the `scyther-proof` tool to automatically generate machine-checked correctness proofs for these repaired protocols.

6.1. The ISO/IEC 9798 Standard

6.1.1. Overview

Entity authentication is a core building block for security in networked systems. In its simplest form, entity authentication boils down to establishing that a party's claimed identity corresponds to its real identity. In practice, stronger guarantees are usually required, such as mutual authentication, agreement among the participating parties on the identities of their peers, or authentication of transmitted data.

The ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) jointly provide standards for Information Technology. Their standard 9798 specifies a family of entity authentication protocols. This standard is mandated by numerous other standards that require entity authentication as a building block. Examples include the Guidelines on Algorithms Usage and Key Management [Eur09] by the European Committee for Banking Standards and the ITU-T multimedia standard H.235 [ITU03].

Note that the ISO/IEC 9798 standard has already been previously analyzed, which lead to the discovery of several weaknesses [DNL99, AC08, CM10]. It has been revised several times to address these weaknesses and other ambiguities, with the latest updates stemming from 2010. Nevertheless, when we tried to prove the correctness of the ISO/IEC 9798 standard, we not only found that several previously reported weaknesses are still present, but we also found new weaknesses. In particular, many of the protocols guarantee only weak authentication properties and, under some circumstances, even no authentication at all.

Protocol	Description
Part 2: Symmetric-key Cryptography	
9798-2-1	One-pass unilateral authentication
9798-2-2	Two-pass unilateral authentication
9798-2-3	Two-pass mutual authentication
9798-2-4	Three-pass mutual authentication
9798-2-5	Four-pass with TTP
9798-2-6	Five-pass with TTP
Part 3: Digital Signatures	
9798-3-1	One-pass unilateral authentication
9798-3-2	Two-pass unilateral authentication
9798-3-3	Two-pass mutual authentication
9798-3-4	Three-pass mutual authentication
9798-3-5	Two-pass parallel mutual authentication
9798-3-6	Five-pass mutual authentication with TTP, initiated by A
9798-3-7	Five-pass mutual authentication with TTP, initiated by B
Part 4: Cryptographic Check Functions	
9798-4-1	One-pass unilateral authentication
9798-4-2	Two-pass unilateral authentication
9798-4-3	Two-pass mutual authentication
9798-4-4	Three-pass mutual authentication

Table 6.1.: Protocols specified by Parts 1-4 of the standard

In the following, we give a brief overview of the ISO/IEC 9798 standard. We consider here the first four parts of the standard. Part 1 is general and provides background for the other parts. The protocols are divided into three groups. Protocols using symmetric encryption are described in Part 2, those using digital signatures are described in Part 3, and those using cryptographic check functions such as MACs are described in Part 4.

Because the standard has been revised, we also take into account the most recent technical corrigenda and amendments. Our analysis covers the protocols specified by the following documents. For the first part of the standard, we cover ISO/IEC 9798-1:2010 [Int10a]. For the second part, we cover ISO/IEC 9798-2:2008 [Int08] and Corrigendum 1 from 2010 [Int10b]. For the third part, we cover ISO/IEC 9798-3:1998 [Int98], the corrigendum from 2009 [Int09a], and the amendment from 2010 [Int10c]. Finally, for the fourth part, our analysis covers ISO/IEC 9798-4:1999 [Int99] and the corrigendum from 2009 [Int09b]. In this chapter, we write "the standard" to refer to the above documents.

Table 6.1 lists the 17 associated protocols. For each cryptographic mechanism, there are unilateral and bilateral authentication variants. The number of messages and passes differs among the protocols as well as the communication structure. Some of the protocols also use a trusted third party (TTP).

Note that there is no consistent protocol naming scheme shared by the different parts of the ISO/IEC 9798 standard. The symmetric-key based protocols are referred to in [Int08] as "mechanism 1", "mechanism 2", etc., whereas the protocols in [Int98, Int10c, Int09b] are referred to by their informal name, e.g., "One-pass unilateral authentication". In

1. $A \to B: \quad TN_A \parallel Text_2 \parallel f_{K_{AB}}(TN_A \parallel I_B \parallel Text_1)$
2. $B \to A: \quad TN_B \parallel Text_4 \parallel f_{K_{AB}}(TN_B \parallel I_A \parallel Text_3)$

Figure 6.1.: The 9798-4-3 two-pass mutual authentication protocol using a cryptographic check function

this chapter we will refer to the protocols consistently by combining the document identifier, e.g., "9798-2" with a number n to identify the n-th protocol in that document. For protocols proposed in an amendment, we continue the numbering from the base document. Hence we refer to the first protocol in [Int10c] as "9798-3-6". The resulting identifiers are listed in Table 6.1.

Most of the protocols are parametrized by the following elements:

- All text fields included in the protocol specification are optional and their purpose is application-dependent.
- Many fields used to ensure uniqueness or freshness may be implemented either by random numbers, sequence numbers, or timestamps.
- Some protocols specify alternative message contents.
- Some identifier fields may be dropped, depending on implementation details.

6.1.2. Additional Notation

We use the security protocol model from Chapter 3 to formalize the protocols in the standard. To improve the readability of our examples, we introduce the following additional notation. We use the operator $\parallel$ to denote uniquely parseable concatenation, i.e., pairing. We assume this operator to be right-associative. Thus, $X \parallel Y \parallel Z$ denotes $\langle X, \langle Y, Z \rangle \rangle$. We write $\{\!|\, X \,|\!\}_k^{\mathsf{enc}}$ instead of $\{\!|\, X \,|\!\}_k$ to emphasize that X is encrypted with a symmetric key k. The application of a cryptographic check function f, keyed with key k, to a message m, is denoted by $f_k(m)$. Formally, we model this as hashing the constant $\mathtt{ccf} \in \mathit{Const}$ together with the message m and the key k, i.e., as $\mathsf{h}(\mathtt{ccf} \parallel m \parallel k)$.

In the standard, *TVP* denotes a Time-Variant Parameter, which may be a sequence number, a random value, or a timestamp. *TN* denotes a time stamp or sequence number. I_X denotes the identity of agent X. $Text_n$ refers to a text field. These fields are always optional and their use is not specified within the standard. The protocols based on symmetric keys can be used both with unidirectional and bidirectional long-term keys. Where the directionality of these keys is of no concern, we use K_{AB} to denote both the bidirectional and unidirectional long-term symmetric keys shared by A and B. By convention, we use lower case strings for fresh session keys, like *kab*.

6.1.3. Protocol Examples

Example 1: 9798-4-3

The 9798-4-3 protocol is a two-pass mutual authentication protocol based on cryptographic check functions, e.g., message authentication codes. Its design, depicted

$$
\begin{array}{lll}
1. & A \to P: & TVP_A \parallel I_B \parallel Text_1 \\
2. & P \to A: & Token_{PA} \\
3. & A \to B: & Token_{AB} \\
4. & B \to A: & Token_{BA}
\end{array}
$$

where

$Token_{PA} := Text_4 \parallel \{\!|\, TVP_A \parallel kab \parallel I_B \parallel Text_3 \,|\!\}^{\mathsf{enc}}_{K_{AP}} \parallel \{\!|\, TN_P \parallel kab \parallel I_A \parallel Text_2 \,|\!\}^{\mathsf{enc}}_{K_{BP}}$

$Token_{AB} := Text_6 \parallel \{\!|\, TN_P \parallel kab \parallel I_A \parallel Text_2 \,|\!\}^{\mathsf{enc}}_{K_{BP}} \parallel \{\!|\, TN_A \parallel I_B \parallel Text_5 \,|\!\}^{\mathsf{enc}}_{kab}$

$Token_{BA} := Text_8 \parallel \{\!|\, TN_B \parallel I_A \parallel Text_7 \,|\!\}^{\mathsf{enc}}_{kab}$

Figure 6.2.: The 9798-2-5 four pass protocol with TTP using symmetric encryption

in Figure 6.1, is similar to two related protocols based on symmetric key encryption (9798-2-3) and digital signatures (9798-3-3).

The initiator starts in role A and sends a message that consists of a time stamp or sequence number TN_A, concatenated with an optional text field and a cryptographic check value. This check value is computed by applying a cryptographic check function to the key shared between A and B and a string consisting of TN_A, B's identity, and optionally a text field $Text_1$. When B receives this message he computes the cryptographic check himself and compares the result with the received check value. He then computes the response message in a similar way and sends it to A, who checks it.

Example 2: 9798-2-5

Figure 6.2 depicts the 9798-2-5 protocol, which is based on symmetric-key encryption and uses a Trusted Third Party. A first generates a time-variant parameter TVP_A (which must be non-repeating), and sends it with B's identity I_B and optionally a text field to the trusted party P. P then generates a fresh session key kab and computes $Token_{PA}$, which essentially consists of two encrypted copies of the key, using the long-term shared keys between P and A, and P and B, respectively. Upon receiving $Token_{PA}$, A decrypts the first part to retrieve the session key, and uses the second part to construct $Token_{AB}$. Finally, B retrieves the session key from this message and sends its authentication message $Token_{BA}$ to A.

6.1.4. Optional Fields and Variants

There are variants for each protocol listed in Table 6.1. Each protocol contains *text fields*, whose purpose is not specified, and which may be omitted, giving rise to another protocol variant. As can be seen in the previous examples, some of these text fields are plaintext, whereas others are within the scope of cryptographic operations (i.e., signed, encrypted, or cryptographically checked). Note that the standard does not provide a rationale for choosing among these options.

For some protocols that employ symmetric keys, the standard specifies that if unidirectional keys are used, some identity fields may be omitted from the encrypted (or checked) payload. This yields another variant.

The two protocols 9798-3-6 and 9798-3-7 both provide two options for the tokens included in their messages, giving rise to further variants. Note that in Section 6.4 we verify corrected versions of all 17 protocols in Table 6.1, taking all variants into account.

6.1.5. Threat Model and Security Properties

The ISO/IEC 9798 standard neither specifies a threat model nor defines the security properties that the protocols should satisfy. Instead, the introduction of ISO/IEC 9798-1 simply states that the protocols should satisfy mutual or unilateral authentication. Furthermore, the following attacks are mentioned as being relevant: man-in-the-middle attacks, replay attacks, reflection attacks, and forced-delay attacks. We note that the standard does not explicitly claim that any of the protocols are resilient against the above attacks.

6.2. Protocol Analysis

We analyze the protocols in the standard with respect to three standard authentication properties: *aliveness*, *non-injective agreement*, and *injective agreement*, as explained in Section 3.3.2. To perform our analysis, we use the Scyther tool [Cre08a] to find attacks on the ISO/IEC 9798 protocols. In Section 6.4, we will use our `scyther-proof` tool to generate machine-checked proofs of the corrected versions.

6.2.1. Analysis results

Using Scyther, we performed protocol analysis with respect to aliveness and agreement. Our analysis reveals that the majority of the protocols in the standard ensure agreement on the exchanged data items. However, we also found attacks on five protocols and two protocol variants. These attacks fall into the following categories: role-mixup attacks, type flaw attacks, multiple-role TTP attacks, and reflection attacks. In all cases, when an agent finishes his role of the protocol, the protocol has not been executed as expected. This can lead the agent to proceed with false assumptions about the state of the other involved agents.

In Table 6.2 we list the attacks we found using Scyther. The rows list the protocols, the properties violated, and any additional assumptions required for the attacks. We have omitted in the table all attacks that are entailed by the attacks listed. For example, since 9798-2-5 does not satisfy aliveness from B's perspective, it also does not satisfy any stronger properties such as (weak) agreement. We now describe the classes of attacks in more detail.

6.2.2. Role-Mixup Attacks

Some protocols are vulnerable to a *role-mixup attack* in which an agent's assumptions on another agent's role are wrong. The two agreement properties require that when Alice finishes her role apparently with Bob, then Alice and Bob not only agree on the exchanged data, but additionally Alice can be sure that Bob was performing in the intended role. Protocols vulnerable to role-mixup attacks therefore violate agreement.

Protocol	Role	Violated property	With	Data	Assumptions
9798-2-3	A	Non-injective agreement	B	TN_B, $Text_3$	
9798-2-3	B	Non-injective agreement	A	TN_A, $Text_1$	
9798-2-3 (UDK)	A	Non-injective agreement	B	TN_B, $Text_3$	
9798-2-3 (UDK)	B	Non-injective agreement	A	TN_A, $Text_1$	
9798-2-5	A	Aliveness	B		Alice-talks-to-Alice
9798-2-5	B	Aliveness	A		
9798-2-6	A	Aliveness	B		
9798-2-6	B	Aliveness	A		
9798-3-3	A	Non-injective agreement	B	TN_B, $Text_3$	
9798-3-3	B	Non-injective agreement	A	TN_A, $Text_1$	
9798-3-7-1	A	Non-injective agreement	B	R_A, R_B, $Text_8$	Type-flaw
9798-4-3	A	Non-injective agreement	B	TN_B, $Text_3$	
9798-4-3	B	Non-injective agreement	A	TN_A, $Text_1$	
9798-4-3 (UDK)	A	Non-injective agreement	B	TN_B, $Text_3$	
9798-4-3 (UDK)	B	Non-injective agreement	A	TN_A, $Text_1$	

Table 6.2.: Overview of attacks found. (UDK) indicates the protocol variants where unidirectional keys are used

Figure 6.3 shows an example of a role-mixup attack on the 9798-4-3 protocol from Figure 6.1. Agents perform actions such as sending and receiving messages, resulting in message transmissions represented by horizontal arrows. Actions are executed within threads, represented by vertical lines. The box at the top of each thread denotes the parameters involved in the thread's creation. Claims of security properties are denoted by hexagons and a crossed-out hexagon denotes that the claimed property is violated.

In this attack, the adversary uses a message from Alice in role A (thread 1) to trick Alice in role B (thread 3) into thinking that Bob is executing role A and is trying to initiate a session with her. However, Bob (thread 2) is only replying to a message provided to him by the adversary, and is executing role B. The adversary thereby tricks Alice into thinking that Bob is in a different state than he actually is.

Additionally, when the optional text fields $Text_1$ and $Text_3$ are used, the role-mixup attack also violates the agreement property with respect to these fields: Alice will end the protocol believing that the optional field data she receives from Bob was intended as $Text_1$, whereas Bob actually sent this data in the $Text_3$ field. Depending on the use of these fields, this can constitute a serious security problem. Note that exploiting these attacks, as well as the other attacks described below, does not require "breaking" cryptography. Rather, the adversary exploits similarities among messages and the willingness of agents to engage in the protocol.

Summarizing, we found role-mixup attacks on the following protocols: 9798-2-3 with bidirectional or unidirectional keys, 9798-2-5, 9798-3-3, and 9798-4-3 with bidirectional or unidirectional keys.

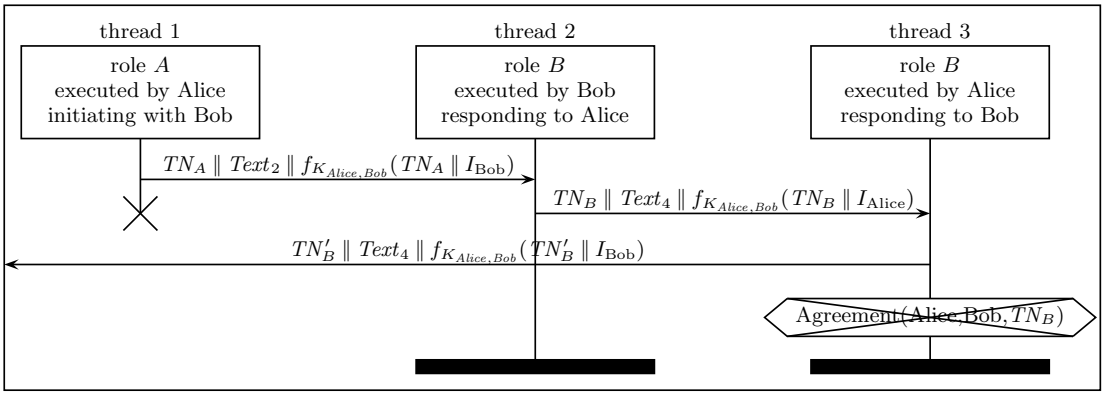


Figure 6.3.: Role-mixup attack on 9798-4-3: when Alice finishes thread 3 she wrongly assumes that Bob was performing the A role

6.2.3. Type Flaw Attacks

Some protocol implementations are vulnerable to *type flaw attacks* where data of one type is parsed as data of another type. Consider, for example, an implementation where agent names are encoded into bit-fields of length n, which is also the length of the bit-fields representing nonces. It may then happen that an agent who expects to receive a value that it has not seen before (such as a nonce generated in another role) accepts a bit string that was intended to represent an agent name.

Scyther finds such an attack on the 9798-3-7 protocol, also referred to as "Five pass authentication (initiated by B)" [Int10c, p. 4]. In the attack, both (agent) Alice and (trusted party) Terence mistakenly accept the bit string corresponding to the agent name "Alice" as a nonce.

6.2.4. Attacks Involving TTPs that Perform Multiple Roles

Another class of attacks occurs when parties can perform both the role of the trusted third party and another role. This scenario is not excluded by the standard.

In Figure 6.4 we show an attack on 9798-2-5, from Figure 6.2. The attack closely follows a regular protocol execution. In particular, threads 1 and 3 perform the protocol as expected. The problem is thread 2. Threads 1 and 3 assume that the participating agents are Alice (in the A role), Bob (in the B role), and Pete (in the P role). From their point of view, Alice should be executing thread 2. Instead, thread 2 is executed by Pete, under the assumption that Alice is performing the P role. Thread 2 receives only a single message in the attack, which is $Token_{PA}$. Because the long-term keys are bidirectional, thread 2 cannot determine from the part of the message encrypted with K_{AP} that thread 1 has different assumptions. Thread 2 just forwards the other encrypted message part blindly to thread 3, as it does not expect to be able to decrypt it. Finally, thread 3 cannot detect the confusion between Alice and Pete, because the information in $Token_{AB}$ that was added by thread 2 only includes Bob's name.

This attack violates aliveness because Bob was apparently talking to Alice, but Alice

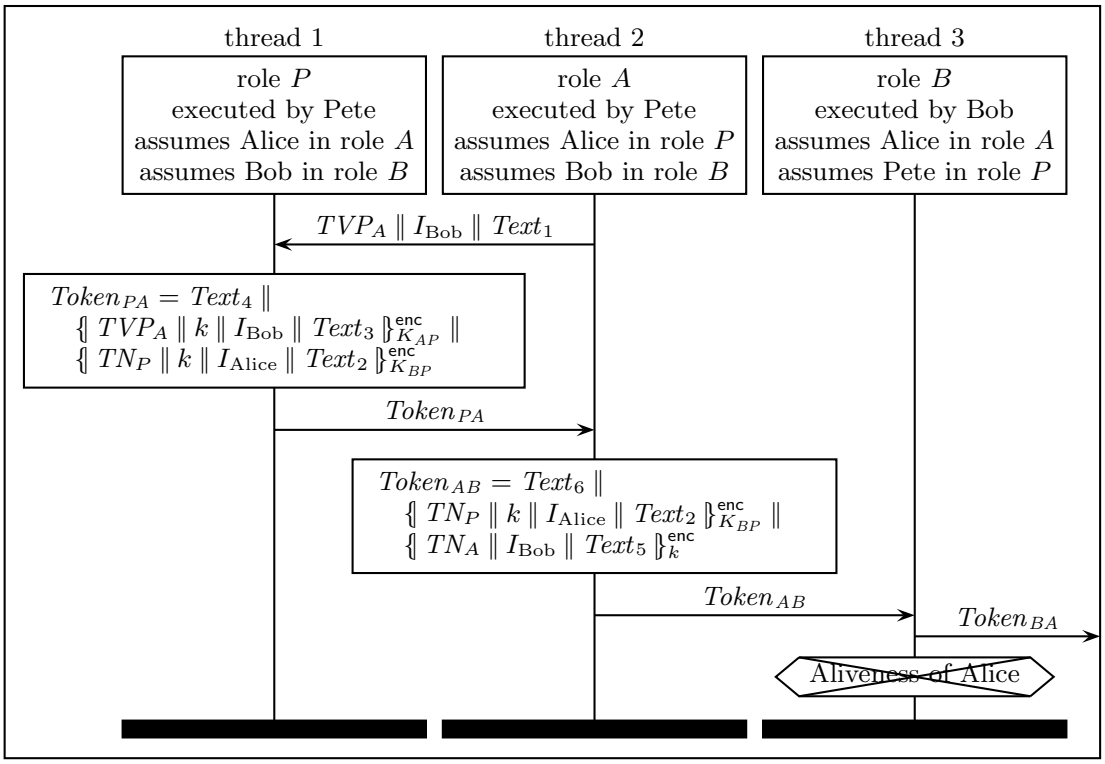


Figure 6.4.: Attack on the 9798-2-5 protocol where the trusted third party Pete performs both the P role and the A role. The assumptions of thread 1 and 3 agree. Bob wrongly concludes that Alice is alive.

never performed any action. In this case, the protocol fails to achieve even the weakest form of authentication.

Summarizing, we found attacks involving TTPs that perform multiple roles on the 9798-2-5 and 9798-2-6 protocol.

6.2.5. Reflection Attacks

Reflection attacks occur when agents may start sessions communicating with themselves, a so-called *Alice-talks-to-Alice* scenario. The feasibility and relevance of this scenario depends on the application and its internal checks. For example, it may be that Alice uses the same private key on different physical machines, and uses an Alice-talks-to-Alice scenario to communicate from one of her machines to another.

If an Alice-talks-to-Alice scenario is possible, some protocols are vulnerable to reflection attacks. The Message Sequence Chart in Figure 6.5 shows an example for the 9798-4-3 protocol from Figure 6.1. In this attack, the adversary (not depicted) reflects the time stamp (or nonce) and cryptographic check value from the message sent by Alice back to the same thread, while prepending the message Text4.

The attack violates both agreement properties, because even though the apparent partner (Alice) has performed an action, she did not perform the expected role. Further-

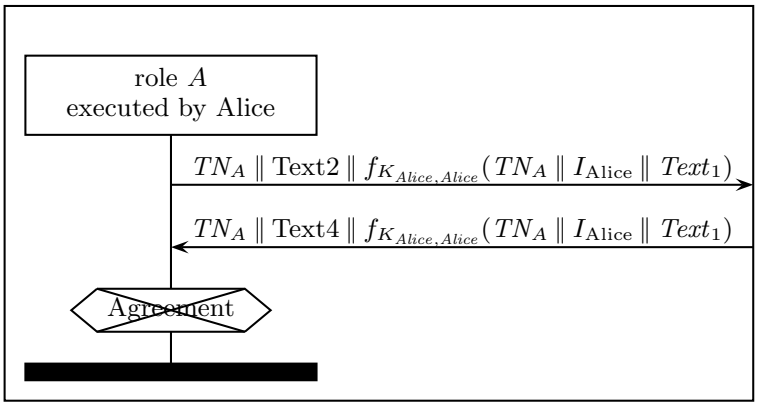


Figure 6.5.: Reflection attack on 9798-4-3

more, this attack violates one of the main requirements explicitly stated in the ISO/IEC 9798-1 introduction, namely absence of reflection attacks.

Summarizing, we found reflection attacks on the following protocols: 9798-2-3 with bidirectional or unidirectional keys, 9798-2-5, 9798-3-3, and 9798-4-3 with bidirectional or unidirectional keys.

6.3. Repairing the Protocols

6.3.1. Root Causes of the Problems

We identify two shortcomings in the design of the protocols, which together account for all of the weaknesses detected.

1) Cryptographic Message Elements May Be Accepted at Wrong Positions. In both the reflection and role mixup attacks, the messages that are received in a particular step of a role were not intended to be received at that position. By design, the protocol messages are all similar in structure, making it impossible to determine at which point in the protocols the messages were intended to be received.

As a concrete example, consider the reflection attack in Figure 6.5. Here, the message sent in the protocol's first step can be accepted in the second step, even though this is not part of the intended message flow.

2) Underspecification of the Involved Identities and their Roles. As noted, the symmetric-key based protocols with a TTP, 9798-2-5 and 9798-2-6, do not explicitly state that entities performing the TTP role cannot perform other roles. Hence it is consistent with the standard for Alice to perform both the role of the TTP as well as role A or B. In these cases, the aliveness of the partner cannot be guaranteed, as explained in Section 6.2.4. The source of this problem is that one cannot infer from each message which identity is associated to which role.

For example, consider the first encrypted component from the third message in the

9798-2-5 protocol with bidirectional keys, in Figure 6.2.

$$\{\!|\, TN_P \,\|\, kab \,\|\, I_A \,\|\, Text_2 \,|\!\}^{\mathsf{enc}}_{K_{BP}}$$

This message implicitly includes the identities of the three involved agents: the identity of the agent performing the A role is explicitly included in the encryption, and the shared long-term key K_{BP} implicitly associates the message to the key shared between the agent performing the B and P roles. However, because the key is bidirectional, the recipient cannot determine which of the two agents (say, Bob and Pete) sharing the key performed which role: either Bob performed the B role and Pete the P role, or vice versa. Our attack exploits this ambiguity.

6.3.2. Associated Design Principles

To remedy these problems, we propose two principles for designing security protocols. These principles are in the spirit of Abadi and Needham's eleven principles for prudent engineering practice for cryptographic protocols [AN96].

Our first principle concerns tagging.

> **Principle: positional tagging.** Cryptographic message components should contain information that uniquely identifies their origin. In particular, the information should identify the protocol, the protocol variant, the message number, and the particular position within the message, from which the component was sent.

This is similar in spirit to Abadi and Needham's Principle 1, which states that *"Every message should say what it means: the interpretation of the message should depend only on its content. It should be possible to write down a straightforward English sentence describing the content — though if there is a suitable formalism available that is good too."* Our principle does not depend on the meaning of the message as intended by the protocol's designer. Instead, it is based solely on the structure of the protocol messages and their acceptance conditions.

Note that we consider protocols with optional fields to consist of multiple protocol variants. Thus, a message component where fields are omitted should contain information to uniquely determine which fields were omitted.

Our second principle is a stricter version of Abadi and Needham's Principle 3.

> **Principle: inclusion of identities and their roles.** Each cryptographic message component should include information about the identities of all the agents involved in the protocol run and their roles, unless there is a compelling reason to do otherwise.

A compelling reason to leave out identity information might be that *identity hiding* is a requirement, i.e., Alice wants to hide that she is communicating with Bob. However, such requirements can usually be met by suitably encrypting identity information.

Contrast this principle with the Abadi and Needham's Principle 3: *"If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message."* The original principle is only invoked when

Amendment 1:

The cryptographic data (encryptions, signatures, cryptographic check values) used at different places in the protocols must not be interchangeable. This may be enforced by including in each encryption/signature/CCF value the following two elements:

1. The object identifier as specified in Annex B [Int10c, p. 6], in particular identifying the ISO standard, the part number, and the authentication mechanism.
2. For protocols that contain more than one cryptographic data element, each such element must contain a constant that uniquely identifies the position of the element within the protocol.

The recipient of a message must verify that the object identifier and the position identifiers are as expected. The cryptographic keys used by implementations of the ISO/IEC 9798 protocols must be distinct from the keys used by other protocols.

Amendment 2:

When optional fields, such as optional identities or optional text fields, are not used then they must be set to empty. In particular, the message encoding must ensure that the concatenation of a field and an empty optional field is uniquely parsed as a concatenation. This can be achieved by implementing optional fields as variable-length fields. If the optional field is not used, the length of the field is set to zero.

Amendment 3:

Entities that perform the role of the TTP in the 9798-2-5 and 9798-2-6 protocols must not perform the A or B role.

Figure 6.6.: Proposed amendments to the ISO/IEC 9798 standard

the identity is essential. Instead, we propose to always include information on all the identities as well as their roles. This principle would have prevented attacks on many protocols, including the attacks on the 9798-2-5 and 9798-2-6 protocols, as well as the Needham-Schroeder protocol [Low96].

For protocols with a fixed number of roles, this principle can be implemented by including an ordered sequence of the identities involved in each cryptographic message component, such that the role of an agent can be inferred from its position in the sequence.

6.3.3. Proposed Modifications to the Standard

All the previously mentioned attacks on the ISO/IEC 9798 can be prevented by applying the previous two principles. Specifically, we propose three modifications to the ISO standard, shown in Figure 6.6. The first two directly follow from the principles and the third modification restricts the use of two protocols in the standard. Afterwards we give an example of a repaired protocol.

Note that in this section we only give informal arguments why our modifications prevent the attacks. In Section 6.4, we provide machine-checked proofs that this is the case.

Ensuring that Cryptographic Data Cannot Be Accepted at the Wrong Point

We factor the first principle (positional tagging) into two parts and propose two corresponding amendments to the standard. First, we explicitly include in each cryptographic message component constants that uniquely identify the protocol, the message number, and the position within the message. Second, we ensure that protocol variants can be uniquely determined from the messages.

In our first amendment, shown in Figure 6.6, we implement unique protocol identifiers by using an existing part of the standard: the object identifier from Annex B of the standard, which specifies an encoding of a unique protocol identifier. We also introduce a unique identifier for the position of the cryptographic component within the protocol.

Amendment 1 prevents all reflection attacks because messages sent in one step will no longer be accepted in another step. Furthermore, it prevents all role mixup attacks, because the unique constants in the messages uniquely determine the sending role. The final part of Amendment 1, stating that cryptographic keys should not be used by other protocols, provides distinctness of cryptographic messages with respect to any other protocols.

Our second amendment, also shown in Figure 6.6, ensures that the protocol variant (determined by the omission of optional fields) can be uniquely determined from the messages. We implement this by requiring that the recipient of a message can uniquely determine which optional fields, if any, were omitted.

To see why protocols with omitted optional fields must be considered as protocol variants, consider the following example: Consider a protocol in which a message contains the sequence $X \parallel I_A \parallel \mathit{Text}$, where I_A is an identity field that may be dropped (e.g., with unidirectional keys) and *Text* is an optional text field. Then, it may be the case that in one protocol variant, an agent expects a message of the form $X \parallel I_A$, whereas the other implementation expects a message of the form $X \parallel \mathit{Text}$. The interaction between the two interpretations can result in attacks. For example, the text field is used to insert a fake agent identity, or an agent identity is wrongly assumed to be the content of the text field.

If we follow the second amendment in the above example, the expected messages correspond to $X \parallel I_A \parallel \perp$ and $X \parallel \perp \parallel \mathit{Text}$, respectively, where $\perp$ denotes the zero-length field. Because the ISO/IEC 9798 standard requires that concatenated fields can be uniquely decomposed into their constituent parts, misinterpretation of the fields is no longer possible.

Together, Amendments 1 and 2 implement our first principle.

Addressing Underspecification of the Role Played by Agents

Almost all the protocols in the ISO/IEC 9798 standard already adhere to our second principle: unique identification of the involved parties and their roles. However, all protocols in the standard conform to Abadi and Needham's third principle because the messages uniquely determine the identities of all involved parties.

There are two protocols in the standard that conform to Abadi and Needham's principle but not to our second principle: 9798-2-5 and 9798-2-6. For example, the messages of the 9798-2-5 protocol identify all parties involved by their association to

1. $A \to B:\quad TN_A \,\|\, Text_2 \,\|\, f_{K_{AB}}(\textbf{“9798-4-3 ccf1”} \,\|\, TN_A \,\|\, I_B \,\|\, \perp)$
2. $B \to A:\quad TN_B \,\|\, Text_4 \,\|\, f_{K_{AB}}(\textbf{“9798-4-3 ccf2”} \,\|\, TN_B \,\|\, I_A \,\|\, Text_3)$

Figure 6.7.: Repaired version of the 9798-4-3 protocol with omitted $Text_1$ field

the long-term keys. However they do not conform to our second principle because the roles of the involved identities cannot be uniquely determined from the messages. This is the underlying reason why, as currently specified, the 9798-2-5 and 9798-2-6 protocols do not guarantee the aliveness of the partner, as shown in Section 6.2.4.

This problem can be solved by applying our principle, i.e., including the identities of all three participants in each message, so that their roles can be uniquely determined. This is an acceptable solution and we have formally verified it using the method of Section 6.4. However, from our analysis with Scyther, we observe that the attacks require that the Trusted Third Party also performs other roles (A or B). Under the assumption that in actual applications a TTP will, by definition, not perform the A or B role, the protocols already provide strong authentication. Thus, an alternative solution is to leave the protocols unchanged and make this restriction explicit. This results in more streamlined protocols and also requires minimal changes to the standard. This is the proposal made in Amendment 3 in Figure 6.6. We have also verified this solution as described in Section 6.4.

Repaired Protocols

Applying our principles and proposed amendments to the standard, we obtain repaired versions of the protocols. As an example, we show the repaired version of the 9798-4-3 protocol with bidirectional keys in Figure 6.7. In this example, the $Text_1$ field is not used, and is therefore replaced by $\perp$. Each use of a cryptographic primitive (in this case the cryptographic check function) includes a constant that uniquely identifies the protocol (`9798-4-3`) as well as the position within the protocol specification (`ccf1` and `ccf2`).

6.4. Proving the Correctness of the Repaired Protocols

The principles and amendments proposed in the previous section are motivated by our analysis of the attacks and the protocol features that enable them. Consequently, the principles and amendments are designed to eliminate these undesired behaviors. Such principles are useful guides for protocol designers but their application does not strictly provide any security guarantees. In order to ensure that the repaired protocols actually have the intended strong authentication properties, we provide machine-checked correctness proofs.

6.4.1. Generating Machine-Checked Correctness Proofs

We use our `scyther-proof` tool to generate proofs of the authentication properties that are machine-checked by Isabelle/HOL. As explained in Section 5.2, `scyther-proof`

Repaired protocol	Role	Property	With	Data
9798-2-1	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-2-1 (UDK)	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-2-2	B	Injective agreement	A	$A, B, R_B, Text_2$
9798-2-2 (UDK)	B	Injective agreement	A	$A, B, R_B, Text_2$
9798-2-3	A	Non-injective agreement	B	$A, B, TN_B, Text_3$
	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-2-3 (UDK)	A	Non-injective agreement	B	$A, B, TN_B, Text_3$
	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-2-4	A	Injective agreement	B	$A, B, R_A, R_B, Text_2, Text_4$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$
9798-2-4 (UDK)	A	Injective agreement	B	$A, B, R_A, R_B, Text_2, Text_4$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$
9798-2-5 (DR)	A	Injective agreement	B	$A, B, P, K_{AB}, TN_A, Text_5, TN_B, Text_7$
	B	Non-injective agreement	A	$A, B, P, K_{AB}, TN_A, Text_5$
	A	Injective agreement	P	$A, B, P, K_{AB}, TVP_A, Text_3$
	B	Non-injective agreement	P	$A, B, P, K_{AB}, TN_P, Text_2$
9798-2-5 (UDK)	A	Injective agreement	B	$A, B, P, K_{AB}, TN_A, Text_5, TN_B, Text_7$
	B	Non-injective agreement	A	$A, B, P, K_{AB}, TN_A, Text_5$
	A	Injective agreement	P	$A, B, P, K_{AB}, TVP_A, Text_3$
	B	Non-injective agreement	P	$A, B, P, K_{AB}, TN_P, Text_2$
9798-2-6 (DR)	A	Injective agreement	B	$A, B, P, K_{AB}, R'_A, R_B, Text_6, Text_8$
	B	Injective agreement	A	$A, B, P, K_{AB}, R'_A, R_B, Text_6$
	A	Injective agreement	P	$A, B, P, R_A, K_{AB}, Text_4$
	B	Injective agreement	P	$A, B, P, R_B, K_{AB}, Text_3$
9798-2-6 (UDK)	A	Injective agreement	B	$A, B, P, K_{AB}, R'_A, R_B, Text_6, Text_8$
	B	Injective agreement	A	$A, B, P, K_{AB}, R'_A, R_B, Text_6$
	A	Injective agreement	P	$A, B, P, R_A, K_{AB}, Text_4$
	B	Injective agreement	P	$A, B, P, R_B, K_{AB}, Text_3$

Table 6.3.: Properties proven of the repaired protocols in Part 2 of the standard. (UDK) indicates the protocol variants where unidirectional keys are used. (DR) indicates the variants where participants that perform the A or B role cannot also perform the P role, conform Amendment 3 in Figure 6.6.

searches for proofs with the fewest number of applications of the Chain rule, i.e., case distinctions on the possible sources of messages that are known to the intruder. Therefore, the generated proof scripts are amenable to human inspection and understanding. For example, in the proofs of our repaired protocols, two such case distinctions are required on average to prove a security property. To simplify the task of understanding how the proofs work and, hence, why the protocol is correct, the tool also generates proof outlines. These consist of a representation of the security property proven and a tree of case distinctions constituting the proof.

6.4.2. Parallel Composition

We verify the parallel composition of all repaired protocols of the ISO/IEC 9798 standard as follows. Given the disjoint encryption theorem [GT00], which implies that protocols that do not share keying material can be safely composed, it is sufficient to verify only

Repaired protocol	Role	Property	With	Data
9798-3-1	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-3-2	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$
9798-3-3	A	Non-injective agreement	B	$A, B, TN_B, Text_3$
	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-3-4	A	Injective agreement	B	$A, B, R_A, R_B, Text_2, Text_4$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$
9798-3-5	A	Injective agreement	B	$A, B, R_A, R_B, Text_5$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_3, Text_5$
9798-3-6 (Opt. 1)	A	Injective agreement	B	$A, B, R_A, R_B, Text_2$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_8$
	A	Injective agreement	T	$B, T, R'_A, PK_B, Text_6$
	B	Injective agreement	T	$A, T, R_B, PK_A, Text_5$
9798-3-6 (Opt. 2)	A	Injective agreement	B	$A, B, R_A, R_B, Text_2$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_8$
	A	Injective agreement	T	$A, B, T, R'_A, R_B, PK_A, PK_B, Text_5$
	B	Injective agreement	T	$A, B, T, R'_A, R_B, PK_A, PK_B, Text_5$
9798-3-7 (Opt. 1)	A	Injective agreement	B	$A, B, R_A, R_B, Text_8$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_6$
	A	Injective agreement	T	$B, T, R'_A, PK_B, Text_4$
	B	Injective agreement	T	$A, T, R_B, PK_A, Text_3$
9798-3-7 (Opt. 2)	A	Injective agreement	B	$A, B, R_A, R_B, Text_8$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_6$
	A	Injective agreement	T	$A, B, T, R'_A, R_B, PK_A, PK_B, Text_3$
	B	Injective agreement	T	$A, B, T, R'_A, R_B, PK_A, PK_B, Text_3$

Table 6.4.: Properties proven of the repaired protocols in Part 3 of the standard

Repaired protocol	Role	Property	With	Data
9798-4-1	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-4-1 (UDK)	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-4-2	B	Injective agreement	A	$A, B, R_B, Text_2$
9798-4-2 (UDK)	B	Injective agreement	A	$A, B, R_B, Text_2$
9798-4-3	A	Non-injective agreement	B	$A, B, TN_B, Text_3$
	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-4-3 (UDK)	A	Non-injective agreement	B	$A, B, TN_B, Text_3$
	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-4-4	A	Injective agreement	B	$A, B, R_A, R_B, Text_2, Text_4$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$
9798-4-4 (UDK)	A	Injective agreement	B	$A, B, R_A, R_B, Text_2, Text_4$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$

Table 6.5.: Properties proven of the repaired protocols in Part 4 of the standard. (UDK) indicates the protocol variants where unidirectional keys are used.

Repaired version of 9798-4-3

```
protocol isoiec_9798_4_3_bdkey_repaired
{
  leak_A. A ->  : TNa
  leak_B. B ->  : TNb

  text_1.   -> A: Text1, Text2
       1. A -> B: A, B, TNa, Text2, Text1, h(('CCF', k[A,B]), ('isoiec_9798_4_3_ccf_1',
                  TNa, B, Text1))
  text_2.   -> B: Text3, Text4
       2. B -> A: B, A, TNb, Text4, Text3, h(('CCF', k[A,B]), ('isoiec_9798_4_3_ccf_2',
                  TNb, A, Text3))
}
properties (of isoiec_9798_4_3_bdkey_repaired)
  A_non_injective_agreement: niagree(A_2[A,B,TNb,Text3] -> B_2[A,B,TNb,Text3], {A, B})
  B_non_injective_agreement: niagree(B_1[A,B,TNa,Text1] -> A_1[A,B,TNa,Text1], {A, B})
```

Figure 6.8.: Example input provided to the `scyther-proof` tool

the parallel composition of protocols that use the same cryptographic primitive and the same keys. We verify the properties of each protocol when composed in parallel with all other protocols that use the same cryptographic primitives and the same keys. Note that in the corresponding proofs, the case distinctions on the source of messages known to the intruder range over the roles of each protocol in the protocol group. Despite the substantial increase in the scope of these case distinctions, the proof structure of the composed protocols is the same as for the individual protocols, as all additional cases are always trivially discharged due to tagging: cryptographic components received by a thread of one protocol contain tags that do not match with the tags in messages produced by roles from other protocols.

6.4.3. Details of the Proven Properties

In Tables 6.3, 6.4, and 6.5 we provide details of the authentication properties proven using `scyther-proof` for each repaired protocol and its variants. For example, for the 9798-2-1 protocol with bidirectional keys, we have that if Bob successfully completes a thread of role B, apparently with Alice, then there must be a thread of Alice performing role A which agrees on the agent names, TN_A, and $Text_1$.

Non-injective agreement

For each repaired protocol, we use `scyther-proof` to prove that it satisfies at least non-injective agreement on all data items within the scope of cryptographic operators in the presence of a Dolev-Yao intruder. Moreover, we prove that this holds even when all the protocols from the standard are executed in parallel using the same key infrastructure, provided that the set of bidirectional keys is disjoint from the set of unidirectional keys. As the content of text fields is underspecified in the standard, we assume that the intruder chooses their content immediately before they are sent. We model timestamps and sequence numbers by random numbers that are public and chosen at the beginning of the execution of a role.

Example 16. Figure 6.8 specifies our model of the repaired 9798-4-3 protocol with bidirectional keys in the input language of the `scyther-proof` tool. The `leak_A` and

```
1  property (of isoiec_9798_4_3_bdkey_repaired)
2    A_non_injective_agreement:
3    "All #i.
4      role(#i) = isoiec_9798_4_3_bdkey_repaired_A &
5      step(#i, isoiec_9798_4_3_bdkey_repaired_A_2) &
6      uncompromised( A#i, B#i )
7      ==> (Ex #j. role(#j) = isoiec_9798_4_3_bdkey_repaired_B &
8                  step(#j, isoiec_9798_4_3_bdkey_repaired_B_2) &
9                  (A#j, B#j, TNb#j, Text3#j) = (A#i, B#i, TNb#i, Text3#i)) "
10 sources( h(('CCF', k[A#i,B#i]), ('isoiec_9798_4_3_ccf_2', TNb#i, A#i, Text3#i)) )
11   case fake
12   contradicts secrecy of k[A#i,B#i]
13 next
14   case (isoiec_9798_4_3_bdkey_B_2_repaired_hash #k)
15   tautology
16 qed
```

Figure 6.9.: Example proof outline generated by the `scyther-proof` tool

the `leak_B` steps model that the timestamps (represented here as randomly generated numbers) are publicly known by leaking them to the intruder. We model that the contents of the `Text_1` through `Text_4` fields are chosen by the intruder by defining them as variables that receive their content from the network, and therefore from the intruder. We model the cryptographic check function by the hash function h.

Figure 6.9 shows the proof outline for non-injective agreement for the `A`-role of this protocol, which is automatically generated by our tool. We have taken minor liberties here in its presentation to improve readability. In the figure, `#i` is a symbolic variable denoting some thread i and `A#i` is the value of the `A`-variable in the thread i. Lines 3-9 state the security property: for each thread `#i` that executes the `A`-role and has executed its Step `2` with uncompromised (honest) agents `A#i` and `B#i`, there exists some thread `#j` that executed Step `2` of the `B`-role and thread `#j` agrees with thread `#i` on the values of `A`, `B`, `TNb`, and `Text3`.

The proof proceeds by observing that thread `#i` executed Step `2` of the `A`-role. Therefore, thread `#i` received the hash in line 10 from the network, and therefore the intruder knows this hash. For our protocol, a case distinction on the sources of this hash results in two cases: (1) the intruder could have constructed (faked) this hash by himself or (2) the intruder could have learned this hash from some thread `#k` that sent it in Step `2` of the `B`-role. There are no other cases because all other hashes have different tags. Case 1 is contradictory because the intruder does not know the long-term key shared between the two uncompromised agents `A#i` and `B#i`. In Case 2, the security property holds because we can instantiate thread `#j` in the conclusion with thread `#k`. Thread `#k` executed Step `2` of the `B`-role and agrees with thread `#i` on all desired values because they are included in the hash. ♠

Injective agreement

The protocols in the ISO/IEC 9798 standard are also designed to provide *injective* agreement. In this case, if an agent completes the A or B role n times, he can be sure that the agent performing the other roles has participated at least n times. In the terminology of the ISO/IEC standard, this concept is called *uniqueness*; it is used to rule

out replay attacks. The standard uses three techniques to achieve uniqueness: random numbers, time stamps, and sequence numbers.

Random numbers About half of the protocols achieve uniqueness by using random numbers for challenge-response. The party that wants to ensure uniqueness generates a fresh random value and sends this to the other party. The other party must include the random value in his response, thereby uniquely binding the response to the request.

For each protocol in the standard that uses random numbers, we prove injective agreement for its repaired version.

Time stamps and sequence numbers The other protocols use time stamps or sequence numbers to achieve uniqueness. This includes all one-pass protocols because including a challenge-response structure requires at least two messages.

Each time a party runs a new instance of a role, it uses a new sequence number which differs from all numbers it previously used. Recipients of messages that include a sequence number are required to verify that the received number is different from all previously received sequence numbers. Similarly, time stamps are used to achieve uniqueness by only accepting them within a fixed time window, and verifying that the time stamp has not already been received in the time window. For both mechanisms, the required checks can only be performed by a thread if it can access data of other threads of the same agent. In other words, checking uniqueness of sequence numbers or time stamps requires shared memory among the different threads of an agent.

In our protocol model, we can not precisely model the required uniqueness checks of sequence numbers and timestamps. The reason for this is that our protocol model assumes that an agent's threads do not share any state other than the agent's long-term keys. Hence we can not prove injective agreement for protocols that use time stamps or sequence numbers within our model. However, it is straightforward to see that if an agent verifies that the messages received in each thread are different from those received in previous threads (which is possible because the messages include either a time stamp or a sequence number), injective authentication follows from non-injective authentication.

From our analysis we conclude that all repaired protocols in the standard that provably satisfy non-injective agreement, also satisfy injective agreement.

6.4.4. Performance

All protocol models (including the property specifications) are distributed together with the source-code of the `scyther-proof` tool [Mei12]. Using an Intel i7 Quad Core laptop with 4GB of RAM, the full proof script generation requires less than 15 seconds, and Isabelle's proof checking requires less than 1.5 hours.

Part II.

Extending the Scope

7. Security Protocol Model

We model the execution of a security protocol in the presence of an adversary as a labeled transition system, whose state consists of the adversary's knowledge, the messages on the network, information about freshly generated values, and the protocol's state. We formalize this transition system as a set of (labeled) multiset rewriting rules, which model both the adversary actions and the protocol steps. Security properties are modeled as trace properties of this transition system.

In the following, we first describe how we model cryptographic messages using equational theories and introduce the notion of labeled multiset rewriting. Then, we describe how we model the execution of a security protocol using labeled multiset rewriting. Afterwards, we define our property specification language and illustrate our protocol model on two examples.

7.1. Cryptographic Messages

To model cryptographic messages, we use an order-sorted term algebra with the sort *msg* and two incomparable subsorts *fresh* and *pub* for fresh and public names. We use fresh names to model freshly generated data (nonces, coin-flips, etc..) and public names to model publicly known constants (agent names, message tags, etc.). We assume that there are two countably infinite sets FN and PN of *fresh* and *public names*.

For the specification of security protocols, our approach allows choosing an arbitrary signature Σ and an equational theory E that formalizes the semantics of the function symbols in Σ. For the verification of security protocols, further restrictions on the equational theory E are often necessary. These restrictions depend on the verification theory used for constructing the security proofs. We state the restrictions exploited by the verification theory developed in this thesis, when explaining it in Chapter 8.

Cryptographic *messages* are modeled by the ground terms in $\mathcal{T}_\Sigma(PN \cup FN \cup \mathcal{V})$, which we abbreviate as $\mathcal{M}_\Sigma$. The following example illustrates the use of term algebras and equational theories to model cryptographic operators.

Example 17 (Pairing, Hashing, and Symmetric Encryption)**.** We use $\mathcal{M}_{\Sigma_{PHS}}$ with

$$\Sigma_{PHS} := \{\langle \_, \_ \rangle, \mathrm{h}(\_), \mathrm{enc}(\_, \_), \mathrm{dec}(\_, \_)\ \mathrm{fst}(\_), \mathrm{snd}(\_)\}$$

to model cryptographic messages built using pairing, hashing, and symmetric encryption. Intuitively, the function symbols in Σ_{PHS} denote calls to the algorithms implementing pairing, hashing, symmetric encryption and decryption, and accessing the first and second component of a pair. Let $n \in FN$ model a nonce and let $s \in FN$ model a secret. An example of a cryptographic message in $\mathcal{M}_{\Sigma_{PHS}}$ is $\mathrm{enc}(n, \mathrm{h}(s))$, the encryption of the nonce n with the hash of the secret s. Another example of a message in $\mathcal{M}_{\Sigma_{PHS}}$

is $\mathrm{fst}(\langle n, s\rangle)$, the first component of the pair built from the fresh names n and s. We expect that $\mathrm{fst}(\langle n, s\rangle)$ is equal to n.

We formalize the interaction between the functions in Σ_{PHS} using the equational theory E_{PHS} generated by the equations

$$\mathrm{dec}(\mathrm{enc}(m, k), k) \simeq m\ , \qquad \mathrm{fst}(\langle x, y\rangle) \simeq x\ , \qquad \mathrm{snd}(\langle x, y\rangle) \simeq y\ .$$

Examples of equalities and inequalities modulo E_{PHS} are $n =_{E_{PHS}} \mathrm{fst}(\langle n, s\rangle) =_{E_{PHS}} \mathrm{dec}(\mathrm{enc}(n, \mathrm{h}(s)), \mathrm{h}(s))$, $n \neq_{E_{PHS}} \mathrm{h}(s)$, and $n \neq_{E_{PHS}} \mathrm{enc}(n, \mathrm{h}(s))$. ♠

We will use Σ_{PHS} and E_{PHS} for most of our examples in this thesis. We stress however that our approach is not limited to this simple equational theory. As we show in [SMCB12a], our approach supports the combination of an equational theory modeling Diffie-Hellman exponentiation with an arbitrary subterm-convergent rewriting theory modeling the properties of a set of user-defined cryptographic operators. Note that subterm-convergent rewriting theories suffice to model asymmetric encryption, signatures, and similar operators. Schmidt moreover shows in his thesis [Sch12] that our approach can also be extended with support for an equational theory modeling multisets and bilinear pairings.

To simplify the presentation of the rest of this thesis, we write $\mathcal{T}$ for $\mathcal{T}_\Sigma(FN \cup PN \cup \mathcal{V})$ and $\mathcal{M}$ for $\mathcal{T}_\Sigma(FN \cup PN)$ if Σ is irrelevant or clear from the context.

7.2. Labeled Multiset Rewriting

Multiset rewriting is commonly used to model concurrent systems because it naturally supports independent transitions. If two rewriting steps rewrite the state at positions that do not overlap, then they can be applied in parallel. Multiset rewriting has been used in the context of security protocol analysis by Cervesato *et al.* [CDL+02] and in Maude [EMM07]. It is also possible to give a translation from applied π processes, as for example used in ProVerif [Bla09], into our dialect of multiset rewriting.

The intuition for modeling a concurrent system using multiset rewriting is the following. The system's state is modeled by a multiset of facts and the system itself is modeled by a set of multiset rewriting rules. The execution of the system starts with the empty multiset of facts. In each execution step, some instance of one of the system's multiset rewriting rules is chosen and used to replace one multiset of facts in the state by another multiset of facts. The semantics of the different facts in the state is therefore given by the system's multiset rewriting rules.

In this thesis, we use three extensions to multiset rewriting that simplify the specification and verification of security protocols. First, we partition the set of facts into linear and persistent facts. Linear facts model resources that can only be consumed once, whereas persistent facts model inexhaustible resources that can be consumed arbitrarily often. We use persistent facts for example to model the adversary knowledge. Second, we label the multiset rewriting rules with actions and use (action-)traces to denote executions of multiset rewriting systems. This allows us to specify security properties as trace properties of security protocols modeled as multiset rewriting systems. Third, we build-in fresh name generation, i.e., we allow multiset rewriting rules to generate

fresh names that are guaranteed to be different from all other fresh names generated previously.

Formally, we assume an unsorted signature Σ_{Fact} partitioned into *linear* and *persistent* fact symbols. We moreover assume that Σ_{Fact} contains the linear, unary fact symbol Fr, which we use to mark freshly generated fresh names. We define the set of *facts* as

$$\mathcal{F} := \{F(t_1, .., t_k) \mid t_i \in \mathcal{T},\ F \in \Sigma^k_{Fact}\}$$

where Σ^k_{Fact} denotes all function symbols of arity k in Σ_{Fact}. We denote the set of *ground facts* by $\mathcal{G}$. We say that a fact $F(t_1, .., t_k)$ is *linear* if F is linear and *persistent* if F is persistent.

A *(labeled) multiset rewriting rule* is a triple (l, a, r) with $l, a, r \in \mathcal{F}^*$, denoted by $l -[\ a\ ]\!\!\rightarrow r$. For a multiset rewriting rule $ri := l -[\ a\ ]\!\!\rightarrow r$, we define its *premises* as $prems(ri) := l$, its *actions* as $acts(ri) := a$, and its *conclusions* as $concs(ri) := r$. We often write multiset rewriting rules as inference rules with multiple premises and conclusions. That is, we often write a rule $[l_1, \ldots, l_k] -[\ a_1, \ldots, a_n\ ]\!\!\rightarrow [r_1, \ldots, r_m]$ as

$$\frac{l_1 \quad \ldots \quad l_k}{r_1 \quad \ldots \quad r_m}[a_1, \ldots, a_n] \quad \text{or} \quad \frac{l_1 \quad \ldots \quad l_k}{r_1 \quad \ldots \quad r_m} \text{ if the rule has no associated actions.}$$

For a set of multiset rewriting rules R, we use $insts(R)$ to denote the *set of instances* of R and $ginsts(R)$ to denote the *set of ground instances* of R.

A *(labeled) multiset rewriting system* R is a finite set of labeled multiset rewriting rules such that

MSR1 no rule $l -[\ a\ ]\!\!\rightarrow r \in R$ contains a fresh name,

MSR2 no conclusion of a rule $l -[\ a\ ]\!\!\rightarrow r \in R$ contains a Fr fact, and

MSR3 for each rule instance $l -[\ a\ ]\!\!\rightarrow r \in_E ginsts(R)$, $\left(\bigcap_{r' =_E r} St(r') \cap FN\right) \subseteq St(l) \cap FN$.

As we will see below, these three conditions ensure that all fresh names are created with the multiset rewriting rule $\textsc{Fresh} := ([] -[]\!\!\rightarrow \mathsf{Fr}(x{:}fresh))$, which formalizes our built-in support for fresh name generation. Intuitively, Condition **MSR3** formalizes that no conclusion of a rule instance can contain fresh names that do not occur in one of the instance's premises. To see the insufficiency of the simpler syntactic condition, $vars(r) \subseteq vars(l) \cup \mathcal{V}_{pub}$ for each rule $l -[\ a\ ]\!\!\rightarrow r \in R$, consider the rule $\mathrm{fst}(y) -[]\!\!\rightarrow \mathrm{snd}(y)$ and the equational theory E_{PHS}.

The labeled transition relation $\Longrightarrow_{R,E} \subseteq \mathcal{G}^\sharp \times \mathcal{P}(\mathcal{G}) \times \mathcal{G}^\sharp$ for a multiset rewriting system R and an equational theory E is defined by the transition rule

$$\frac{l -[\ a\ ]\!\!\rightarrow r \in_E ginsts(R \cup \{\textsc{Fresh}\}) \quad lfacts(l) \subseteq^\sharp S \quad pfacts(l) \subseteq set(S)}{S \overset{set(a)}{\Longrightarrow}_{R,E} ((S \setminus^\sharp lfacts(l)) \cup^\sharp mset(r))},$$

where $lfacts(l)$ is the multiset of all linear facts in l and $pfacts(l)$ is the set of all persistent facts in l. This transition rule models rewriting the state with a ground instance of a multiset rewriting rule in R or the $\textsc{Fresh}$ rule. Since we perform multiset rewriting modulo E, we use $\in_E$ for the rule instance. As linear facts are consumed upon rewriting, we use multiset inclusion to check that all facts in $lfacts(l)$ occur sufficiently many times

in S. For persistent facts, we only check that every fact in $pfacts(l)$ occurs in S. To obtain the successor state, we remove the consumed linear facts and add the generated facts. The label associated to the transition is the set of actions of the rule instance.

A *trace* is a sequence of sets of ground facts denoting the sequence of actions that happened during the execution of a multiset rewriting system. We model the execution of a multiset rewriting system R modulo an equational theory E by the set of R, E-*traces* defined as

$$\begin{aligned} traces_E(R) := \{[A_1, \ldots, A_n] \mid \exists S_1, \ldots, S_n \in \mathcal{G}^\sharp.\ \varnothing^\sharp \overset{A_1}{\Longrightarrow}_{R,E} S_1 \overset{A_2}{\Longrightarrow}_{R,E} \ldots \overset{A_n}{\Longrightarrow}_{R,E} S_n \\ \wedge\ \forall i \neq j.\ \forall x.\ (S_{i+1} \setminus^\sharp S_i) = \{\mathsf{Fr}(x)\}^\sharp \Rightarrow \\ (S_{j+1} \setminus^\sharp S_j) \neq \{\mathsf{Fr}(x)\}^\sharp \}\ . \end{aligned}$$

The second conjunct ensures that each instance of the Fresh rule is used at most once in a trace. Each consumer of a $\mathsf{Fr}(n)$ fact therefore obtains a different fresh name, as the Fresh rule is the only rule that produces Fr facts and we consider only executions with unique instances of this rule.

7.3. Protocol Specification and Execution

In this section, we explain how we use labeled multiset rewriting to model security protocols that communicate over a public channel controlled by a Dolev-Yao style adversary. Such an adversary can see and block every message sent on the channel he controls. He can furthermore send any message deducible from the messages that he has seen on this channel.

We only model a single Dolev-Yao style adversary controlling a single public channel. This simplifies the presentation of the corresponding verification theory in Section 8.4. Extending our model and verification theory with support for multiple adversaries and public channels is straightforward.

When modeling protocols, we assume that Σ_{Fact} includes an arbitrary number of protocol-specific fact symbols to describe the protocol state and the special fact symbols K, Out, and In, which we use to model a public channel controlled by a Dolev-Yao style adversary. A persistent fact $\mathsf{K}(m)$ denotes that m is known to the adversary. A linear fact $\mathsf{Out}(m)$ denotes that the protocol has sent the message m, which can be received by the adversary. A linear fact $\mathsf{In}(m)$ denotes that the adversary has sent the message m, which can be received by the protocol. The semantics of these three fact symbols is given by the following set of *message deduction rules*.

$$\begin{aligned} MD_\Sigma := \left\{ \frac{\mathsf{Out}(x)}{\mathsf{K}(x)},\ \frac{\mathsf{K}(x)}{\mathsf{In}(x)}[\mathsf{K}(x)],\ \frac{}{\mathsf{K}(x{:}pub)},\ \frac{\mathsf{Fr}(x{:}fresh)}{\mathsf{K}(x{:}fresh)} \right\} \\ \cup \left\{ \frac{\mathsf{K}(x_1) \quad \ldots \quad \mathsf{K}(x_k)}{\mathsf{K}(f(x_1, \ldots, x_k))} \;\middle|\; f \in \Sigma^k \right\} \end{aligned}$$

The first and second rule allow the adversary to receive messages from the protocol and send messages to the protocol. The $\mathsf{K}(x)$ action in the second rule makes the messages sent by the adversary observable in a protocol's trace. We exploit this to specify secrecy

properties. The third and fourth rule allow the adversary to learn public names and to generate fresh names by himself. The remaining rules allow the adversary to apply functions from Σ to known messages. In their definition, we write Σ^k to denote the set of all k-ary function symbols in the signature Σ.

In principle, we could define security protocols as multiset rewriting systems containing the message deduction rules MD_Σ. To simplify the development of our verification theory, we however restrict the use of the special fact symbols in protocol rules such that they are only used according to their intended meaning.

A *protocol rule* is a multiset rewriting rule $l -[\, a \,]\!\to r$ such that

P1 K and Out facts do not occur in l and

P2 K and In facts do not occur in r.

A *protocol* P is a finite set of protocol rules. In the context of a signature Σ, its corresponding multiset rewriting system is $P \cup MD_\Sigma$ and its execution modulo an equational theory E is given by $traces_E(P \cup MD_\Sigma)$.

Note that our formal notion of a protocol encompasses both the rules executed by the honest participants and the adversary's capabilities, like revealing long-term keys. We illustrate our protocol model on the following artificial protocol, which we use as a running example in the rest of this thesis.

Example 18 (Artificial Protocol)**.** We consider a protocol that runs in the presence of an adversary that can reveal short-term keys. We model this adversary by marking every short-term key k using a linear fact $\mathsf{Key}(k)$ and adding $[\mathsf{Key}(k)] -[\, \mathsf{Rev}(k) \,]\!\to [\mathsf{Out}(k)]$ to the protocol rules. The Rev-action of this rule logs every key-reveal performed by the adversary in the trace. We use these logged Rev-actions to later exclude disallowed key-reveals in the formalization of our protocol's security properties.

Informally, our protocol works as follows. It chooses a fresh message $x \in FN$ and a fresh short-term key $k \in FN$ upon its start. It then sends the encryption of x with k on the public channel and waits until it receives the pair $\langle x, x \rangle$ on the public channel. Once it receives this pair, it stops and states that it finished a run of the protocol with message x and key k.

We consider this protocol artificial, as it can only finish a run if the adversary performs a key reveal, decrypts the sent message, and constructs the expected pair. We provide example formalizations of non-artificial protocols in Section 7.5.

Using the signature Σ_{PHS} and the equational theory E_{PHS} from Example 17, we formalize our artificial protocol together with the adversary's capability to reveal short-term keys as

$$P_{Ex} := \left\{ \frac{\mathsf{Fr}(x) \quad \mathsf{Fr}(k)}{\mathsf{St}(x,k) \quad \mathsf{Out}(\mathrm{enc}(x,k)) \quad \mathsf{Key}(k)}\,, \quad \frac{\mathsf{St}(x,k) \quad \mathsf{In}(\langle x, x \rangle)}{}[\mathsf{Fin}(x,k)]\,, \quad \frac{\mathsf{Key}(k)}{\mathsf{Out}(k)}[\mathsf{Rev}(k)] \right\}.$$

The rules of P_{Ex} almost immediately follow from its informal description. The linear fact symbols St, Key, Fin, and Rev are protocol specific and their semantics is given by

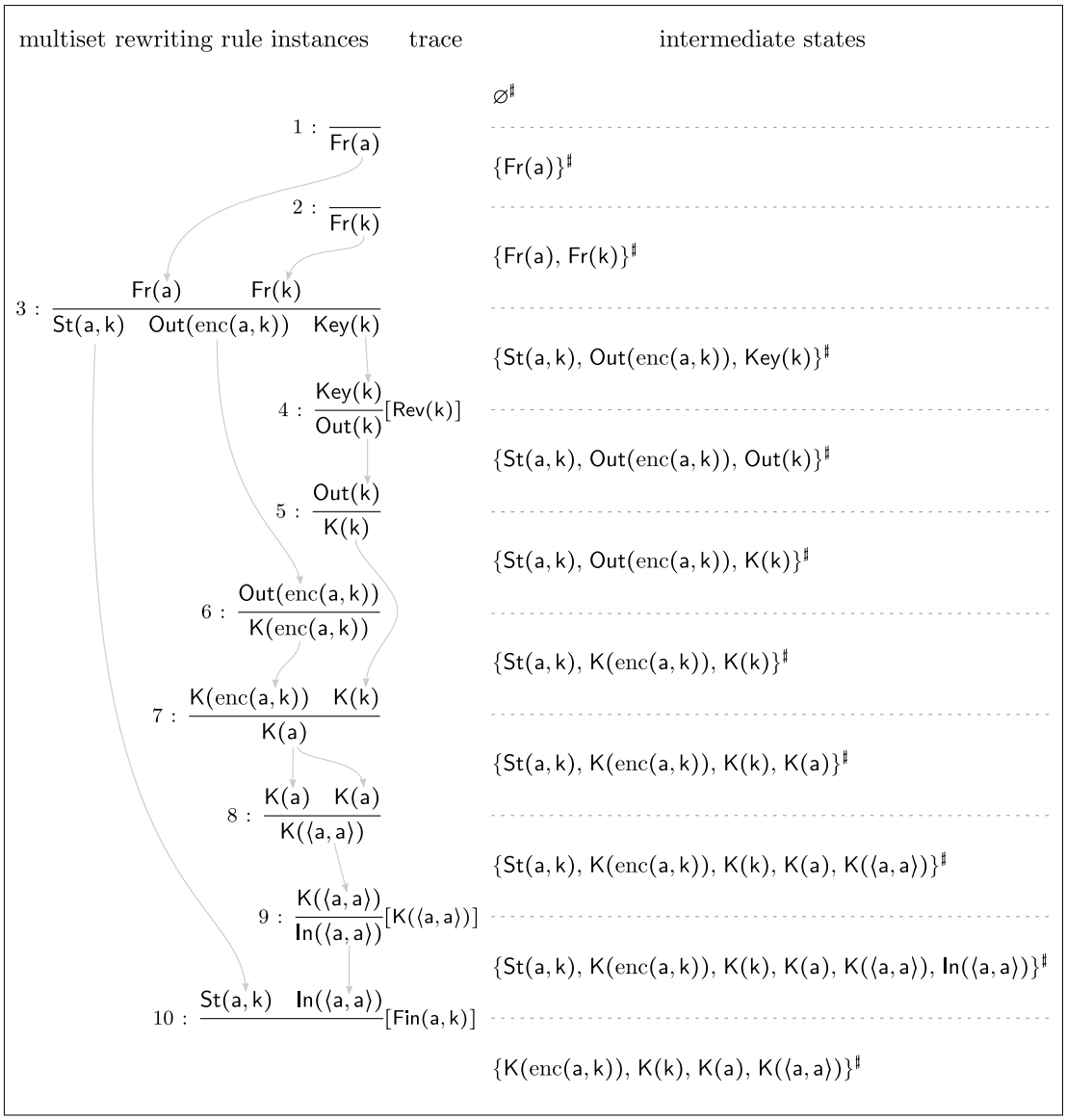


Figure 7.1.: An execution of the P_{Ex} protocol with $\mathsf{a}, \mathsf{k} \in FN$.

the multiset rewriting rules in P_{Ex}. Intuitively, we use St to store the internal state of a participant after it executed the first step. The Key fact symbol marks keys that can be revealed. We use the Fin and Rev fact symbols to log actions required to formalize our security properties. The set of all traces of P_{Ex} is $traces_{E_{PHS}}(P_{Ex} \cup MD_{\Sigma_{PHS}})$.

An example of an execution of the P_{Ex} protocol is depicted in Figure 7.1. On the left-hand side, we depict the instances of the multiset rewriting rule instances used to rewrite the multisets of facts on the right-hand side. We numbered the rule instances to simplify referencing them. The actions of the instances 4, 9, and 10 give rise to the trace of the depicted execution, which is $[\{\mathsf{Rev(k)}\}, \{\mathsf{K}(\langle \mathsf{a}, \mathsf{a} \rangle)\}, \{\mathsf{Fin(a,k)}\}]$.

Note that we also depict the causal dependencies between the conclusions and premises

of the different multiset rewriting rule instances using light-gray arrows. We call the graph comprising all multiset rewriting rule instances and their causal dependencies a dependency graph. In the following chapter, we show that dependency graphs provide an alternative representation of the execution of a multiset rewriting system, which is well-suited for reasoning about a system's security properties. ♠

7.4. Security Properties

We use two-sorted first-order-logic to specify security properties. This logic supports quantification over both messages and timepoints. We thus introduce the sort *temp* for timepoints and write $\mathcal{V}_{temp}$ for the set of *temporal variables*.

A *trace atom* is either *false* $\bot$, a *term equality* $t_1 \approx t_2$, a *timepoint ordering* $i \lessdot j$, a *timepoint equality* $i \doteq j$, or an *action* $f@i$ for a fact f and a timepoint i. A *trace formula* is a first-order formula over trace atoms.

To define the semantics of trace formulas, we associate a domain $\mathbf{D}_s$ with each sort s. The domain for temporal variables is $\mathbf{D}_{temp} := \mathbb{Q}$ and the domains for messages are $\mathbf{D}_{msg} := \mathcal{M}$, $\mathbf{D}_{fresh} := FN$, and $\mathbf{D}_{pub} := PN$. We say that a function θ from $\mathcal{V}$ to $\mathbb{Q} \cup \mathcal{M}$ is a *valuation* if it respects sorts, i.e., $\theta(\mathcal{V}_s) \subseteq \mathbf{D}_s$ for all sorts s. For a term t, we write $t\theta$ for the application of the homomorphic extension of θ to t.

For an equational theory E, the *satisfaction relation* $(tr, \theta) \vDash_E \varphi$ between traces tr, valuations θ, and trace formulas φ is defined as follows.

$(tr, \theta) \vDash_E \bot$	never
$(tr, \theta) \vDash_E f@i$	iff $\theta(i) \in idx(tr)$ and $f\theta \in_E tr_{\theta(i)}$
$(tr, \theta) \vDash_E i \lessdot j$	iff $\theta(i) < \theta(j)$
$(tr, \theta) \vDash_E i \doteq j$	iff $\theta(i) = \theta(j)$
$(tr, \theta) \vDash_E t_1 \approx t_2$	iff $t_1\theta =_E t_2\theta$
$(tr, \theta) \vDash_E \neg\varphi$	iff not $(tr, \theta) \vDash_E \varphi$
$(tr, \theta) \vDash_E \varphi \wedge \psi$	iff $(tr, \theta) \vDash_E \varphi$ and $(tr, \theta) \vDash_E \psi$
$(tr, \theta) \vDash_E \exists x{:}s.\, \varphi$	iff there is $u \in \mathbf{D}_s$ such that $(tr, \theta[x \mapsto u]) \vDash_E \varphi$

The semantics of the remaining logical connectives and quantifiers are defined by translation to the given fragment as usual.

Let $Tr \subseteq (\mathcal{P}(\mathcal{G}))^*$ be a set of traces. We define that φ is *valid for* Tr *modulo* E, written $Tr \vDash^{\forall}_E \varphi$, iff $(tr, \theta) \vDash_E \varphi$ for every trace $tr \in Tr$ and every valuation θ. We define that φ is *satisfiable for* Tr *modulo* E written $Tr \vDash^{\exists}_E \varphi$, iff there exists a trace $tr \in Tr$ and a valuation θ such that $(tr, \theta) \vDash_E \varphi$. Note that validity and satisfiability are dual in the sense that $Tr \vDash^{\forall}_E \varphi$ iff not $Tr \vDash^{\exists}_E \neg\varphi$.

In most cases, we are interested whether a trace formula φ is valid (satisfiable) for all (some) traces of a multiset rewriting system. Overloading notation, we thus define that φ is R, E*-valid*, written $R \vDash^{\forall}_E \varphi$, iff φ is valid for $traces_E(R)$ modulo E. We define that φ is R, E*-satisfiable*, written $R \vDash^{\exists}_E \varphi$, iff φ is satisfiable for $traces_E(R)$ modulo E.

When analyzing security protocols, we use satisfiability claims to formalize that there exists at least one execution of a protocol satisfying certain properties, e.g., that there

exists an execution with honest agents only. We use validity claims to formalize that all executions of a protocol satisfy certain properties, e.g., that the session-key computed by two honest agents is never known to the adversary. We illustrate this in the following example.

Example 19. We formalize two statements about the P_{Ex} protocol. The first statement is that there exists an execution of the P_{Ex} protocol that contains a finishing run. We formalize this statement as

$$P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\exists}_{E_{PHS}} \exists x\, k\, i.\, \mathsf{Fin}(x,k)@i\ .$$

The second statement is that, whenever a run of the P_{Ex} protocol finishes with message x and key k, then key k must have been revealed before this run finished. We formalize this statement as

$$P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\forall}_{E_{PHS}} \forall x\, k\, i.\, \mathsf{Fin}(x,k)@i \Rightarrow \exists j.\, \mathsf{Rev}(k)@j \wedge j \lessdot i\ .$$

Intuitively, both of these statements hold, as the intruder must reveal the short-term key to construct the expected pair. Formally proving the first statement is rather simple, as we just have to exhibit a satisfying trace. Formally proving the second statement without any additional theory is however quite challenging, as we have to reason about all possible executions of the protocol, which includes reasoning about message deduction modulo E_{PHS}. In Chapter 8, we therefore develop a verification theory for proving such statements. ♠

7.5. Extended Examples

In the following two subsections, we illustrate our security protocol model on two protocols from the literature: the NAXOS protocol [LLM07], a Diffie-Hellman based authenticated key exchange protocol, and the TESLA protocol [PCTS00], a stream authentication protocol.

7.5.1. The NAXOS Protocol

We use the NAXOS protocol as an example to illustrate the modeling of the constructions and goals underlying recent Authenticated Key Exchange (AKE) protocols. Figure 7.2 depicts the protocol. Each party x has a long-term private key lk_x and a corresponding public key $pk_x = g^{lk_x}$, where g is a generator of the DH group. To start a session, the initiator $\mathcal{I}$ first creates a fresh nonce $esk_{\mathcal{I}}$, also known as $\mathcal{I}$'s ephemeral (private) key. He then concatenates $esk_{\mathcal{I}}$ with $\mathcal{I}$'s long-term private key $lk_{\mathcal{I}}$, hashes the result using h_1, and sends $g^{h_1(esk_{\mathcal{I}}, lk_{\mathcal{I}})}$ to the responder. The responder $\mathcal{R}$ stores the received value in a variable X, computes a similar value based on his own nonce $esk_{\mathcal{R}}$ and long-term private key $lk_{\mathcal{R}}$, and sends the result to the initiator, who stores the received value in the variable Y. Finally, both parties compute a session key ($k_{\mathcal{I}}$ and $k_{\mathcal{R}}$, respectively) whose computation includes their own long-term private keys, such that only the intended partner can compute the same key.

Note that the messages exchanged are not authenticated, as the recipients cannot verify that the expected long-term key was used in the construction of the message.

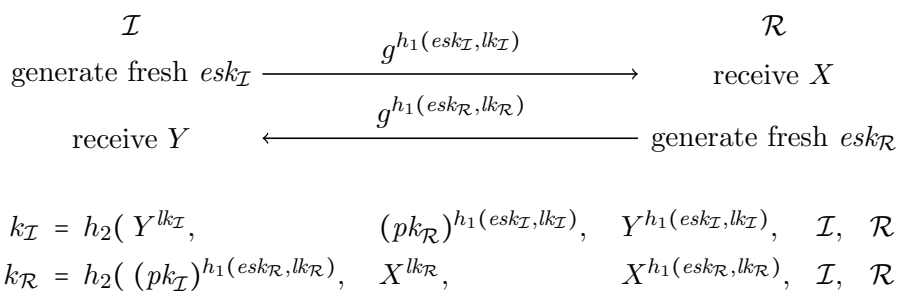


Figure 7.2.: The NAXOS protocol.

The authentication is implicit and only guaranteed through ownership of the correct key. Explicit authentication (e.g., the intended partner was recently alive or agrees on some values) is commonly achieved in AKE protocols by adding a key-confirmation step, where the parties exchange a MAC of the exchanged messages that is keyed with (a variant of) the computed session key.

The key motivation behind recent AKE protocols is that they should achieve their security goals even in the presence of very strong adversaries. For example, the NAXOS protocol is designed to be secure in the eCK security model [LLM07]. In this model, as in the standard Dolev-Yao model, the adversary has complete control over the network and can learn the long-term private keys of all dishonest agents. However, unlike in the Dolev-Yao model, he can additionally, under some restrictions, learn the long-term private key of any agent. This models (weak) *Perfect Forward Secrecy* (wPFS/PFS): even if the adversary learns the long-term private keys of all the agents, the keys of previous sessions should remain secret [Kra05]. Additionally, this models resilience against *Key Compromise Impersonation* (KCI): even if the adversary learns the long-term private key of an agent, he should be unable to impersonate as anybody to this agent [JV96]. Moreover, the adversary can learn the session keys of certain sessions. This models both *Key Independence* (KI), where compromising one session key should not compromise other keys, and resilience against *unknown-key share attacks* (UKS), where the adversary should not be able to trick other sessions into computing the same key. Finally, the adversary can learn any agent's ephemeral keys. This models resilience against corrupted random-number generators. All these attack types are modeled in the eCK security model.

Formal Model of Diffie-Hellman Exponentiation

To model the NAXOS protocol, we need a model of Diffie-Hellman (DH) exponentiation. We model the operations of a single DH group $\mathbb{G}$ using the signature

$$\Sigma_{DH} := \{\_\,\hat{}\,\_,\ \_^{-1},\ \_ * \_,\ 1\}\ ,$$

where all function symbols are of sort $msg \times \ldots \times msg \to msg$. The functions in Σ_{DH} model exponentiation, and inversion, multiplication, and the unit in the group of exponents.

We model the algebraic properties of these functions using the equations

$$E_{DH} := \left\{ \begin{array}{lll} (x \hat{} y) \hat{} z \simeq x \hat{} (y * z), & x \hat{} 1 \simeq x, & x * y \simeq y * x, \\ x * (y * z) \simeq (x * y) * z, & x * 1 \simeq x, & x * x^{-1} \simeq 1 \end{array} \right\} .$$

These equations formalize that repeated exponentiation corresponds to multiplication of the exponents, exponentiation with 1 is the identity, and the exponents form an abelian group. In the remainder of this section, we use g to denote a public name that is used as a fixed generator of the DH group $\mathbb{G}$.

Example 20. Let $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{k}$ denote fresh names and consider the term $((\mathsf{g} \hat{} \mathsf{a}) \hat{} \mathsf{b}) \hat{} \mathsf{a}^{-1}$, which results from exponentiating g with a, followed by b, followed by a inverse. As expected, it holds that $((\mathsf{g} \hat{} \mathsf{a}) \hat{} \mathsf{b}) \hat{} \mathsf{a}^{-1} =_{E_{DH}} \mathsf{g} \hat{} ((\mathsf{a} * \mathsf{b}) * \mathsf{a}^{-1}) =_{E_{DH}} \mathsf{g} \hat{} \mathsf{b}$. ♠

Note that the equational theory E_{DH} does not support protocols that perform multiplication in the DH group $\mathbb{G}$. To define such protocols, an additional function symbol $\circledast$ denoting multiplication in $\mathbb{G}$ is required. The function symbol $*$ denotes multiplication in the group of exponents, which is a different operation. For example, the equality $(\mathsf{g} \hat{} a \circledast \mathsf{g} \hat{} b) \hat{} c = (\mathsf{g} \hat{} (a * c)) \circledast (\mathsf{g} \hat{} (b * c))$ holds in all DH groups, but does not necessarily hold if we replace $\circledast$ by $*$. Moreover, addition of exponents must be modeled for such protocols to avoid missing attacks. Consider for example a protocol that randomly chooses two exponents a and b, sends these exponents, receives some exponent x, and checks if $\mathsf{g} \hat{} a \circledast \mathsf{g} \hat{} b = \mathsf{g} \hat{} x$. This check succeeds iff $x = a + b$. See [Sch12] for further information about our model of DH exponentiation.

Formal Model of the NAXOS Protocol

In our model of the NAXOS protocol, we use terms built over the signature

$$\Sigma_{NAXOS} := \Sigma_{DH} \cup \{\mathrm{h}_1(\_,\_), \mathrm{h}_2(\_,\_,\_,\_)\} .$$

The function symbols h_1 and h_2 model hash functions. The equational theory modeling the properties of the functions in Σ_{NAXOS} is E_{DH}. Note that h_1 and h_2 are free with respect to E_{DH}, which corresponds roughly to modeling them as random oracles.

We formalize the NAXOS protocol for the eCK model by the set of protocol rules P_{NAXOS} defined in Figure 7.3. We prefix persistent facts with ! and consider all other facts except K-facts to be linear by convention. We also use the notational convention that variables without a sorted occurrence are of sort *msg*. The first rule models the generation and registration of long-term asymmetric keys. An exponent lk_A is randomly chosen and stored as the long-term key of an agent A. The persistent facts $!\mathsf{Ltk}(A, lk_A)$ and $!\mathsf{Pk}(A, \mathsf{g} \hat{} lk_A)$ denote the association between A and his long-term private and public keys. The public key is additionally sent to the adversary.

In the rules modeling the initiator and responder, each protocol thread chooses a unique ephemeral key esk_x, which we also use to identify the thread. The first initiator rule chooses the actor I and the intended partner R, looks up I's long-term key, and sends the half-key hk_I. The fact $\mathsf{Init}_1(esk_I, I, R, lk_I, hk_I)$ then stores the state of thread esk_I and the fact $!\mathsf{Ephk}(esk_I, esk_I)$ is added to allow the adversary to reveal the ephemeral key esk_I (the second argument) of the thread esk_I (the first argument). The second initiator

Generate long-term keypair:

$$\frac{\mathsf{Fr}(lk_A)}{!\mathsf{Ltk}(A{:}pub, lk_A) \quad !\mathsf{Pk}(A, \mathsf{g} \hat{} lk_A) \quad \mathsf{Out}(\mathsf{g} \hat{} lk_A)}$$

Initiator step 1:

$$\frac{\mathsf{Fr}(esk_I) \quad !\mathsf{Ltk}(I, lk_I)}{\mathsf{Init}_1(esk_I, I, R{:}pub, lk_I, hk_I) \quad !\mathsf{Ephk}(esk_I, esk_I) \quad \mathsf{Out}(hk_I)}$$

where $hk_I := \mathsf{g} \hat{} \mathrm{h}_1(esk_I, lk_I)$

Initiator step 2:

$$\frac{\mathsf{Init}_1(esk_I, I, R, lk_I, hk_I) \quad !\mathsf{Pk}(R, pk_R) \quad \mathsf{In}(Y)}{!\mathsf{Sessk}(esk_I, k_I)} \left[\begin{array}{l} \mathsf{Accept}(esk_I, I, R, k_I), \\ \mathsf{Sid}(esk_I, \langle \mathsf{Init}, I, R, hk_I, Y \rangle), \\ \mathsf{Match}(esk_I, \langle \mathsf{Resp}, R, I, hk_I, Y \rangle) \end{array}\right]$$

where $k_I := \mathrm{h}_2(Y \hat{} lk_I, pk_R \hat{} \mathrm{h}_1(esk_I, lk_I), Y \hat{} \mathrm{h}_1(esk_I, lk_I), I, R)$

Responder:

$$\frac{\mathsf{Fr}(esk_R) \quad !\mathsf{Ltk}(R, lk_R) \quad !\mathsf{Pk}(I, pk_I) \quad \mathsf{In}(X)}{!\mathsf{Sessk}(esk_R, k_R) \quad !\mathsf{Ephk}(esk_R, esk_R) \quad \mathsf{Out}(\mathsf{g} \hat{} \mathrm{h}_1(esk_R, lk_R))} \left[\begin{array}{l} \mathsf{Accept}(esk_R, R, I, k_R), \\ \mathsf{Sid}(esk_R, \langle \mathsf{Resp}, R, I, X, hk_R \rangle), \\ \mathsf{Match}(esk_R, \langle \mathsf{Init}, I, R, X, hk_R \rangle) \end{array}\right]$$

where $hk_R := \mathsf{g} \hat{} \mathrm{h}_1(esk_R, lk_R)$,
and $k_R := \mathrm{h}_2(pk_I \hat{} \mathrm{h}_1(esk_R, lk_R), X \hat{} lk_R, X \hat{} \mathrm{h}_1(esk_R, lk_R), I, R)$

Key Reveals for the eCK model:

$$\frac{!\mathsf{Sessk}(tid, k)}{\mathsf{Out}(k)}[\mathsf{SesskRev}(tid)] \qquad \frac{!\mathsf{Ltk}(A, lk_A)}{\mathsf{Out}(lk_A)}[\mathsf{LtkRev}(A)] \qquad \frac{!\mathsf{Ephk}(tid, esk_A)}{\mathsf{Out}(esk_A)}[\mathsf{EphkRev}(tid)]$$

Figure 7.3.: The multiset rewriting rules defining P_{NAXOS}

rule reads the thread's state, looks up the public key of the intended partner, and receives the half-key Y. The key k_I is then computed. The action $\mathsf{Accept}(esk_I, I, R, k_I)$ denotes that the thread esk_I finished with the given parameters.

To specify when two threads are intended communication partners ("matching sessions"), we include $\mathsf{Sid}(esk_I, sid)$ and $\mathsf{Match}(esk_I, sid')$ actions. A thread s' matches a thread s if there exists a sid such that $\mathsf{Sid}(s', sid)$ and $\mathsf{Match}(s, sid)$ occur in the trace. By appropriately defining the Match actions and session identifier sid, various definitions of matching can be modeled [Cre11].

Finally, the fact $!\mathsf{Sessk}(esk_I, k_I)$ is added to the second initiator rule to allow revealing the session key k_I. The responder rule works analogously. The final three rules model that, in the eCK model, the adversary can reveal any session, long-term, or ephemeral key. We model the restrictions on key reveals as part of the security property and thus

$$\varphi_{\text{eCK}} := \forall i_1\, i_2\, s\, A\, B\, k.\, (\mathsf{Accept}(s, A, B, k)@i_1 \wedge \mathsf{K}(k)@i_2) \Rightarrow$$

// If the session key of the test thread s is known, then s must be "not clean".
// Hence either there is a session key reveal for s,
$$\Big((\exists i_3.\, \mathsf{SesskRev}(s)@i_3)$$

// or a session key reveal for a matching session,
$$\vee\, (\exists s' i_3\, i_4\, sid.\, (\mathsf{Sid}(s', sid)@i_3 \wedge \mathsf{Match}(s, sid)@i_4) \wedge (\exists i_5.\, \mathsf{SesskRev}(s')@i_5))$$

// or if a matching session exists,
$$\vee\, \Big(\exists s' i_3\, i_4\, sid.\, (\mathsf{Sid}(s', sid)@i_3 \wedge \mathsf{Match}(s, sid)@i_4)$$

// both lk_A and esk_A, or both lk_B and esk_B are revealed,
$$\wedge\, \big((\exists i_5\, i_6.\, \mathsf{LtkRev}(A)@i_5 \wedge \mathsf{EphkRev}(s)@i_6)$$
$$\vee\, (\exists i_5\, i_6.\, \mathsf{LtkRev}(B)@i_5 \wedge \mathsf{EphkRev}(s')@i_6)\big)\Big)$$

// or if no matching session exists,
$$\vee\, \Big(\neg\big(\exists s' i_3\, i_4\, sid.\, (\mathsf{Sid}(s', sid)@i_3 \wedge \mathsf{Match}(s, sid)@i_4)\big)$$

// either both lk_A and esk_A, or lk_B are revealed.
$$\wedge\, \big((\exists i_5\, i_6.\, \mathsf{LtkRev}(A)@i_5 \wedge \mathsf{EphkRev}(s)@i_6) \vee (\exists i_5.\, \mathsf{LtkRev}(B)@i_5)\big)\Big)\ \Big)$$

Figure 7.4.: eCK security definition

record all key reveals in the trace.

We formalize the security of NAXOS in the eCK model by the validity claim

$$P_{NAXOS} \cup MD_{\Sigma_{NAXOS}} \vDash^{\forall}_{E_{DH}} \varphi_{\text{eCK}}\ ,$$

where the trace formula φ_{eCK} is defined in Figure 7.4. Note that φ_{eCK} is a one-to-one mapping of the original definition of eCK security given in [LLM07]. Intuitively, the formula states that if the adversary knows the session key of a thread esk_I, then he must have performed forbidden key reveals. The left-hand side of the implication states that the key k is known and the right-hand side disjunction states the restrictions on key reveals. We describe each disjunct in the comment above it. Further motivation and variants of these restrictions can be found in [LLM07, Cre11].

7.5.2. The TESLA Protocol

The TESLA protocol [PCTS00] is a *stream authentication protocol*, i.e., a protocol that authenticates a continuous stream of packets, broadcast over an unreliable and untrusted medium to a group of receivers. One could use a stream authentication protocol for example to provide an authenticated video stream (e.g., a newscast) over the Internet. An important factor in the design of TESLA is throughput. Therefore, apart from the initial message, the protocol does not make use of expensive cryptographic primitives such as public key signatures. The packets of the stream are authenticated

using message authentication codes (MACs) only. Furthermore, the MACs are cleverly combined with a timed, key commitment and disclosure scheme to ensure a unidirectional, receiver-independent dataflow.

In [PCTS00], two schemes of the TESLA protocol with different message formats are introduced. Scheme 1 generates one fresh MAC key per packet. This scheme does not tolerate any packet loss. Scheme 2 allows for packet-loss using an inverted hash chain. We have modeled both schemes in Tamarin [MS12]. Here, we focus on Scheme 1, as it is slightly simpler.

Following [BL02], the message exchange of Scheme 1 looks as follows from the sender's perspective.

$$
\begin{array}{lll}
\text{Msg 0a.} & S \leftarrow R\colon & n_R \\
\text{Msg 0b.} & S \rightarrow R\colon & \langle \mathrm{h}(k_1), \mathrm{sign}(lk_S, \langle \mathrm{h}(k_1), n_R \rangle) \rangle \\
\text{Msg 1.} & S \rightarrow R\colon & \langle d_1, \mathrm{mac}(k_1, d_1) \rangle \qquad \text{where } d_1 := \langle p_1, \mathrm{h}(k_2) \rangle \\
\text{Msg } n. & S \rightarrow R\colon & \langle d_n, \mathrm{mac}(k_n, d_n) \rangle \qquad \text{where } d_n := \langle p_n, \langle \mathrm{h}(k_{n+1}), k_{n-1} \rangle \rangle
\end{array}
$$

Before sending the stream, the sender accepts setup requests by prospective receivers. Given the nonce n_R, generated by a receiver to ensure freshness, the sender returns a signed commitment, $\mathrm{h}(k_1)$, to the MAC key k_1 that she will use to authenticate the first packet. Before sending the first packet, the sender generates the MAC key k_2 that she will use to authenticate the second packet. In the first message, the sender then sends the commitment to this second key, $\mathrm{h}(k_2)$, together with the first packet p_1. Once a receiver received the first message, he thus has the data d_1 sent in the first message, its corresponding MAC ma_1, a commitment co_1 to the first MAC key, and a commitment co_2 to the second hash key. Only the commitment co_1 is authenticated at this point. The TESLA protocol delays the authentication of data to achieve receiver-independence despite the use of symmetric cryptography. After the receipt of the first message, the receiver's state is setup for the receipt of subsequent packets.

In the n-the step, the sender discloses the MAC key k_{n-1} that was used to authenticate the data of step $n-1$. Receivers can then check whether this key corresponds to the commitment co_{n-1}, and thereby authenticate this key. If this check succeeds, the receivers can also authenticate the data d_{n-1} by validating it against the received MAC ma_{n-1}. To ensure the security of key disclosure, the TESLA protocol *assumes* that the clocks of the sender and the receivers are loosely synchronized, and that a receiver does not accept a commitment to a key if it might have been disclosed already. Provided that this security assumption holds, the TESLA protocol guarantees *stream authentication*, i.e., that no receiver authenticates data that was not sent by the server.

We formalize the statement that Scheme 1 of the TESLA protocol provides stream authentication as follows. We define the signature Σ_{TESLA} and the equations E_{TESLA} as

$$\Sigma_{TESLA} := \{\langle \_, \_ \rangle, \mathrm{fst}(\_), \mathrm{snd}(\_), \mathrm{h}(\_), \mathrm{mac}(\_, \_), \mathrm{sign}(\_, \_), \mathrm{pk}(\_), \mathrm{verify}(\_, \_, \_), \mathrm{true}\}$$

$$E_{TESLA} := \{\mathrm{fst}(\langle x, y \rangle) \simeq x,\ \mathrm{snd}(\langle x, y \rangle) \simeq y,\ \mathrm{verify}(\mathrm{pk}(k), \mathrm{sign}(k, m), m) \simeq \mathrm{true}\}\ .$$

Using Σ_{TESLA} and E_{TESLA}, we can model terms built from pairing, projection, hashing, keyed MACs, and message-hiding signatures. A term $\mathrm{sign}(k, m)$ denotes the result of signing the message m with the private-key k. A term $\mathrm{pk}(k)$ denotes the public-key

corresponding to the private-key k, and a term $\mathrm{verify}(x, sig, m)$ denotes a call to the signature verification algorithm with public-key x, signature sig, and signed message m. If this signature verification succeeds (i.e., if $x = \mathrm{pk}(k)$ and $sig = \mathrm{sign}(k, m)$), then $\mathrm{verify}(x, sig, m)$ returns the constant true, as formalized by the last equation in E_{TESLA}.

The rules given in Figure 7.5 define the protocol P_{TESLA}, which formalizes Scheme 1 of the TESLA protocol. The first two rules model the public-key infrastructure. As in our other protocol models, we represent agent identities by public names. The first rule models that an agent A generates a fresh private key lk_A, stores it in his table of long-term private keys, and registers the corresponding public key $\mathrm{pk}(A)$ with the certificate authority, which publishes it. Our use of the !Pk-facts corresponds to modeling that certificates are predistributed using an authentic channel. The second rule models that the adversary may compromise the private key of any agent. The LtkRev-action logs this compromise in the trace such that we can refer to it in the specification of stream authentication.

The third and fourth rule model the setup of a new authenticated stream by some sender S. The third rule models that the sender generates a fresh key k_1, stores it for the later sending of the first message, $\mathsf{Snd_1}(S, k_1)$, and forks a process that answers setup requests by receivers, $!\mathsf{Snd_{0a}}(S, k_1)$. The fourth rule models the answering of a receiver's setup request.

The fifth and sixth rule model the setup of a receiver. We model equality checks by adding an Eq-action and focusing on only the traces that satisfy the trace formula

$$\alpha_{\mathsf{Eq}} := (\forall x\ y\ i.\ \mathsf{Eq}(x, y)@i \Rightarrow x \approx y)\ .$$

Said differently, we filter out all traces that contain an $\mathsf{Eq}(t, s)$ action with $t \neq_{E_{TESLA}} s$. We use the nonce n_R generated by a receiver to identify him. The action $\mathsf{Setup}(n_R)$ in the sixth rule states that the receiver n_R is setup for receiving data from the sender. We use this action later in our formalization of stream authentication.

The seventh and eight rule model the sending of the packets of a stream, as described before. We use Sent-actions to mark the sent data. We moreover mark a key commitment as expired whenever its corresponding key is disclosed. We use this to formalize the security assumption underlying the TESLA protocol.

Finally, the last two rules model the receipt of an authenticated stream of packets, as described before. The required checks are performed by the last rule, which works as follows. It retrieves the state $\mathsf{Rcv_n}(n_R, S, d_{n-1}, ma_{n-1}, co_{n-1}, co_n)$ from the previous step and waits for the receipt of a message of the form $\langle\langle p_n, \langle co_{n+1}, k_{n-1}\rangle\rangle, ma_n\rangle$. Upon the receipt of such a message, it checks that the commit co_n is not yet expired. In a timed model, this check would be performed by checking whether the current time is less than the expected disclosure time of key k_n. As we do not model time explicitly, we use the trace formula

$$\alpha_{\mathsf{NotExpired}} := \forall n_R\ co\ j_{ne}\ j_e.\ \mathsf{NotExpired}(n_R, co)@j_{ne} \wedge \mathsf{Expired}(co)@j_e \Rightarrow j_{ne} \lessdot j_e$$

to focus on the traces where all non-expiredness checks succeed. We use Eq-actions to implement the equality checks and we use the $\mathsf{Auth}(n_R, S, d_{n-1})$ action to log that the receiver n_R claims that data d_{n-1} was sent by server S.

Public-key infrastructure:

$$\frac{\mathsf{Fr}(lk_A)}{!\mathsf{Ltk}(A{:}pub, lk_A) \quad !\mathsf{Pk}(A, \mathrm{pk}(lk_A)) \quad \mathsf{Out}(\mathrm{pk}(lk_A))} \qquad \frac{!\mathsf{Ltk}(A, lk_A)}{\mathsf{Out}(lk_A)}[\mathsf{LtkRev}(A)]$$

Sender setup:

$$\frac{\mathsf{Fr}(k_1)}{\mathsf{Snd_1}(S{:}pub, k_1) \quad !\mathsf{Snd_{0a}}(S, k_1)} \qquad \frac{!\mathsf{Snd_{0a}}(S, k_1) \quad \mathsf{In}(\langle S, n_R\rangle) \quad !\mathsf{Ltk}(S, lk_S)}{\mathsf{Out}(\langle \mathrm{h}(k_1), \mathrm{sign}(lk_S, \langle \mathrm{h}(k_1), n_R\rangle)\rangle)}$$

Receiver setup:

$$\frac{\mathsf{Fr}(n_R)}{\mathsf{Rcv_{0b}}(n_R, S{:}pub) \quad \mathsf{Out}(\langle S, n_R\rangle)}$$

$$\frac{\mathsf{Rcv_{0b}}(n_R, S) \quad \mathsf{In}(\langle co_1, sig\rangle) \quad !\mathsf{Pk}(S, pk_S)}{\mathsf{Rcv_1}(n_R, S, co_1)}\left[\begin{array}{l}\mathsf{Eq}(\mathrm{verify}(pk_S, sig, \langle co_1, n_R\rangle), \mathrm{true}),\\ \mathsf{Setup}(n_R)\end{array}\right]$$

Stream sending: for $d_1 := \langle p_1, \mathrm{h}(k_2)\rangle$ and $d_n := \langle p_n, \langle \mathrm{h}(k_{n+1}), k_{n-1}\rangle\rangle$

$$\frac{\mathsf{Snd_1}(S, k_1) \quad \mathsf{Fr}(p_1) \quad \mathsf{Fr}(k_2)}{\mathsf{Snd_n}(S, k_1, k_2) \quad \mathsf{Out}(\langle d_1, \mathrm{mac}(k_1, d_1)\rangle)}[\mathsf{Sent}(S, d_1)]$$

$$\frac{\mathsf{Snd_n}(S, k_{n-1}, k_n) \quad \mathsf{Fr}(p_n) \quad \mathsf{Fr}(k_{n+1})}{\mathsf{Snd_n}(S, k_n, k_{n+1}) \quad \mathsf{Out}(\langle d_n, \mathrm{mac}(k_n, d_n)\rangle)}[\mathsf{Sent}(S, d_n), \mathsf{Expired}(\mathrm{h}(k_{n-1}))]$$

Stream receiving: for $d_1 := \langle p_1, co_2\rangle$ and $d_n := \langle p_n, \langle co_{n+1}, k_{n-1}\rangle\rangle$

$$\frac{\mathsf{Rcv_1}(n_R, S, co_1) \quad \mathsf{In}(\langle d_1, ma_1\rangle)}{\mathsf{Rcv_n}(n_R, S, d_1, ma_1, co_1, co_2)}$$

$$\frac{\mathsf{Rcv_n}(n_R, S, d_{n-1}, ma_{n-1}, co_{n-1}, co_n) \quad \mathsf{In}(\langle d_n, ma_n\rangle)}{\mathsf{Rcv_n}(n_R, S, d_n, ma_n, co_n, co_{n+1})}\left[\begin{array}{l}\mathsf{NotExpired}(n_R, co_n),\\ \mathsf{Eq}(co_{n-1}, \mathrm{h}(k_{n-1})),\\ \mathsf{Eq}(ma_{n-1}, \mathrm{mac}(k_{n-1}, d_{n-1})),\\ \mathsf{Auth}(n_R, S, d_{n-1})\end{array}\right]$$

Figure 7.5.: The multiset rewriting rules defining P_{TESLA}

A first attempt at formalizing that Scheme 1 of the TESLA protocol satisfies stream authentication, provided its security assumption holds, is the validity claim

$$P_{TESLA} \cup MD_{\Sigma_{TESLA}} \vDash^{\forall}_{E_{TESLA}} \alpha_{\mathsf{Eq}} \wedge \alpha_{\mathsf{NotExpired}} \Rightarrow \varphi'_{\mathsf{Auth}}$$

$$\text{for } \varphi'_{\mathsf{Auth}} := \forall n_R\ S\ d\ i.\ \mathsf{Auth}(n_R, S, d)@i \Rightarrow (\exists j.\ \mathsf{Sent}(S, d)@j) \vee (\exists j.\ \mathsf{LtkRev}(S)@j)\ .$$

This claim states that stream authentication is guaranteed under the assumption that the non-expiredness checks of *all* receivers succeed and the sender's long-term key was *never* revealed. This claim holds, but we prefer a statement that only excludes failed non-expiredness checks for the receiver making the authentication claim. Moreover, we also expect that it is sufficient to exclude reveals of the sender's long-term key *before* the receiver was setup. We formalize this version of stream authentication by the validity claim

$$P_{TESLA} \cup MD_{\Sigma_{TESLA}} \vDash^{\forall}_{E_{TESLA}} \alpha_{\mathsf{Eq}} \Rightarrow \varphi_{\mathsf{Auth}}$$

$$\begin{aligned}\text{for } \varphi_{\mathsf{Auth}} := \forall n_R\ S\ d\ i.\ & \mathsf{Auth}(n_R, S, d)@i \Rightarrow \\ & (\exists j.\ \mathsf{Sent}(S, d)@j) \\ (1) \quad \vee\ & (\exists j_s\ j_r.\ \mathsf{Setup}(n_R)@j_s \wedge \mathsf{LtkRev}(S)@j_r \wedge j_r \lessdot j_s) \\ (2) \quad \vee\ & (\exists co\ j_e\ j_{ne}.\ \mathsf{NotExpired}(n_R, co)@j_{ne} \wedge \\ & \qquad \mathsf{Expired}(co)@j_e \wedge j_e \lessdot j_{ne} \wedge j_{ne} \lessdot i).\end{aligned}$$

This claim states that all data authenticated by a receiver n_R was sent by the claimed sender, provided that (1) the sender's long-term key was not revealed before receiver n_R received the first key commitment and (2) no earlier expiredness check of receiver n_R failed. This claim also holds and its proof represents strong evidence that Scheme 1 of the TESLA protocol works as intended, even in the presence of a strong adversary. To reduce the chance of modeling errors, we further verify that an authentication claim is reachable under the assumption that all non-expiredness checks succeed; i.e., we verify the satisfiability claim

$$P_{TESLA} \cup MD_{\Sigma_{TESLA}} \vDash^{\exists}_{E_{TESLA}} \alpha_{\mathsf{Eq}} \wedge \alpha_{\mathsf{NotExpired}} \wedge (\exists n_R\ S\ d\ i.\ \mathsf{Auth}(n_R, S, d)@i)\ .$$

The Tamarin prover verifies both the validity of φ_{Auth} and this executability check in less than three seconds, as shown in Section 9.3.

8. Verification Theory

In this chapter, we give a calculus for proving and disproving R, E-satisfiability and R, E-validity claims. We exploit the duality of validity and satisfiability to convert all validity claims to satisfiability claims. We reason about satisfiability claims using constraint solving to perform an exhaustive, symbolic search for satisfying traces. We use constraint systems to represent the intermediate states of our search and we give a set of constraint-reduction rules that solve individual constraints in these systems. The solving of individual constraints corresponds to an incremental construction of a satisfying trace. We develop and explain our calculus in four sections, which we outline below.

In Section 8.1, we introduce dependency graphs, which are a more precise representation of the execution of multiset rewriting systems than (action-)traces. They consist of the sequence of rewriting rule instances corresponding to a system's execution and their causal dependencies, similar to strand spaces [THG99]. We use dependency graphs to denote the solutions of our constraint systems.

In Section 8.2 we introduce guarded trace properties, which are an expressive subset of trace formulas. We then show how to use constraint solving to check R, E-satisfiability of guarded trace properties modulo equational theories for which a complete and finitary unification algorithm exists. This constraint solving can be seen as a basic backwards reachability analysis. It works well for reasoning about non-looping parts of protocols, e.g., the linear roles of security protocols like NAXOS [LLM07] (described in Section 7.5.1). However, it does not suffice to effectively reason about message deduction or looping parts of protocols, e.g., the streaming of the authenticated packets in the TESLA protocol [PCTS00] (described in Section 7.5.2). We remedy these problems in sections 8.3 and 8.4.

In Section 8.3, we show how to exploit induction over the traces of a multiset rewriting system to improve the effectiveness of our constraint-reduction rules. In our case studies, we exploit trace induction to effectively reason about several types of loops, e.g., the ones occurring in the TESLA protocol.

In Section 8.4, we give additional constraint-reduction rules that allow us to effectively reason about message deduction. We develop these rules by generalizing the ideas behind decryption-chain reasoning from Part I to certain equational theories, e.g., a theory modeling Diffie-Hellman exponentiation combined with an arbitrary subterm-convergent rewriting theory.

We do not have a formal characterization of the set of protocols that can be analyzed using the constraint-reduction rules given in this chapter. We therefore validate their usefulness on a wide range of case studies in Section 9.3. Moreover, as none of our rules is specifically tailored to an individual case study, we expect our rules to be sufficient to analyze many further case studies that rely on correctness arguments similar to the ones underlying our case studies. Nevertheless, there will always be protocols that one

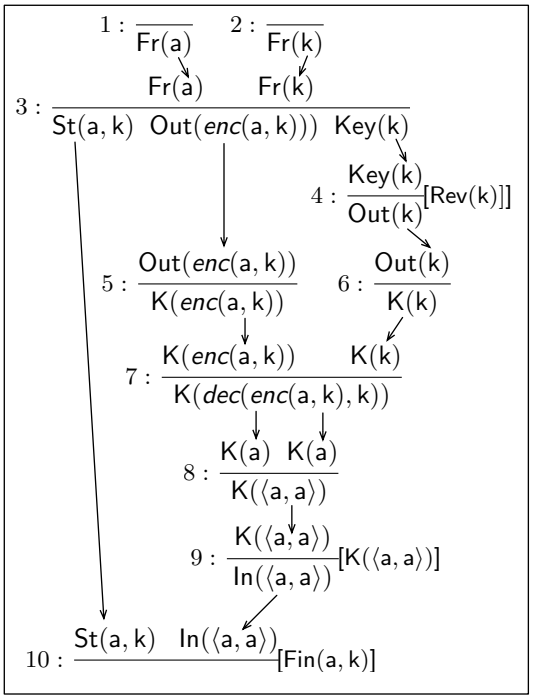


Figure 8.1.: Dependency graph modulo E_{PHS}.

cannot analyze using our constraint-reduction rules, as R, E-satisfiability is undecidable in general. For such protocols, one may use sections 8.3 and 8.4 as blueprints for the development of further constraint-reduction rules that codify the key correctness arguments underlying these protocols.

Note that many of our notions are parametrized by an equational theory and a set of multiset rewriting rules; e.g., R, E-traces or R, E-satisfiability. In the rest of this thesis, we take the liberty to omit these parameters when they are clear from the context. We moreover use the convention that, by default, R denotes a multiset rewriting system, P a protocol, and E an equational theory.

8.1. Dependency Graphs

We use dependency graphs to represent protocol executions together with their causal dependencies. A dependency graph consists of nodes labeled with rule instances and edges represent dependencies between the nodes. We first present an example of a dependency graph and then give its formal definition.

Example 21 (Dependency Graph)**.** Figure 8.1 shows a dependency graph for an execution of P_{Ex} from Example 18. Note that all rule instances are ground. Nodes 1 and 2 are rule instances that create fresh names. Node 3 is an instance of the first protocol rule. Node 4 is an instance of the key reveal rule. Nodes 5–9 are instances of message deduction rules and denote that the adversary receives a ciphertext and its key, decrypts the ciphertext, pairs the resulting cleartext with itself, and sends the

result to an instance of the second protocol rule, Node 10. The edges denote causal dependencies: an edge from a conclusion of node i to a premise of node j denotes that the corresponding fact is generated by i and consumed by j. Since this is a dependency graph modulo E_{PHS}, it is sufficient that each pair of generated and consumed facts is equal modulo E_{PHS}. ♠

Formally, let E be an equational theory and R be a multiset rewriting system. We say that $dg := (I, D)$ is an R, E-*dependency graph* if $I \in \mathit{ginsts}(R \cup \{\text{FRESH}\})^*$, $D \subseteq \mathbb{N}^2 \times \mathbb{N}^2$, and dg satisfies the conditions **DG1–4** listed below. To state these conditions, we introduce the following definitions. We call $\mathit{idx}(I)$ the *nodes* and D the *edges* of dg. We write $(i, u) \nrightarrow (j, v)$ for the edge $((i, u), (j, v))$. Let $I := [l_1 -[\ a_1\ ]\rightarrow r_1, \ldots, l_n -[\ a_n\ ]\rightarrow r_n]$. The *trace* of dg is $\mathit{trace}(dg) := [\mathit{set}(a_1), \ldots, \mathit{set}(a_n)]$. A *conclusion* of dg is a pair (i, u) such that i is a node of dg and $u \in \mathit{idx}(r_i)$. The corresponding *conclusion fact* is $(r_i)_u$. A *premise* of dg is a pair (i, u) such that i is a node of dg and $u \in \mathit{idx}(l_i)$. The corresponding *premise fact* is $(l_i)_u$. A conclusion or premise is *linear* if its fact is linear.

DG1 For every edge $(i, u) \nrightarrow (j, v) \in D$, it holds that $i < j$ and the conclusion fact of (i, u) is equal modulo E to the premise fact of (j, v).

DG2 Every premise of dg has exactly one incoming edge.

DG3 Every linear conclusion of dg has at most one outgoing edge.

DG4 The FRESH rule instances in I are unique.

We denote the set of all R, E-dependency graphs by $\mathit{dgraphs}_E(R)$.

Note that dependency graphs provide us with an alternative formulation of the multiset rewriting semantics given in Chapter 7, as formalized by the following theorem. We exploit this alternative semantics in our backwards reachability analysis by incrementally constructing dependency graphs instead of (action-)traces.

Theorem 3. *For every multiset rewriting system R and every equational theory E,*

$$\mathit{traces}_E(R) =_E \{\mathit{trace}(dg) \mid dg \in \mathit{dgraphs}_E(R)\}\ .$$

Proof. By induction over the sequence of multiset rewriting steps. □

8.2. Basic Backwards Reachability Analysis

Let R be a multiset rewriting system and E be an equational theory with a complete and finitary unification algorithm. In this section, we explain how we use constraint solving to perform a basic backwards reachability analysis for checking R, E-satisfiability of guarded trace properties. We explain this as follows.

We first define guarded trace properties. Then, we give the syntax and semantics of constraints and constraint systems. Finally, we give the set of constraint-reduction rules that we use for our basic backwards reachability analysis and illustrate their use on an example.

8.2.1. Guarded Trace Properties

In the remainder of this section, let f range over facts and i, j over temporal variables. A trace formula φ is in *negation normal form* if it is built such that negation is only applied to trace atoms and all its other logical connectives are $\wedge$, $\vee$, $\forall$, or $\exists$. Such a trace formula φ is a *guarded trace formula* if all its quantifiers are of the form $\exists \vec{x}.(f@i) \wedge \psi$ or $\forall \vec{x}.\neg(f@i) \vee \psi$ for an action $f@i$, a guarded trace formula ψ, and $x \in \mathit{vars}(f@i) \cap (\mathcal{V}_{msg} \cup \mathcal{V}_{temp})$ for every $x \in \vec{x}$.

Note that we restrict both universal and existential quantification and, as a result, the set of guarded trace properties is closed under negation. This, together with the support for quantifier alternations and the explicit comparison of timepoints, makes guarded trace properties well-suited for specifying security properties. The restriction of quantification to temporal and message variables only is a technical requirement, which simplifies later proofs. This restriction was no practical limitation in our case studies and it can be lifted if required.

A guarded trace formula φ is a *guarded trace property* if it is closed and $t \in \mathcal{V} \cup PN$ holds for all terms t occurring in φ. Note that all terms in a guarded trace property are either variables or public names. This is not a limitation in practice since the terms required to express a security property can be added to the actions of a protocol's rewriting rules. This restriction of guarded trace properties implies that all their quantified variables are instantiated with terms or indices that occur in the trace. We exploit this during constraint solving.

Note that in our case studies, it was possible to automatically convert the specified security properties, including the eCK model from Figure 7.4 and the stream authentication from Section 7.5.2, to guarded trace properties. The conversion first rewrites the given formula to negation normal form, pushes quantifiers inward, and reorders conjunctions and disjunctions to ensure that the guarding actions come first. Then, it replaces each body φ of a universal quantifier that is not a disjunction with $\neg\varphi \vee \bot$. The rewriting for existential quantifiers is analogous.

Example 22 (Guarded Trace Property)**.** Recall the validity claim

$$P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\forall}_{E_{PHS}} \varphi \qquad \text{for } \varphi := \forall x\, k\, i.\ \mathsf{Fin}(x,k)@i \Rightarrow \exists j.\ \mathsf{Rev}(k)@j \wedge j \lessdot i$$

from Example 19 on page 82, which formalizes that, for a run of the P_{Ex} protocol to finish, the corresponding short-term key must have been revealed. The trace formula characterizing all counter-examples to this validity claim is $\neg\varphi$. Using our conversion, we obtain the guarded trace property $\exists x\, k\, i.\ \mathsf{Fin}(x,k)@i \wedge \forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \neg(j \lessdot i)$. which is equivalent to $\exists x\, k\, i.\ \mathsf{Fin}(x,k)@i \wedge (\forall j.\ \mathsf{Rev}(k)@j \Rightarrow \neg(j \lessdot i))$. ♠

8.2.2. Syntax and Semantics of Constraint Systems

In the remainder of this section, let ri range over multiset rewriting rule instances, u and v over natural numbers, and φ over guarded trace formulas. A *graph constraint* is either a *node* $i : ri$, a *premise* $f \blacktriangleright_v i$, or an *edge* $(i,u) \rightarrowtail (j,v)$. A *constraint* is either a guarded trace formula or a graph constraint. We denote the set of all constraints by *Constraint*. A *constraint system* is a finite set of constraints.

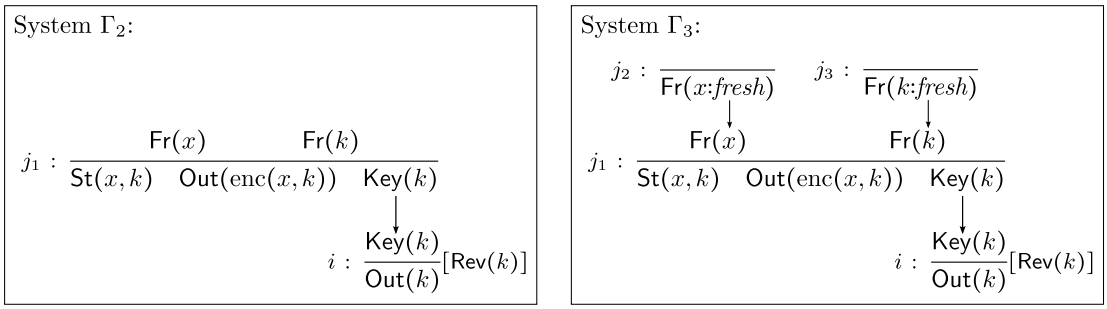


Figure 8.2.: Depiction of the constraint systems Γ_2 and Γ_3 from Example 23.

A *structure* is a tuple (dg, θ) of a dependency graph $dg = (I, D)$ and a valuation θ. We denote the application of the homomorphic extension of θ to a rule instance ri by $ri\theta$. We define when the structure (dg, θ) *satisfies a constraint* γ *modulo* E, written $(dg, \theta) \Vdash_E \gamma$, as follows.

$$
\begin{aligned}
&(dg,\theta) \Vdash_E i : ri && \text{iff } \theta(i) \in idx(I) \text{ and } ri\theta =_E I_{\theta(i)} \\
&(dg,\theta) \Vdash_E f \blacktriangleright_v i && \text{iff } \theta(i) \in idx(I) \text{ and } f\theta =_E prems(I_{\theta(i)})_v \\
&(dg,\theta) \Vdash_E (i,u) \nrightarrow (j,v) && \text{iff } (\theta(i),u) \nrightarrow (\theta(j),v) \in D \\
&(dg,\theta) \Vdash_E \varphi && \text{iff } (trace(dg),\theta) \vDash_E \varphi
\end{aligned}
$$

The structure (dg, θ) satisfies a constraint system Γ modulo E, written $(dg, \theta) \Vdash_E \Gamma$, if (dg, θ) satisfies each constraint in Γ modulo E.

Let R be a multiset rewriting system. We say that (dg, θ) is an R, E-*model* of Γ, if dg is an R, E-dependency graph and $(dg, \theta) \Vdash_E \Gamma$. An R, E-*solution* of Γ is an R, E-dependency graph dg such that there is a valuation θ with $(dg, \theta) \Vdash_E \Gamma$. Note that the free variables of a constraint system are therefore existentially quantified. We write $sols_{R,E}(\Gamma)$ to denote the set of all R, E-solutions of Γ.

8.2.3. Basic Constraint-Reduction Rules

We first give an example to provide intuition for our backwards reachability analysis. Then, we define the basic constraint-reduction rules. Together they give rise to the constraint-reduction relation $\rightsquigarrow^{\mathsf{basic}}_{R,E}$, which formalizes our basic backwards reachability analysis. Afterwards, we prove the correctness and completeness of $\rightsquigarrow^{\mathsf{basic}}_{R,E}$, define when a constraint system is solved, and show how to construct solutions from solved constraint systems. Finally, we explain how to use $\rightsquigarrow^{\mathsf{basic}}_{R,E}$ for formally proving R, E-validity statements and give an example of such a proof.

Example 23 (Intuition for Our Backwards Reachability Analysis)**.** Consider the protocol P_{Ex} from Example 18 and the satisfiability claim

$$P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\exists}_{E_{PHS}} \varphi \qquad \text{for } \varphi := \exists k\, i.\, \mathsf{Rev}(k)@i.$$

We can prove this claim by exhibiting a trace of P_{Ex} that satisfies φ. The main difficulty in such a proof is finding a satisfying trace. In our approach, we search for a satisfying

trace by searching for a $(P_{Ex} \cup MD_{\Sigma_{PHS}}), E_{PHS}$-dependency graph whose trace satisfies φ. To represent the state of this search, we use our constraint systems as follows.

We start by observing that the traces of all $(P_{Ex} \cup MD_{\Sigma_{PHS}}), E_{PHS}$-solutions of the constraint system $\{\varphi\}$ are exactly the traces of P_{Ex} that satisfy φ. This holds due to the definition of $\Vdash$ and Theorem 3. It is therefore sufficient to search for a $(P_{Ex} \cup MD_{\Sigma_{PHS}}), E_{PHS}$-solution of $\{\varphi\}$ to find a satisfying trace of P_{Ex}.

In this example, we search for such a solution by transforming the constraint system $\{\varphi\}$ in a step-by-step manner without changing its set of solutions until we arrive either at

(i) a trivially unsolvable constraint system or

(ii) a system that is solved in the sense that we can trivially construct a solution for it.

Because we ensure that the set of solutions is invariant under these transformations, Case (i) would entail that P_{Ex} does not satisfy φ, while Case (ii) provides us with a concrete trace witnessing that P_{Ex} satisfies φ. Note that, in general, we need case distinctions to make progress, and the state of our search is therefore a set of constraint systems. We show later how to generalize the above termination condition accordingly. In the following three steps, we transform $\{\varphi\}$ to a solved constraint system without changing its set of solutions.

First, note that the solutions of $\{\varphi\} = \{\exists k\, i.\, \mathsf{Rev}(k)@i\}$, are equal to the solutions of $\{\mathsf{Rev}(k)@i\}$, as the free variables of a constraint system are existentially quantified. As there is only one rule in $P_{Ex} \cup MD_{\Sigma_{PHS}} \cup \{\textsc{Fresh}\}$ whose instances have a Rev-action, the solutions of $\{\varphi\}$ are therefore equal to the solutions of

$$\Gamma_1 := \left\{ i : \frac{\mathsf{Key}(k)}{\mathsf{Out}(k)}[\mathsf{Rev}(k)] \right\} .$$

Second, in all solutions of the constraint system Γ_1, the Key-premise of node i must have an incoming edge from a Key-conclusion. As there is only one rule in $P_{Ex} \cup MD_{\Sigma_{PHS}} \cup \{\textsc{Fresh}\}$ whose instances have a Key-conclusion, the solutions of Γ_1 are therefore equal to the solutions of

$$\Gamma_2 := \left\{ i : \frac{\mathsf{Key}(k)}{\mathsf{Out}(k)}[\mathsf{Rev}(k)],\ j_1 : \frac{\mathsf{Fr}(x) \qquad \mathsf{Fr}(k)}{\mathsf{St}(x,k) \quad \mathsf{Out}(\mathrm{enc}(x,k)) \quad \mathsf{Key}(k)},\ (j_1,3) \twoheadrightarrow (i,1) \right\}.$$

As one can see, the textual representation of constraint systems quickly becomes unwieldy and we therefore use a graphical representation analogous to the one we use for dependency graphs. We depict the constraint system Γ_2 in Figure 8.2.

Third, note that system Γ_2 contains two Fr-premises without incoming edges. There is only one rule in $P_{Ex} \cup MD_{\Sigma_{PHS}} \cup \{\textsc{Fresh}\}$ that provides a corresponding Fr-conclusion, the Fresh rule. Hence, the solutions of Γ_2 are equal to the solutions of the constraint system Γ_3 depicted in Figure 8.2.

The system Γ_3 is the solved constraint system that we were looking for. We can construct a solution from it by topologically sorting its nodes and instantiating all variables with the corresponding trace indices or distinct fresh names. The corresponding trace $[\{\mathsf{Rev}(k)\}]$ witnesses that P_{Ex} satisfies φ. ♠

$\mathtt{S}_{@}$: $\Gamma \rightsquigarrow \|_{ri \in R} \|_{f' \in acts(ri)} (i : ri,\ f \approx f',\ \Gamma)$ if $\underline{(f@i)} \in \Gamma$; $\boxed{f@i \notin_E as(\Gamma)}$

$\mathtt{S}_{\approx}$: $\Gamma \rightsquigarrow \|_{\sigma \in unify_E^{vars(\Gamma)}(t_1,t_2)} (\Gamma\sigma)$ if $\underline{(t_1 \approx t_2)} \in \Gamma$; $\boxed{t_1 \neq_E t_2}$

$\mathtt{S}_{\doteq}$: $\Gamma \rightsquigarrow \Gamma\{i/j\}$ if $\underline{(i \doteq j)} \in \Gamma$; $\boxed{i \neq j}$

$\mathtt{S}_{\bot}$: $\Gamma \rightsquigarrow \bot$ if $\underline{\bot} \in \Gamma$

$\mathtt{S}_{\neg,@}$: $\Gamma \rightsquigarrow \bot$ if $\underline{\neg(f@i)} \in \Gamma$ and $(f@i) \in_E as(\Gamma)$

$\mathtt{S}_{\neg,\approx}$: $\Gamma \rightsquigarrow \bot$ if $\underline{\neg(t_1 \approx t_2)} \in \Gamma$ and $t_1 =_E t_2$

$\mathtt{S}_{\neg,\doteq}$: $\Gamma \rightsquigarrow \bot$ if $\underline{\neg(i \doteq i)} \in \Gamma$

$\mathtt{S}_{\neg,\lessdot}$: $\Gamma \rightsquigarrow (i \lessdot j, \Gamma) \parallel (i \doteq j, \Gamma)$ if $\underline{\neg(j \lessdot i)} \in \Gamma$; $\boxed{\text{neither } i \lessdot_\Gamma j \text{ nor } i = j}$

$\mathtt{S}_{\vee}$: $\Gamma \rightsquigarrow (\varphi_1, \Gamma) \parallel (\varphi_2, \Gamma)$ if $\underline{(\varphi_1 \vee \varphi_2)} \in \Gamma$; $\boxed{\varphi \notin_E \Gamma \text{ and } \varphi_2 \notin_E \Gamma}$

$\mathtt{S}_{\wedge}$: $\Gamma \rightsquigarrow (\varphi_1, \varphi_2, \Gamma)$ if $\underline{(\varphi_1 \wedge \varphi_2)} \in \Gamma$; $\boxed{\text{not } \{\varphi_1, \varphi_2\} \subseteq_E \Gamma}$

$\mathtt{S}_{\exists}$: $\Gamma \rightsquigarrow (\varphi\{y/x\}, \Gamma)$
if $\underline{(\exists x{:}s.\ \varphi)} \in \Gamma$ and $y{:}s$ fresh; $\boxed{\varphi\{w/x\} \notin_E \Gamma \text{ for every term } w \text{ of sort } s}$

$\mathtt{S}_{\forall}$: $\Gamma \rightsquigarrow (\psi\sigma, \Gamma)$ if $(\forall \vec{x}.\ \neg(f@i) \vee \psi) \in \Gamma$ and $dom(\sigma) = set(\vec{x})$ and $(f@i)\sigma \in_E as(\Gamma)$; $\boxed{\psi\sigma \notin_E \Gamma}$

$\mathtt{DG}_{\mathtt{lbl}}$: $\Gamma \rightsquigarrow (ri \approx ri', \Gamma)$ if $\{i : ri,\ i : ri'\} \subseteq \Gamma$; $\boxed{ri \neq_E ri'}$

$\mathtt{DG}_{\lessdot}$: $\Gamma \rightsquigarrow \bot$ if $i \lessdot_\Gamma i$

$\mathtt{DG}_{\rightarrowtail}$: $\Gamma \rightsquigarrow (f \approx f', \Gamma)$ if $c \rightarrowtail p \in \Gamma$ and $(c, f) \in cs(\Gamma)$ and $(p, f') \in ps(\Gamma)$; $\boxed{f \neq_E f'}$

$\mathtt{DG}_{\blacktriangleright}$: $\Gamma \rightsquigarrow (f \blacktriangleright_v i, \Gamma)$ if $((i, v), f) \in ps(\Gamma)$; $\boxed{(f \blacktriangleright_v i) \notin_E \Gamma}$

$\mathtt{S}_{\blacktriangleright}$: $\Gamma \rightsquigarrow \|_{ri \in R \cup \{\text{FRESH}\}} \|_{u \in idx(concs(ri))} (i : ri,\ (i, u) \rightarrowtail (j, v),\ \Gamma)$
if $\underline{(f \blacktriangleright_v j)} \in \Gamma$ and i fresh; $\boxed{\text{there is no } c \text{ such that } (c \rightarrowtail (j, v)) \in \Gamma}$

$\mathtt{DG}_{\mathtt{in}}$: $\Gamma \rightsquigarrow (\text{if } u = v \text{ then } (i \doteq j, \Gamma) \text{ else } \bot)$
if $\{(i, v) \rightarrowtail p,\ (j, u) \rightarrowtail p\} \subseteq \Gamma$; $\boxed{(i, v) \neq (j, u)}$

$\mathtt{DG}_{\mathtt{out}}$: $\Gamma \rightsquigarrow (\text{if } u = v \text{ then } (i \doteq j, \Gamma) \text{ else } \bot)$
if $\{c \rightarrowtail (i, v), c \rightarrowtail (j, u)\} \subseteq \Gamma$ and c linear in Γ; $\boxed{(i, v) \neq (j, u)}$

$\mathtt{DG}_{\mathtt{Fr}}$: $\Gamma \rightsquigarrow (i \doteq j, \Gamma)$ if $\{i : \dashv[]\rightarrow \mathsf{Fr}(m),\ j : \dashv[]\rightarrow \mathsf{Fr}(m)\} \subseteq_E \Gamma$; $\boxed{i \neq j}$

We assume that the multiset rewriting rules in R are renamed apart from Γ. We write $\rightsquigarrow^{\mathtt{basic}}_{R,E}$ as $\rightsquigarrow$ and the empty set of constraint systems $\varnothing$ as $\bot$. We write $\Gamma \rightsquigarrow \Gamma_1 \parallel \ldots \parallel \Gamma_n$ for $\Gamma \rightsquigarrow \{\Gamma_1, \ldots, \Gamma_n\}$, which denotes an n-fold case distinction. Note that $\Gamma \rightsquigarrow \Gamma'$ abbreviates $\Gamma \rightsquigarrow \{\Gamma'\}$.

Figure 8.3.: Rules defining the constraint-reduction relation $\rightsquigarrow^{\mathtt{basic}}_{R,E}$

We formalize the reasoning that we used in the above example by the constraint-reduction relation $\rightsquigarrow^{\text{basic}}_{R,E}$, defined shortly. Its definition relies on the following additional conventions and definitions. We extend the equality $\approx$ over terms to facts and rule instances by interpreting the constructors for facts and rule instances as free function symbols. For a constraint system Γ, its *actions* $as(\Gamma)$, *premises* $ps(\Gamma)$, and *conclusions* $cs(\Gamma)$ of Γ are

$$
\begin{aligned}
as(\Gamma) &:= \{f@i \mid \exists r\, a.\ (i : l\text{–}[\ a\ ]\!\!\rightarrow r) \in \Gamma \wedge f \in a\},\\
ps(\Gamma) &:= \{((i,u), l_u) \mid \exists r\, a.\ (i : l\text{–}[\ a\ ]\!\!\rightarrow r) \in \Gamma \wedge u \in idx(l)\}, \text{ and}\\
cs(\Gamma) &:= \{((i,v), r_v) \mid \exists l\, a.\ (i : l\text{–}[\ a\ ]\!\!\rightarrow r) \in \Gamma \wedge v \in idx(r)\}.
\end{aligned}
$$

The *temporal order of* Γ is $(\lessdot_\Gamma) := \{(i,j) \mid (i \lessdot j) \in \Gamma \vee \exists u\, v.\ (i,u) \twoheadrightarrow (j,v) \in \Gamma\}^+$.

A *constraint-reduction relation* $\rightsquigarrow$ is a relation between constraint systems and sets of constraint systems. We use sets of constraint systems to represent case distinctions. We say that $\rightsquigarrow$ is R,E*-correct* if $sols_{R,E}(\Gamma) \supseteq \bigcup_{\Gamma' \in \mathbf{\Gamma'}} sols_{R,E}(\Gamma')$ for every $\Gamma \rightsquigarrow \mathbf{\Gamma'}$. We say that $\rightsquigarrow$ is R,E*-complete* if $sols_{R,E}(\Gamma) \subseteq \bigcup_{\Gamma' \in \mathbf{\Gamma'}} sols_{R,E}(\Gamma')$ for every $\Gamma \rightsquigarrow \mathbf{\Gamma'}$. Intuitively, correctness ensures that $\rightsquigarrow$ does not introduce false solutions, while completeness ensures that $\rightsquigarrow$ does not remove any solutions. Note that we use a bold greek letters, like $\mathbf{\Gamma}$, to denote sets of constraint systems.

The *basic constraint-reduction relation* $\rightsquigarrow^{\text{basic}}_{R,E}$ is defined by the constraint-reduction rules given in Figure 8.3. Our naming scheme embodies that an $\mathtt{S}_\gamma$-rule solves the constraint γ, while a $\mathtt{DG}_x$-rule ensures the consistency of the constraint systems with one of the properties **DG1-4** of dependency graphs. In the following, we first explain the $\mathtt{S}_@$ rule together with notational details. We then explain the rest of the rules for solving guarded trace formulas. Afterwards, we explain the rules for ensuring consistency with the properties of dependency graphs.

The rule $\mathtt{S}_@$ solves an action constraint $f@i$. It performs a case distinction over all actions f' of every multiset rewriting rule ri in R and introduces in each case the constraints $i : ri$ and $f \approx f'$, which formalize that the rule instance ri might give rise to the action $f@i$. For our search to be complete, we require case distinctions to be finite. This is the case for the rule $\mathtt{S}_@$, as a multiset rewriting system is a finite set of multiset rewriting rules. The framed part of the rule's side-condition excludes applications to solved action constraints. The underlined constraint in the rule's side-condition marks the constraint that is solved by this rule.

In general, the unframed part of a rule's side-condition ensures that the rule is complete. The framed part additionally excludes applying a rule to a constraint system from which we can extract a solution. This is crucial for our later definition of a solved constraint system as a constraint system to which no further constraint-reduction rule applies.

The rest of the rules for solving guarded trace formulas work as follows. The rule $\mathtt{S}_\approx$ solves an equality between two terms by performing a case distinction over all possible E-unifiers. This case distinction is finite, as we restrict ourselves to equational theories with finitary unification. The rule $\mathtt{S}_\doteq$ solves an equality between two temporal variables by substituting all occurrences of one by the other. The rules $\mathtt{S}_\bot$, $\mathtt{S}_{\neg,@}$, $\mathtt{S}_{\neg,\approx}$, $\mathtt{S}_{\neg,\doteq}$ prune constraint systems that contain $\bot$ or contradict a negated action constraint or a negated equality. The rule $\mathtt{S}_{\neg,\lessdot}$ performs a case distinction to solve negated timepoint ordering constraints. The rule $\mathtt{S}_\vee$ performs a case distinction to solve a disjunction. The rule $\mathtt{S}_\wedge$

adds the conjuncts of a conjunction to the constraint system. The rule $\mathtt{S}_\exists$ ensures that, for every existential quantification $\exists x{:}s.\varphi$, there is at least one term w of sort s such that $\varphi\{w/x\}$ holds. The rule $\mathtt{S}_\forall$ saturates the constraint system with all consequences implied by the actions of the constraint system and its universally quantified guarded trace formulas. It uses matching modulo E to find candidate substitutions. It checks whether $\psi\sigma \notin_E \Gamma$ modulo renaming of bound variables to avoid introducing the consequence multiple times. This check suffices to avoid redundant applications, as we never remove a guarded trace formula from the constraint system.

The rules for ensuring consistency with the properties of dependency graphs work as follows. The rule $\mathtt{DG}_{\mathtt{lbl}}$ ensures that nodes are uniquely labeled. It relies on the rule $\mathtt{S}_\approx$ for solving the introduced equality. The rule $\mathtt{DG}_\lessdot$ prunes constraint systems where $\lessdot_\Gamma$ is not a strict partial order. The rule $\mathtt{DG}_{\nrightarrow}$ ensures that edges connect equal facts. Like rule $\mathtt{DG}_{\mathtt{lbl}}$, it relies on the rule $\mathtt{S}_\approx$ for solving the introduced equality. The rule $\mathtt{DG}_\blacktriangleright$ generates premise constraints for the premises of all nodes. The rule $\mathtt{S}_\blacktriangleright$ then solves these constraints and relies on the rule $\mathtt{DG}_{\nrightarrow}$ to enforce the equality between conclusions and premises. Together, the rules $\mathtt{DG}_\blacktriangleright$ and $\mathtt{S}_\blacktriangleright$ therefore ensure that every premise of every node has at least one incoming edge. Note that we split the generation and solving of premise constraints to simplify the later definition of special constraint-reduction rules for solving premise constraints stemming from message deduction rules. The rule $\mathtt{DG}_{\mathtt{in}}$ ensures that every premise has at most one incoming edge and the rule $\mathtt{DG}_{\mathtt{out}}$ ensures that each linear conclusion has at most one outgoing edge. The rule $\mathtt{DG}_{\mathtt{Fr}}$ ensures that instances of the FRESH multiset rewriting rule are unique.

Formal Properties of $\rightsquigarrow^{\mathtt{basic}}_{R,E}$

We first prove that $\rightsquigarrow^{\mathtt{basic}}_{R,E}$ is correct and complete. Then, we define the notions of wellformed constraint systems and wellformed constraint-reduction relations and show that $\rightsquigarrow^{\mathtt{basic}}_{R,E}$ is a wellformed constraint-reduction relation. Finally, we define the notion of solved constraint systems and show that we can extract a solution from every solved constraint system.

Theorem 4. *The constraint-reduction relation $\rightsquigarrow^{\mathtt{basic}}_{R,E}$ is correct and complete.*

Proof (sketch of the complete proof on page 177). Note that all rules are trivially correct, as they only add constraints or apply a substitution to the whole constraint system. Completeness follows from the definition of $sols_{R,E}(\cdot)$, $\Vdash_E$, $\vDash_E$, and $\rightsquigarrow^{\mathtt{basic}}_{R,E}$. □

Let Γ be a constraint system. We say that Γ is *R,E-wellformed* if it satisfies the wellformedness conditions **WF1-4**.

WF1 For every node $(i : ri) \in \Gamma$, it holds that $ri \in_E insts(R \cup \{\text{FRESH}\})$.

WF2 For every premise requirement $(f \blacktriangleright_v i) \in \Gamma$, it holds that $((i, v), f) \in_E ps(\Gamma)$.

WF3 For every edge $(c \twoheadrightarrow p) \in \Gamma$, it holds that $c \in dom(cs(\Gamma))$ and $p \in dom(ps(\Gamma))$.

WF4 For every trace formula $\varphi \in \Gamma$ and every universal quantifier $\forall \vec{x}.\,\omega$ in φ, it holds that $\omega = (\neg f(\vec{t})@i \vee \psi)$ for some fact symbol f, some temporal variable i, some sequence of terms $\vec{t}$ with $set(\vec{x}) \subseteq set(\vec{t} \cdot [i])$, and some trace formula ψ.

A constraint-reduction relation is R, E-*wellformed* if it relates R, E-wellformed constraint systems to R, E-wellformed constraint systems only. A constraint system Γ is $\rightsquigarrow^{\text{basic}}_{R,E}$-*solved* if it is R, E-wellformed and $\rightsquigarrow^{\text{basic}}_{R,E}$-irreducible, i.e., $\Gamma \notin dom(\rightsquigarrow^{\text{basic}}_{R,E})$.

Intuitively, condition **WF1** ensures that nodes are labeled with instances of $R \cup \{\text{FRESH}\}$. Conditions **WF2** and **WF3** ensure that premise and edge constraints refer to nodes with corresponding labels. Condition **WF4** generalizes guardedness to trace formulas. We exploit these conditions below in the construction of a solution for a solved constraint system.

Lemma 1. *The constraint-reduction relation $\rightsquigarrow^{\text{basic}}_{R,E}$ is wellformed.*

Proof. The rules introducing node, premise, edge constraints, and trace formulas maintain properties **WF1-4** and these properties are invariant under applying a substitution to the whole constraint system. □

Theorem 5 (Solution Construction)**.** *We can construct an R, E-model for every $\rightsquigarrow^{\text{basic}}_{R,E}$-solved constraint system Γ.*

Proof (sketch of the complete proof on page 180). We construct an R, E-model of Γ as follows. We first use a topological sorting to construct some sequence $[i_1, \ldots, i_n]$ of all temporal variables in Γ ordered consistently with $\lessdot_\Gamma$. Such a sequence exists because rule $\texttt{DG}_{\lessdot}$ is not applicable, as Γ is solved.

Then, we assign a rule instance ri_k to every temporal variable i_k with $1 \leq k \leq n$ as follows. If there exists some constraint $(i_k : ri) \in \Gamma$, then we use ri as the rule instance ri_k of i_k. This choice of ri_k is unique modulo E, as the constraint-reduction rule $\texttt{DG}_{\texttt{1bl}}$ is not applicable. If there exists no such constraint, then we use some instance of the FRESH rule not occurring in Γ as the rule instance of i_k.

We choose θ to be a valuation that is injective, maps variables in $\mathcal{V}_{pub} \cup \mathcal{V}_{msg}$ to public names and variables in $\mathcal{V}_{fresh}$ to fresh names, has a range disjoint from $St(\Gamma) \cup St([ri_1, \ldots, ri_k])$, and satisfies $\theta(i_k) = k$ for $1 \leq k \leq n$. Such a θ exists because there are infinitely many public and fresh names.

Finally, we define

$$dg := (D, I) := ([ri_1\theta, \ldots, ri_n\theta], \{(c\theta, p\theta) \mid c \twoheadrightarrow p \in \Gamma\})$$

and prove that (dg, θ) is an R, E-model of Γ.

Most cases of this proof follow straightforwardly from the fact that Γ is $\rightsquigarrow^{\text{basic}}_{R,E}$-irreducible, i.e., that none of the basic constraint-reduction rules from Figure 8.3 is applicable to Γ. The proof that dg satisfies **DG1-4** and the premise and edge constraints additionally exploits the wellformedness conditions **WF1-3**. The proof that the graph constraints in Γ are satisfied follows from our construction of (dg, θ) and condition **WF3**. The proof that $(trace(dg), \theta)$ satisfies all guarded trace formulas in Γ uses induction over the size of the guarded trace formulas in Γ. Its only non-trivial case concerns formulas of the form $\forall \vec{x}. \neg(f@i) \vee \psi$. In this case we exploit the condition **WF4**, the injectivity of θ, and the fact that the range of θ is disjoint from the subterms of Γ. Together these facts ensure that, from every ground substitution ρ that makes $(f@i)\rho$ true in $(trace(dg), \theta)$, we can reconstruct a substitution σ of the variables $\vec{x}$ with terms of Γ such that $(f@i)\sigma \in_E as(\Gamma)$. The saturation under the rule $\texttt{S}_\forall$ thus guarantees that the constructed model $(trace(dg), \theta)$ satisfies all formulas of the form $\forall \vec{x}. \neg(f@i) \vee \psi$. □

Proving R, E-validity and R, E-satisfiability Claims

Recall that a trace formula φ is R, E-valid iff $\neg\varphi$ is not R, E-satisfiable. We can therefore use the following lemma to translate both R, E-satisfiability and R, E-validity claims to logically equivalent statements about the existence of solutions of corresponding constraint systems.

Lemma 2. *A guarded trace formula φ is R, E-satisfiable iff the constraint system $\{\varphi\}$ has an R, E-solution.*

Proof. Follows from Theorem 3 and the definitions of R, E-satisfiability and $\Vdash_E$. □

In general, we use correct, complete, and wellformed constraint-reduction relations like $\rightsquigarrow^{\mathsf{basic}}_{R,E}$ to reason about the solutions of the constraint systems resulting from an application of Lemma 2. Such constraint-reduction relations can thus be seen as calculi suitable for constructing formal satisfiability and validity proofs. We illustrate this in the following example.

Example 24 (Formal Validity Proof)**.** We prove that $P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\forall}_{E_{PHS}} \varphi$ for

$$\varphi := \forall x_1\, x_2\, k\, i_1\, i_2.\ \mathsf{Fin}(x_1, k)@i_1 \wedge \mathsf{Fin}(x_2, k)@i_2 \Rightarrow (i_1 \doteq i_2) \wedge (x_1 \approx x_2)\ .$$

First, we use Lemma 2 to justify that $P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\forall}_{E_{PHS}} \varphi$ holds iff the constraint system $\{\hat{\varphi}\}$ has no solutions for

$$\hat{\varphi} := \exists x_1\, k\, i_1.\ \mathsf{Fin}(x_1, k)@i_1 \wedge (\exists x_2\, i_2.\ \mathsf{Fin}(x_2, k)@i_2 \wedge (\neg(i_1 \doteq i_2) \vee \neg(x_1 \approx x_2)))\ .$$

Note that $\hat{\varphi}$ is a guarded trace property, which we obtained by rewriting $\neg\varphi$ to negation normal form and pushing the quantifiers inwards as far as possible. We show that $sols(\{\hat{\varphi}\}) = \varnothing$ using our basic constraint-reduction rules.

We start by applying the rules $\mathsf{S}_\exists$, $\mathsf{S}_\wedge$, $\mathsf{S}_\exists$, and then $\mathsf{S}_\wedge$ to $\{\hat{\varphi}\}$. The resulting constraint system is

$$\begin{aligned}\Gamma := \{\ & \exists x_1\, k\, i_1.\ \mathsf{Fin}(x_1, k)@i_1 \wedge (\exists x_2\, i_2.\ \mathsf{Fin}(x_2, k)@i_2 \wedge (\neg(i_1 \doteq i_2) \vee \neg(x_1 \approx x_2))) \\ & , \mathsf{Fin}(x_1, k)@i_1 \wedge (\exists x_2\, i_2.\ \mathsf{Fin}(x_2, k)@i_2 \wedge (\neg(i_1 \doteq i_2) \vee \neg(x_1 \approx x_2))) \\ & , \mathsf{Fin}(x_1, k)@i_1\ , (\exists x_2\, i_2.\ \mathsf{Fin}(x_2, k)@i_2 \wedge (\neg(i_1 \doteq i_2) \vee \neg(x_1 \approx x_2))) \\ & , \mathsf{Fin}(x_2, k)@i_2 \wedge (\neg(i_1 \doteq i_2) \vee \neg(x_1 \approx x_2)) \\ & , \mathsf{Fin}(x_2, k)@i_2\ ,\ \neg(i_1 \doteq i_2) \vee \neg(x_1 \approx x_2)\ \}\ .\end{aligned}$$

Here, we can observe how our basic constraint-reduction rules iteratively decompose trace formulas. Note that all constraints in Γ except the ones with a light gray background are solved in the sense that none of the S_γ-rules applies to them. We delay the case distinction on $\neg(i_1 \doteq i_2) \vee \neg(x_1 \approx x_2)$ to avoid duplicating the work of solving the two Fin-action constraints.

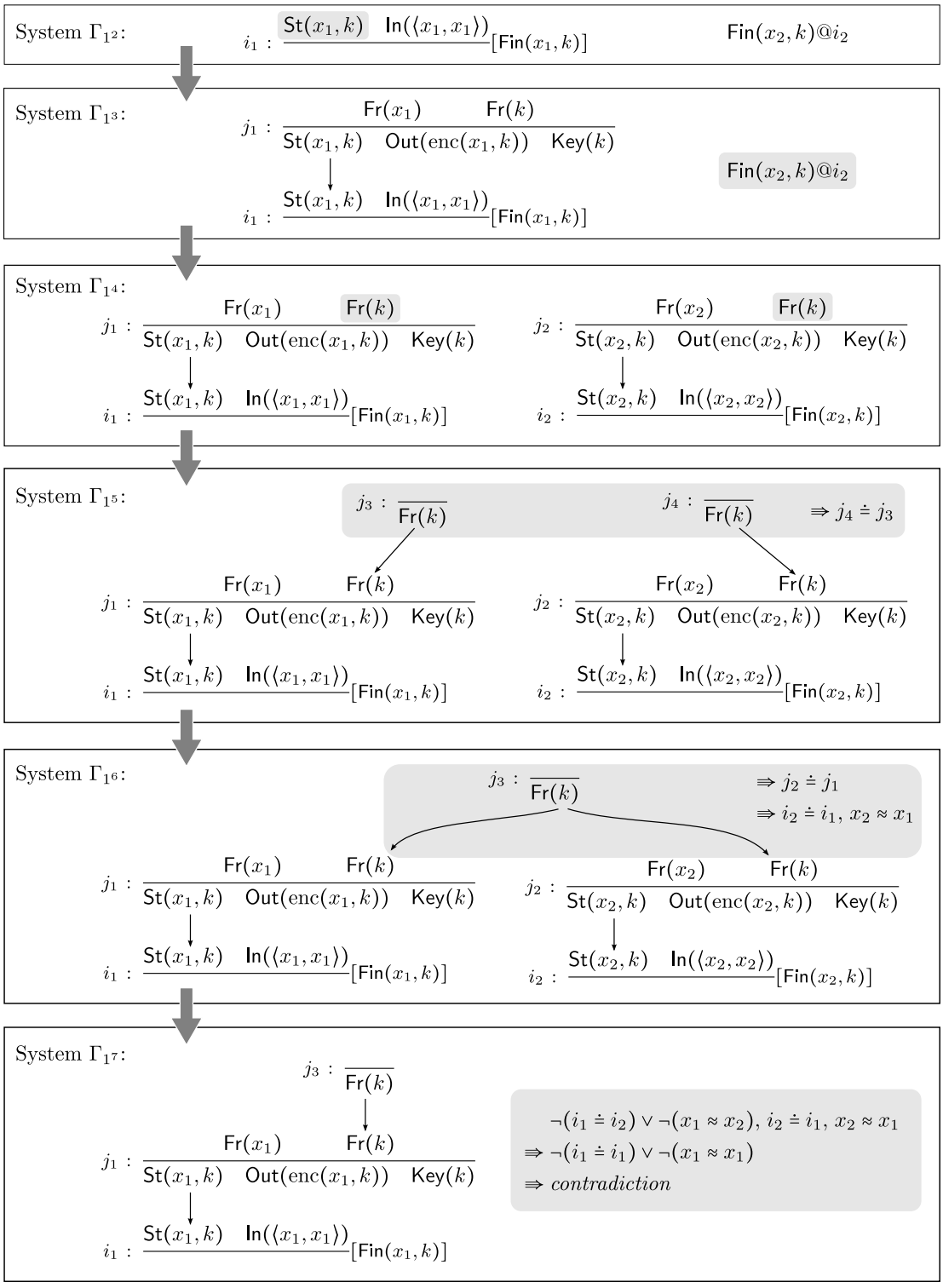


Figure 8.4.: Depiction of the constraint systems and constraint-reduction steps from Example 24.

We continue by solving the $\mathsf{Fin}(x_1, k)@i_1$ constraint using rule $\mathtt{S}_@$. The two resulting constraint systems are

$$\Gamma_1 := \Gamma \cup \left\{ i_1 : \frac{\mathsf{St}(x', k') \quad \mathsf{In}(\langle x', x' \rangle)}{[\mathsf{Fin}(x', k')],\ \mathsf{Fin}(x_1, k) \approx \mathsf{Fin}(x', k')} \right\} \text{ and}$$

$$\Gamma_2 := \Gamma \cup \left\{ i_1 : \frac{\mathsf{Key}(k')}{\mathsf{Out}(k')}[\mathsf{Rev}(k')],\ \mathsf{Fin}(x_1, k) \approx \mathsf{Rev}(k') \right\} .$$

As our constraint-reduction rules are correct and complete, it holds that

$$sols(\{\hat{\varphi}\}) = sols(\Gamma) = sols(\Gamma_1) \cup sols(\Gamma_2) .$$

Moreover, applying $\mathtt{S}_\approx$ to $\mathsf{Fin}(x_1, k) \approx \mathsf{Rev}(k')$ reduces Γ_2 to the empty set of constraint systems, which implies that $sols(\Gamma_2) = \varnothing$. Thus, it remains to show that $sols(\Gamma_1) = \varnothing$.

We proceed by solving the $\mathsf{Fin}(x_1, k) \approx \mathsf{Fin}(x', k')$ constraint in Γ_1 using the rule $\mathtt{S}_\approx$. The resulting constraint system is

$$\Gamma_{1,1} := \Gamma_1 \cup \left\{ i_1 : \frac{\mathsf{St}(x_1, k) \quad \mathsf{In}(\langle x_1, x_1 \rangle)}{[\mathsf{Fin}(x_1, k)],\ \mathsf{Fin}(x_1, k) \approx \mathsf{Fin}(x_1, k)} \right\} .$$

At this point, the syntactic manipulations performed by our constraint-reduction rules should be clear. To improve the readability of the remainder of this example, we switch from formally defining the occurring constraint systems to depicting them graphically. Furthermore, we abbreviate the sequences of case numbers in the indices of the constraint systems using exponentiation. For example, we write $\Gamma_{1,1}$ as Γ_{1^2} and $\Gamma_{1,1,2,1,1,1}$ as $\Gamma_{1^2,2,1^3}$.

The constraint system Γ_{1^2} is depicted in Figure 8.4 together with the constraint systems occurring in the remainder of this example. We explain these systems after we explain the notation used in this figure. The set of large gray arrows leaving a constraint system Γ denotes the reduction of this system Γ to a set of constraint systems $\mathbf{\Gamma'}$ with equal solutions, i.e., $sols(\Gamma) = \bigcup_{\Gamma' \in \mathbf{\Gamma'}} sols(\Gamma')$. This is a bit difficult to see in this example, as all reductions except the last one result in a single constraint system. The last reduction reduces System Γ_{1^7} to the empty set of constraint systems, which implies $sols(\Gamma_{1^7}) = \varnothing$. A constraint-reduction step may be the result of multiple applications of constraint-reduction rules. We use a light-gray background to mark the constraints leading to the next constraint-reduction step. We do not depict the guarded trace formulas that are just copied from the previous system.

We reduce System Γ_{1^2} to System Γ_{1^3} by solving the premise $\mathsf{St}(x_1, k) \blacktriangleright_1 i_1$. Formally, we first apply the rule $\mathtt{DG}_\blacktriangleright$ to introduce the $\mathsf{St}(x_1, k) \blacktriangleright_1 i_1$ constraint, which we then solve using the rule $\mathtt{S}_\blacktriangleright$. This results in one case per conclusion of a rule in $P_{Ex} \cup MD \cup \{\text{FRESH}\}$. After solving their equality constraints using the rule $\mathtt{S}_\approx$ only System Γ_{1^3} remains.

Note that this reduction of System Γ_{1^2} to System Γ_{1^3} is just one among a number possible reductions. For example, we could also apply rule $\mathtt{S}_@$ to the $\mathsf{Fin}(x_2, k)@i_2$ constraint in System Γ_{1^2}. Another option would be to apply the rule $\mathtt{S}_\vee$ to $\neg(i_1 \doteq i_2) \vee \neg(x_1 \approx x_2)$. It is up to us to choose one of these constraint-reduction rules, and use it to reduce System Γ_{1^2} to a resulting set of constraint systems $\mathbf{\Gamma}$. The correctness

and completeness of our constraint-reduction rules guarantees for every choice that the set of solutions remains unchanged. In general, we choose constraint-reduction steps that result in short proofs. In our constraint solver, we use the heuristic described in Section 9.2.3 to choose constraint-reduction steps.

We continue the proof in this example by solving the $\mathsf{Fin}(x_2, k)@i_2$ constraint in Γ_{1^3}. The corresponding constraint-reduction steps are analogous to the ones reducing System Γ_1 to System Γ_{1^2}. The resulting System is Γ_{1^4}. Intuitively, this constraint system has no solutions because nodes i_1 and i_2 require the same $\mathsf{Fr}(k)$-premise, which implies indirectly that $i_1 \doteq i_2$ and $x_1 \approx x_2$ and therefore contradicts the constraint $\neg(i_1 \doteq i_2) \vee \neg(x_1 \approx x_2)$. We formalize this as follows.

We solve both $\mathsf{Fr}(k)$-premises in System Γ_{1^4} using the rule $\mathtt{DG}_{\blacktriangleright}$ to introduce the corresponding premise constraints, followed by the rules $\mathtt{S}_{\blacktriangleright}$ and $\mathtt{S}_{\approx}$ to solve them. This results in System Γ_{1^5}, which we reduce further to System Γ_{1^6} using the rule $\mathtt{DG}_{\mathtt{Fr}}$, which enforces the uniqueness of instances of the Fresh rule. We reduce System Γ_{1^6} to System Γ_{1^7} using the rule $\mathtt{DG}_{\mathtt{out}}$, which merges multiple outgoing edges of linear facts, and the rules $\mathtt{DG}_{\mathtt{lbl}}$, $\mathtt{S}_{\doteq}$, and $\mathtt{S}_{\approx}$. Note that the substitutions resulting from solving the introduced equalities are applied to the whole constraint system. The constraint $\neg(i_1 \doteq i_2) \vee \neg(x_1 \approx x_2)$ thus becomes $\neg(i_1 \doteq i_1) \vee \neg(x_1 \approx x_1)$, which is obviously false. We formally derive this by applying rule $\mathtt{S}_{\vee}$ to $\neg(i_1 \doteq i_1) \vee \neg(x_1 \approx x_1)$ and reducing the resulting constraint systems to the empty set of constraint systems using the rules $\mathtt{S}_{\neg,\doteq}$ and $\mathtt{S}_{\neg,\approx}$. This implies that $sols(\Gamma_{1^7}) = \varnothing$, which concludes our proof that $P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\forall}_{E_{PHS}} \varphi$. ♠

The previous example provides intuition on the interplay between the different basic constraint-reduction rules. It also highlights that we can view a constraint-reduction relation like $\rightsquigarrow^{\mathtt{basic}}_{R,E}$ as a calculus for proving R, E-validity and R, E-satisfiability claims. Constraint solving then corresponds to proof search in this calculus.

8.3. Trace Induction

We first motivate trace induction on a simple example that cannot be verified using our basic constraint-reduction rules. Afterwards, we prove the key theorem formalizing trace induction. This theorem makes use of an additional trace atom, which denotes the last timepoint of a trace. We therefore introduce the constraint-reduction relation $\rightsquigarrow^{\mathtt{last}}_{R,E}$, which extends $\rightsquigarrow^{\mathtt{basic}}_{R,E}$ with support for reasoning about this new trace atom. Finally, we show that our motivating example can be verified using trace induction.

8.3.1. Motivation

Example 25 (Motivation for Trace Induction)**.** Consider the multiset rewriting system

$$R_{loop} := \left\{ \frac{\mathsf{Fr}(x)}{\mathsf{A}(x)}[\mathsf{Start}(x)],\ \frac{\mathsf{A}(x)}{\mathsf{A}(x)}[\mathsf{Loop}(x)] \right\} .$$

We would like to use our constraint-reduction rules to prove that

$$R_{loop} \vDash^{\forall}_{E_{PHS}} (\forall x\ i.\ \mathsf{Loop}(x)@i \Rightarrow \exists j.\ \mathsf{Start}(x)@j) .$$

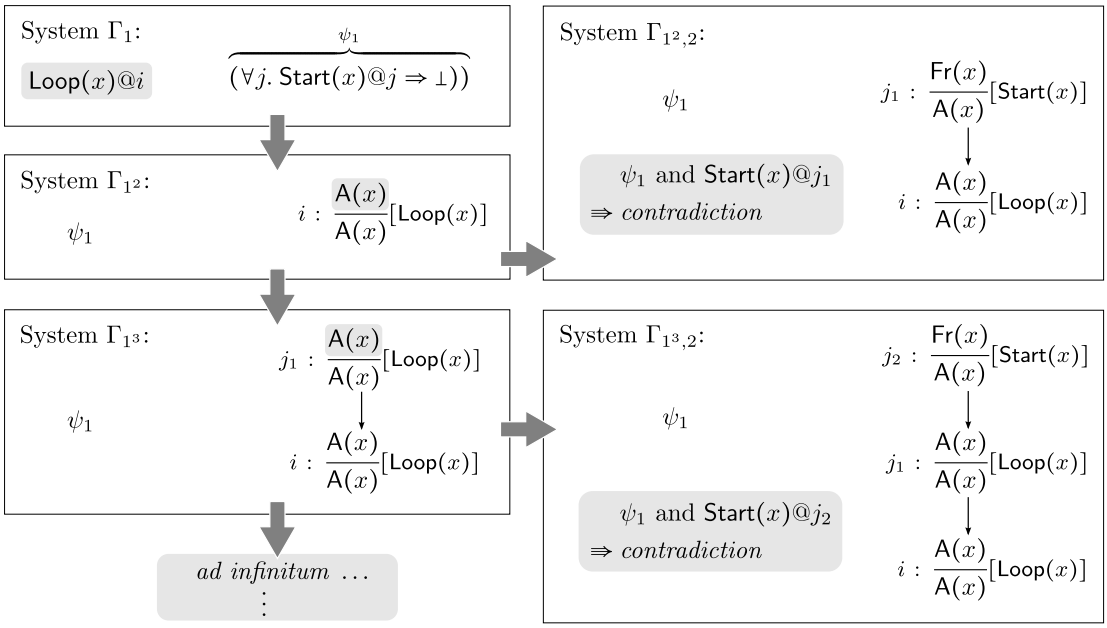


Figure 8.5.: Depiction of the constraint systems and constraint-reduction steps from Example 25, which illustrates the need for trace induction.

However, showing that the corresponding constraint system

$$\Gamma := \{\exists x\ i.\ \mathsf{Loop}(x)@i \wedge (\forall j.\ \mathsf{Start}(x)@j \Rightarrow \bot)\}$$

has no solutions is out of scope for our basic constraint-reduction rules. The problem is depicted in Figure 8.5. After decomposing the guarded trace formula in Γ, we are left with System Γ_1, which reduces to System Γ_{1^2} after solving the $\mathsf{Loop}(x)@i$ constraint. System Γ_{1^2} contains one open constraint, the premise constraint $\mathsf{A}(x) \blacktriangleright_1 i$. Solving this constraint results in two constraint systems. The System $\Gamma_{1^2,2}$ is contradictory, whereas the System Γ_{1^3} again contains an open $\mathsf{A}(x)$-premise constraint. We thus fail to make progress using our basic constraint-reduction rules, which demonstrates their insufficiency for proving that Γ has no solutions.

Informally, we could show that the System Γ has no solutions, if we attempted a proof by induction over the traces of R_{loop}. The induction hypothesis would then state that

$$\forall x\ i.\ \mathsf{Loop}(x)@i \Rightarrow \exists j.\ \mathsf{Start}(x)@j$$

holds for every node except the last one. Node j_1 in System Γ_{1^3} has an outgoing edge and is therefore not the last node. We could thus use our basic constraint-reduction rules to instantiate the induction hypothesis for $\mathsf{Loop}(x)@j_1$, introduce a $\mathsf{Start}(x)@j$ constraint, and derive a contradiction with $\forall j.\ (\mathsf{Start}(x)@j) \Rightarrow \bot$ in System Γ_{1^3}. ♠

8.3.2. Formalization

We formalize induction over the traces of a multiset rewriting system R as a transformation of trace formulas. In the following, we give definitions for transformations $\textsc{bc}(\varphi)$

$$\text{BC}(\varphi) := \begin{cases} \bot & \text{if } \varphi = f@i \\ \neg\text{BC}(\varphi_1) & \text{if } \varphi = \neg\varphi_1 \\ \text{BC}(\varphi_1) \wedge \text{BC}(\varphi_2) & \text{if } \varphi = \varphi_1 \wedge \varphi_2 \\ \exists x.\ \text{BC}(\varphi_1) & \text{if } \varphi = \exists x.\ \varphi_1 \\ \varphi & \text{if } \varphi \text{ is a trace atom other than } f@i \end{cases}$$

$$\text{IH}(\varphi) := \begin{cases} \neg\text{IH}(\varphi_1) & \text{if } \varphi = \neg\varphi_1 \\ \text{IH}(\varphi_1) \wedge \text{IH}(\varphi_2) & \text{if } \varphi = \varphi_1 \wedge \varphi_2 \\ \exists i.\ \text{IH}(\varphi_1) \wedge \neg\text{last}(i) & \text{if } \varphi = \exists i{:}temp.\ \varphi_1 \\ \exists x.\ \text{IH}(\varphi_1) & \text{if } \varphi = \exists x{:}s.\ \varphi_1 \text{ and } s \text{ subsort of } msg \\ \varphi & \text{if } \varphi \text{ is a trace atom} \end{cases}$$

Figure 8.6.: Definition of the transformations $\text{BC}(\varphi)$ and $\text{IH}(\varphi)$

and $\text{IH}(\varphi)$ such that

$$R \vDash^{\forall}_E \varphi \quad \text{iff} \quad R \vDash^{\forall}_E \text{BC}(\varphi) \wedge (\text{IH}(\varphi) \Rightarrow \varphi)$$

for every trace formula φ. Intuitively, $\text{BC}(\varphi)$ expresses whether φ holds for the empty trace and $\text{IH}(\varphi)$ expresses the induction hypothesis, which states that φ holds for all steps other than the last step of the trace.

To formalize the induction hypothesis, we require a means to exclude instantiations of temporal variables with the last index in the trace. We therefore introduce an additional trace atom. A *last atom*, written $\text{last}(i)$, denotes that the temporal variable i is instantiated with the last index of the trace. We extend the satisfaction relation $\vDash_E$ defined in Section 7.4 such that

$$(tr, \theta) \vDash_E \text{last}(i) \qquad \text{iff} \quad \theta(i) = |tr|$$

for every equational theory E, every trace tr, and every valuation θ.

Let φ be a trace formula. We say that φ is *last-free* if it does not contain any last-atom. For a last-free trace formula φ, the transformations $\text{BC}(\varphi)$ and $\text{IH}(\varphi)$ are given in Figure 8.6. Note that, without loss of generality, we define these transformations under the assumption that formulas are encoded using the connectives $\neg$, $\wedge$, and $\exists$ only. The transformations are characterized by the following two lemmas.

Lemma 3. *For every last-free trace formula φ, every trace tr, and every valuation θ,*

$$([], \theta) \vDash_E \varphi \quad \textit{iff} \quad (tr, \theta) \vDash_E \text{BC}(\varphi)\ .$$

Proof. Intuitively, this holds because the validity of $(tr, \theta) \vDash_E \text{BC}(\varphi)$ is independent of the trace tr, as φ is last-free and $\text{BC}(\varphi)$ replaces all action-atoms in φ with $\bot$. We formally prove this by structural induction over φ. □

Lemma 4. *For every last-free trace formula φ, every trace tr, every set of actions A, and every valuation θ with $\forall i \in vars(\varphi) \cap \mathcal{V}_{temp}.\ \theta(i) \neq |tr| + 1$,*

$$(tr, \theta) \vDash_E \varphi \quad \textit{iff} \quad (tr \cdot [A], \theta) \vDash_E \text{IH}(\varphi)\ .$$

Proof (sketch of the complete proof on page 183). Intuitively, this holds, as the $\neg(\text{last}(i))$ restrictions in $\text{IH}(\varphi)$ avoid that the satisfaction of $\text{IH}(\varphi)$ depends on the actions in A at the end of the trace. We formally prove this by structural induction over φ. The only non-trivial cases are the ones for actions and existential quantification over temporal variables. For an action $f@i$, the statement holds because $\theta(i) \neq |tr| + 1$. The proof of the case for an existentially quantified variable i is rather technical. It exploits the restriction $\neg\text{last}(i)$ introduced by $\text{IH}(\varphi)$ and that trace formulas only care about the order of the instantiations of temporal variables. □

The following theorem uses the $\text{BC}(\varphi)$ and $\text{IH}(\varphi)$ transformations to formalize trace induction as a transformation of trace formulas. We prove the theorem for an arbitrary prefix closed set of traces to make it more widely applicable.

Theorem 6 (Trace Induction)**.** *Let Tr be a prefix closed set of traces. For every closed, last-free trace formula φ, it holds that*

$$Tr \vDash^{\forall}_{E} \varphi \quad \textit{iff} \quad Tr \vDash^{\forall}_{E} \text{BC}(\varphi) \wedge (\text{IH}(\varphi) \Rightarrow \varphi) \ .$$

Proof (sketch of the complete proof on page 184). The proof of the implication from left to right uses Lemma 3 to justify that $Tr \vDash^{\forall}_{E} \varphi$ implies $Tr \vDash^{\forall}_{E} \text{BC}(\varphi)$. Moreover, $Tr \vDash^{\forall}_{E} \varphi$ obviously implies $Tr \vDash^{\forall}_{E} \text{IH}(\varphi) \Rightarrow \varphi$.

The proof of the implication from right to left uses wellfounded induction over the prefix-order of traces to prove that $(tr, \theta) \vDash_{E} \varphi$ holds for all traces $tr \in Tr$ and all valuations θ, provided that $Tr \vDash^{\forall}_{E} \text{BC}(\varphi)$ and $Tr \vDash^{\forall}_{E} \text{IH}(\varphi) \Rightarrow \varphi$ hold. If $tr = []$, then $([], \theta) \vDash_{E} \varphi$ holds because of Lemma 3. Otherwise, we use Lemma 4 to justify the induction step. Note that the lemma's side-condition on the valuation is trivially satisfied, as φ is closed. □

As the set of traces of a multiset rewriting system R is prefix closed, we can specialize Theorem 6 as follows.

Corollary 1. *For every last-free trace formula φ,*

$$R \vDash^{\forall}_{E} \varphi \quad \textit{iff} \quad R \vDash^{\forall}_{E} \text{BC}(\varphi) \wedge (\text{IH}(\varphi) \Rightarrow \varphi) \ .$$

Due to the duality of satisfiability and validity claims, we can also use trace induction when searching for a trace satisfying a trace formula φ.

Corollary 2. *For every last-free trace formula φ,*

$$R \vDash^{\exists}_{E} \varphi \quad \textit{iff} \quad R \vDash^{\exists}_{E} \text{BC}(\varphi) \vee (\neg\text{IH}(\varphi) \wedge \varphi) \ .$$

Intuitively, the transformation of φ to $\neg\text{IH}(\varphi) \wedge \varphi$ limits the search to traces that are *minimal* in the sense that none of their prefixes satisfies φ. This provides an alternative view on the soundness of trace induction. It is sound because, from every trace $tr \in Tr$ satisfying φ, we can construct a minimal trace in Tr by removing all superfluous steps at the end of tr *provided that Tr is prefix closed.*

$\mathtt{R}_{\mathtt{basic}}$: $\quad \Gamma \rightsquigarrow \Gamma_1 \parallel \ldots \parallel \Gamma_n \qquad$ if $\Gamma \rightsquigarrow^{\mathtt{basic}}_{R,E} \Gamma_1 \parallel \ldots \parallel \Gamma_n$

$\mathtt{S}_{\mathrm{last},\lessdot}$: $\quad \Gamma \rightsquigarrow \bot \qquad$ if $(\mathrm{last}(i)) \in \Gamma$ and $i \lessdot_\Gamma j$ and $(j : ri) \in \Gamma$

$\mathtt{S}_{\mathrm{last},\mathrm{last}}$: $\quad \Gamma \rightsquigarrow (i \doteq j, \Gamma) \qquad$ if $\underline{\{\mathrm{last}(i), \mathrm{last}(j)\}} \subseteq \Gamma$; $\boxed{i \neq j}$

$\mathtt{S}_{\neg,\mathrm{last}}$: $\quad \Gamma \rightsquigarrow (i \lessdot j, \mathrm{last}(j), \Gamma) \parallel (\mathrm{last}(j), j \lessdot i, \Gamma)$

if $\underline{\neg(\mathrm{last}(i))} \in \Gamma$ and j fresh; $\boxed{\text{neither } i \lessdot_\Gamma k \text{ nor } k \lessdot_\Gamma i \text{ for any } (\mathrm{last}(k)) \in \Gamma}$

We write $\rightsquigarrow^{\mathtt{last}}_{R,E}$ as $\rightsquigarrow$ and the empty set of constraint systems $\varnothing$ as $\bot$.

Figure 8.7.: Rules defining the constraint-reduction relation $\rightsquigarrow^{\mathtt{last}}_{R,E}$

Example 26 (Application of Trace Induction)**.** We illustrate the use of trace induction on the validity claim from Example 25, which we used to motivate trace induction. Recall that we attempted to prove the validity claim

$$R_{loop} \vDash^{\forall}_{E_{PHS}} \varphi \quad \text{with } \varphi := (\forall x\, i.\, \mathsf{Loop}(x)@i \Rightarrow \exists j.\, \mathsf{Start}(x)@j)\ .$$

Because of Corollary 1, this claim is logically equivalent to

$$\begin{aligned} R_{loop} \vDash^{\forall}_{E_{PHS}} & (\ \forall x\, i.\, \bot \Rightarrow (\exists j.\, \bot)\ ) \wedge \\ & (\ (\forall x\, i.\, \mathsf{Loop}(x)@i \Rightarrow \mathrm{last}(i) \vee (\exists j.\, \mathsf{Start}(x)@j \wedge \neg\mathrm{last}(j))) \Rightarrow \\ & \quad (\forall x\, i.\, \mathsf{Loop}(x)@i \Rightarrow (\exists j.\, \mathsf{Start}(x)@j))\ )\ . \end{aligned}$$

During constraint solving, the induction hypothesis $\textsc{ih}(\varphi)$ becomes just another constraint. We can see this in the constraint system Γ', which characterizes all counter-examples to the above validity claim.

$$\begin{aligned} \Gamma' := \{\ & (\exists x\, i.\, \mathsf{Loop}(x)@i \wedge (\forall j.\, \mathsf{Start}(x)@j \Rightarrow \bot))\ , \\ & (\forall x\, i.\, \mathsf{Loop}(x)@i \Rightarrow \mathrm{last}(i) \vee (\exists j.\, \mathsf{Start}(x)@j \wedge \neg\mathrm{last}(j)))\ \} \end{aligned}$$

The first constraint states that there must exist a counter-example to φ, while the second constraint, equivalent to $\textsc{ih}(\varphi)$, states that this counter-example must be minimal. Using trace induction we can thus restrict our search for counter-examples in a property-specific way to minimal counter-examples. ♠

Trace induction sometimes allows to prove a claim that is out of scope for our basic constraint-reduction relation, e.g., the validity claim at the start of the above example. We prove this claim in the next example, after we introduced the constraint-reduction rules for reasoning about last-atoms.

8.3.3. Constraint-Reduction Rules for Reasoning about Last-Atoms

The constraint-reduction rules given in Figure 8.7 define the $\rightsquigarrow^{\mathtt{last}}_{R,E}$ constraint-reduction relation. This relation extends our basic constraint-reduction relation with support for reasoning about last-atoms.

The rule $\mathtt{R}_{\mathtt{basic}}$ includes all basic constraint-reduction rules in $\rightsquigarrow^{\mathsf{last}}_{R,E}$. The rules $\mathtt{S}_{\mathsf{last},\lessdot}$, $\mathtt{S}_{\mathsf{last},\mathsf{last}}$, and $\mathtt{S}_{\neg,\mathsf{last}}$ solve last-atoms. More precisely, they ensure that the solution construction method from the proof of Theorem 5 also succeeds for constraint systems containing last-atoms. The rule $\mathtt{S}_{\mathsf{last},\lessdot}$ prunes constraint systems with conflicting last and node constraints. The rule $\mathtt{S}_{\mathsf{last},\mathsf{last}}$ ensures that a solved constraint system contains at most one last constraint. The rule $\mathtt{S}_{\neg,\mathsf{last}}$ ensures that the constructed solution satisfied all negated last-atoms.

The following two theorems justify the use of $\rightsquigarrow^{\mathsf{last}}_{R,E}$ for constraint solving.

Theorem 7. *The constraint-reduction relation $\rightsquigarrow^{\mathsf{last}}_{R,E}$ is R,E-wellformed, R,E-complete, and R,E-correct.*

Proof (sketch of the complete proof on page 184). This follows from Theorem 4, the definition of $\rightsquigarrow^{\mathsf{last}}_{R,E}$, and the extended semantics of $\vDash_E$. □

Theorem 8 (Solution Construction for $\rightsquigarrow^{\mathsf{last}}_{R,E}$)**.** *We can construct a solution for every wellformed, $\rightsquigarrow^{\mathsf{last}}_{R,E}$-irreducible constraint system.*

Proof (sketch of the complete proof on page 185). The solution construction works analogously to one in the proof of Theorem 5 on page 100. Because of rule $\mathtt{S}_{\mathsf{last},\mathsf{last}}$, there exists at most one $\mathsf{last}(i)$ constraint. This constraint is respected by instantiating i with the length of the constructed sequence of rewriting rule instances. □

The following example illustrates how trace induction strengthens our constraint-solving based search for counter-examples.

Example 27 (Trace Induction and Constraint Solving)**.** We now prove the validity claim $R_{loop} \vDash^{\forall}_{E_{PHS}} (\forall x\ i.\ \mathsf{Loop}(x)@i \Rightarrow \exists j.\ \mathsf{Start}(x)@j)$ from Example 25, which we failed to prove using our basic constraint-reduction rules. Recall from Example 26 that proving this validity claim is equivalent to showing that the constraint system

$$\begin{aligned}\Gamma' := \{\ & (\exists x\ i.\ \mathsf{Loop}(x)@i \wedge (\forall j.\ \mathsf{Start}(x)@j \Rightarrow \bot))\ , \\ & (\forall x\ i.\ \mathsf{Loop}(x)@i \Rightarrow \mathsf{last}(i) \vee (\exists j.\ \mathsf{Start}(x)@j \wedge \neg(\mathsf{last}(j))))\ \}\end{aligned}$$

has no R_{loop}, E_{PHS}-solutions. We prove this using the constraint-reduction relation $\rightsquigarrow^{\mathsf{last}}_{R,E}$.

We first decompose the guarded trace formulas of the System Γ' as usual. This results in exactly one system, which we depict in Figure 8.8 on the next page as System Γ'_1. Compared to System Γ_1 from our previous proof attempt in Example 25, System Γ'_1 contains the additional constraint ψ_2 denoting the induction hypothesis.

We proceed as in Example 25 until we reach System Γ'_{1^3}. In this system, we exploit the constraint ψ_2 to derive $\mathsf{last}(j_1) \vee (\exists j.\ (\mathsf{Start}(x)@j) \wedge \neg(\mathsf{last}(j)))$ instead of solving the premise $\mathsf{A}(x) \blacktriangleright_1 j_1$. After a case-split on the disjunction, the systems Γ'_{1^4} and $\Gamma'_{1^3,2}$ result.

Intuitively, System Γ'_{1^4} covers the proof obligation that we can only apply the induction hypothesis to nodes other than the last node. Formally, we show that System Γ'_{1^4} is contradictory by applying rule $\mathtt{S}_{\mathsf{last},\lessdot}$ to $\mathsf{last}(j_1)$ and node i.

System $\Gamma'_{1^3,2}$ corresponds to a successful application of the induction hypothesis. We hence obtain a non-last node j with a $\mathsf{Start}(x)@j$ action using the rule $\mathtt{S}_{\exists}$. Thus, we can apply the rule $\mathtt{S}_{\forall}$ to derive a contradiction from ψ_1. This concludes the proof of the validity claim from Example 25. ♠

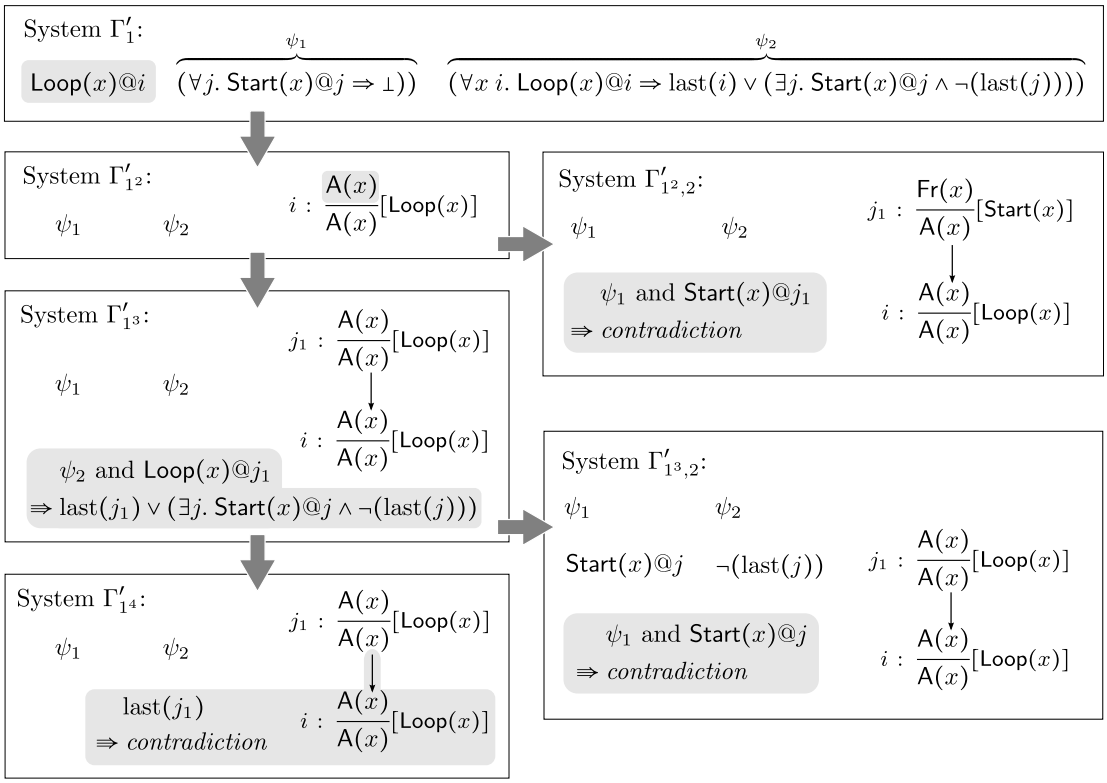


Figure 8.8.: Depiction of the constraint systems and constraint-reduction steps from Example 27 illustrating a proof using trace induction.

This example demonstrates that trace induction increases the scope of our basic constraint-reduction rules. Note that trace induction makes few assumptions on the property of interest and is therefore widely applicable. In Section 8.4.4, we use it for example to prove the soundness of type assertions. We also use trace induction to prove that the TESLA protocol satisfies stream authentication, as formalized in Section 7.5.2.

8.4. Reasoning about Message Deduction

In this section, we extend the constraint-reduction relation $\rightsquigarrow^{\mathsf{last}}_{R,E}$ from the previous section with a set of constraint-reduction rules for reasoning about message deduction. Before explaining the approach we used to design them, we give an example demonstrating the insufficiency of $\rightsquigarrow^{\mathsf{last}}_{R,E}$ for reasoning about message deduction.

Example 28 (Reasoning about Message Deduction using $\rightsquigarrow^{\mathsf{last}}_{R,E}$)**.** Consider the protocol P_{Ex} from Example 18 on page 79 and the trace formula

$$\varphi := \forall x\, k\, i.\ \mathsf{Fin}(x,k)@i \Rightarrow \exists j.\ \mathsf{Rev}(k)@j\ .$$

Intuitively, $P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\forall}_{E_{PHS}} \varphi$ holds, as a protocol run can finish only if the adversary reveals the run's short-term key and constructs the pair required to finish. In

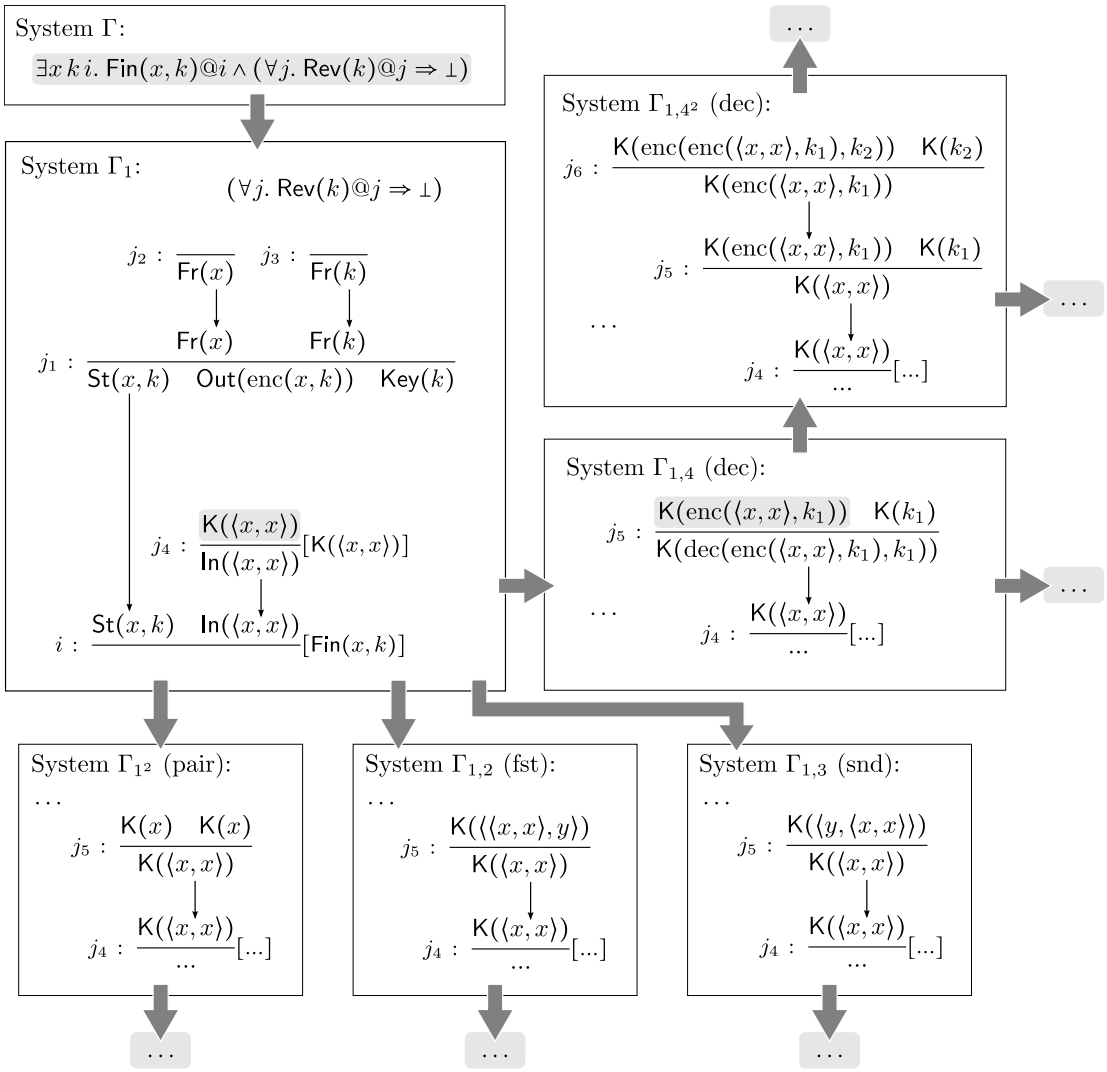


Figure 8.9.: Constraint systems demonstrating the insufficiency of $\rightsquigarrow^{\text{last}}_{R,E}$ for reasoning about message deduction.

the following, we demonstrate that we cannot show this formally using the constraint-reduction relation $\rightsquigarrow^{\text{last}}_{(P_{Ex} \cup MD_{\Sigma_{PHS}}),E_{PHS}}$.

To prove $P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\forall}_{E_{PHS}} \varphi$, we must show that the constraint system Γ depicted in Figure 8.9 has no solutions. We solve constraints as in our previous examples. After solving all constraints except K-premises, we obtain the single constraint system Γ_1.

We would now like to show that the premise $\mathsf{K}(\langle x, x\rangle) \blacktriangleright_1 j_4$ implies the existence of a $\mathsf{Rev}(k)$ action. There is only one way to continue using $\rightsquigarrow^{\text{last}}_{(P_{Ex} \cup MD_{\Sigma_{PHS}}),E_{PHS}}$: applying the $\mathsf{S}_{\blacktriangleright}$ rule to the premise $\mathsf{K}(\langle x, x\rangle) \blacktriangleright_1 j_4$. We thus enumerate all conclusions of multiset-rewriting rules in $P_{Ex} \cup MD_{\Sigma_{PHS}} \cup \{\text{FRESH}\}$ and solve the introduced equality constraint using unification modulo E_{PHS}. There are four resulting constraint systems.

System $\Gamma_{1,1}$ formalizes that the adversary could construct the pair $\langle x, x\rangle$ by himself.

Systems $\Gamma_{1,2}$ and $\Gamma_{1,3}$ formalize that the adversary could extract the term $\langle x, x\rangle$ from some pair by applying the fst or the snd function. System $\Gamma_{1,4}$ formalizes that the adversary could extract the term $\langle x, x\rangle$ from some encryption $\mathrm{enc}(\langle x, x\rangle, k_1)$ by applying the dec function. Note that we did not reduce the term $\mathrm{dec}(\mathrm{enc}(\langle x, x\rangle, k_1), k_1)$ in System $\Gamma_{1,4}$ to $\langle x, x\rangle$ in order to illustrate that all these constraint systems are interpreted modulo E_{PHS}.

It is easy to see from Figure 8.9 that the $\rightsquigarrow^{\mathtt{last}}_{(P_{Ex} \cup MD_{\Sigma_{PHS}}), E_{PHS}}$ relation is not sufficient to finish the proof. One of the problems is that the message deduction rules allow deducing the same message in several different ways. During constraint solving, we must explore all of these ways, as any one of them might lead to a solution. The core problem is however that the backwards-reasoning as implemented by $\rightsquigarrow^{\mathtt{last}}_{R,E}$ does not suffice to reason about messages deduced using decryption and similar deconstruction functions. As we can see in System $\Gamma_{1,4^2}$, backwards-reasoning just introduces larger and larger premises. As we will see, we solve this core problem by reasoning about these functions in a forward fashion starting from the messages sent by the protocol, analogously to decryption-chain reasoning. ♠

Approach Taken In the following, we show that for some equational theories we can reason about message deduction using an approach similar to decryption-chain reasoning, explained in Part I. We derive this approach in three steps, which we outline below. We explain these steps in the following sections.

In the first step, we show that we can use the finite variant property [CLD05] to switch from constraint solving modulo E to constraint solving modulo an equational theory $E' \subset E$. We typically choose E' such that it does not contain cancellation equations, as they require special treatment during constraint solving. For example, the finite variant property allows us to switch from reasoning modulo E_{PHS} to reasoning modulo the empty equational theory, i.e., to use syntactic unification during constraint solving. The corresponding variants of the rules in $MD_{\Sigma_{PHS}}$ correspond to the classical Dolev-Yao message deduction rules for pairing, hashing, and symmetric encryption. The message deduction rules for projection and decryption reflect the successful application of one of the cancellation rules in E_{PHS}. Note that this first step restricts our approach to equational theories that have the finite variant property. Fortunately, many theories relevant for security protocol verification have this property [CLD05].

In the second step, we impose normal form conditions on the use of the variants of the message deduction rules. The key objective of these normal form conditions is to exclude redundant ways of deducing messages and thereby reduce the search space during constraint solving. We define the normal form dependency graphs as the dependency graphs that satisfy these normal form conditions. In the formulation of our normal form conditions, we exploit the finite variant property. It allows us to specify normal form conditions that depend on whether a cancellation rule has been applied.

Note that the concrete normal form conditions depend on the equational theory under consideration. In [SMCB12a], we give normal form conditions that are sufficient for reasoning about the combination of a subterm convergent rewriting theory and an equational theory modeling Diffie-Hellman exponentiation whose exponents form an abelian group. In his thesis [Sch12], Schmidt further extends these normal form conditions to additionally support reasoning about bilinear pairings and multiset union.

Here, we explain this second step only on the simple equational theory E_{PHS} from Example 17. The goal of our exposition is to highlight the key normal form conditions required for reasoning about message deduction. Our normal form conditions generalize the constructions underlying the security protocol semantics from Part I. Their key consequence is that the resulting normal form dependency graphs satisfy the so-called deconstruction-chain property. We exploit this property during constraint solving to search for messages deduced using deconstruction functions (e.g., decryption and projection) in a forward-fashion starting from messages sent by the protocol. As for decryption-chain reasoning, this approach allows us to make local progress when searching for messages deduced using deconstruction functions, as the term resulting from the successful application of a deconstruction function is smaller than the function's arguments.

In the third step, we introduce constraint-reduction rules that allow us to search for normal form dependency graphs. They use deconstruction-chain constraints to reason about messages deduced using deconstruction functions. The resulting search space is often finite and therefore allows the successful construction of validity proofs. We do not have meta-theoretical results that formally characterize the strength of our constraint-reduction rules. In Section 9.3, we therefore validate their effectiveness on a large number of case studies.

8.4.1. Exploiting the Finite Variant Property

We first introduce the notion of finite variants [CLD05, ESM12] and illustrate it on the equational theory E_{PHS}. Then, we show how we can use finite variants to change the equational theory used for constraint solving.

Let $\mathcal{R}$ be a set of term rewriting rules and $\mathcal{AX}$ be a set of equations such that $\mathcal{R}^{\simeq} \cup \mathcal{AX}$ is an equational presentation of E. Assume moreover that $\mathcal{R}$ is $\mathcal{AX}$-convergent and $\mathcal{AX}$-coherent, as defined in Section 2.2. The notion of $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normal terms is thus well-defined and we say that a dependency graph (I, D) is $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$*-normal* if all rule instances in I are $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normal.

We assume that E has the finite variant property [CLD05] for this presentation, which allows us to perform symbolic reasoning about $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normalization. The finite variant property states that, for all terms t, there is a finite set of substitutions $\{\tau_1, \ldots, \tau_k\}$ with $dom(\tau_i) \subseteq vars(t)$ such that for all substitutions σ, there is an $i \in \{1, \ldots, k\}$ and a substitution σ' with $(t\sigma)\downarrow^{\mathcal{R}}_{\mathcal{AX}} =_{\mathcal{AX}} ((t\tau_i)\downarrow^{\mathcal{R}}_{\mathcal{AX}})\sigma'$ and $(x\sigma)\downarrow^{\mathcal{R}}_{\mathcal{AX}} =_{\mathcal{AX}} x\tau_i\sigma'$ for all $x \in vars(t)$. We call such a set $\{(t\tau_i\downarrow^{\mathcal{R}}_{\mathcal{AX}}, \tau_i) \mid 1 \leq i \leq k\}$ a *complete set of* $\mathcal{R}$,$\mathcal{AX}$*-variants of* t. For a given term t, we use folding variant narrowing [ESM12] to compute such a set, which we denote by $\lceil t \rceil^{\mathcal{R}}_{\mathcal{AX}}$. Overloading notation, we also denote $\{s \mid (s, \tau) \in \lceil t \rceil^{\mathcal{R}}_{\mathcal{AX}}\}$ by $\lceil t \rceil^{\mathcal{R}}_{\mathcal{AX}}$. It is straightforward to extend these notions to multiset rewriting rules by considering rules as terms and the required new function symbols as free.

Example 29 (Finite Variants for E_{PHS})**.** We define the rewriting system PHS as

$$PHS := \{\, \mathrm{fst}(\langle x, y\rangle) \to x, \quad \mathrm{snd}(\langle x, y\rangle) \to y, \quad \mathrm{dec}(\mathrm{enc}(m, k), k) \to m \,\}\ .$$

Obviously $PHS^{\simeq} \uplus \varnothing$ is an equational presentation of E_{PHS}. This presentation moreover has the finite variant property, as PHS is a subterm-convergent rewriting system [CLD05].

Intuitively, the $PHS, \varnothing$-variants of a term characterize its $\downarrow_{\varnothing}^{PHS}$-normal instances and allow us to switch from reasoning modulo E_{PHS} to purely syntactic reasoning. For example, we can compute a complete set of E_{PHS}-unifiers for the equation $\mathrm{dec}(x_1, x_2) \approx \mathrm{h}(x_2)$ by jointly computing the variants of both sides of the equation and solving the corresponding syntactic unification problem for each variant. More precisely, a complete set of variants of $\mathrm{dec}(x_1, x_2) \approx \mathrm{h}(x_2)$ is

$$\begin{aligned} V &:= \lceil \mathrm{dec}(x_1, x_2) \approx \mathrm{h}(x_2) \rceil_{\varnothing}^{PHS} \\ &= \{(\quad y_1 \approx \mathrm{h}(y_2), \quad \{x_1 \mapsto \mathrm{enc}(y_1, y_2),\ x_2 \mapsto y_2\} \quad) \\ &\quad , (\quad \mathrm{dec}(y_1, y_2) \approx \mathrm{h}(y_2), \quad \{x_1 \mapsto y_1,\ x_2 \mapsto y_2\} \quad)\} \ . \end{aligned}$$

The first variant covers the cases where the decryption algorithm was successfully applied; i.e., where it was applied to an encryption with the same key. The second variant covers all the cases where decryption failed. From these variants, we compute a complete set of E_{PHS}-unifiers for $\mathrm{dec}(x_1, x_2) \approx \mathrm{h}(x_2)$ as

$$\begin{aligned} U &:= \bigcup_{(t \approx s, \tau) \in \lceil \mathrm{dec}(x_1, x_2) \approx \mathrm{h}(x_2) \rceil_{\varnothing}^{PHS}} \{\tau\sigma \mid \sigma \in \mathit{unify}_{\varnothing}(t \approx s) \text{ where } (t \approx s)\sigma \text{ is } \downarrow_{\varnothing}^{PHS}\text{-normal}\} \\ &= \{\{x_1 \mapsto \mathrm{enc}(\mathrm{h}(y_2), y_2), x_2 \mapsto y_2\}\} \ . \end{aligned}$$

The set of unifiers U is complete because of the following reasoning. We must show that, for every unifier σ of $\mathrm{dec}(x_1, x_2) \approx \mathrm{h}(x_2)$, there exists a substitution $\rho \in U$ and a substitution ρ' such that σ is equal to $\rho\rho'$ modulo E_{PHS}. As V is a complete set of variants, there must be a variant $(t \approx s, \tau) \in V$ and a substitution σ' such that $((\mathrm{dec}(x_1, x_2) \approx \mathrm{h}(x_2))\sigma)\downarrow_{\varnothing}^{PHS} =_{\varnothing} (t \approx s)\sigma'$ and $(x\sigma)\downarrow_{\varnothing}^{PHS} =_{\varnothing} x\tau\sigma'$ for all $x \in \mathit{vars}(\mathrm{dec}(x_1, x_2) \approx \mathrm{h}(x_2))$. Note that σ' is a $\varnothing$-unifier of $t \approx s$, as σ is an E_{PHS}-unifier of $\mathrm{dec}(x_1, x_2) \approx \mathrm{h}(x_2)$. Thus, there exists a substitution $\alpha \in \mathit{unify}_{\varnothing}(t \approx s)$ such that $\sigma' =_{\varnothing} \alpha\alpha'$ for some substitution α'. Moreover, $(t \approx s)\alpha$ must $\downarrow_{\varnothing}^{PHS}$-normal, as $(t \approx s)\alpha\alpha'$ is $\downarrow_{\varnothing}^{PHS}$-normal. We thus have that $\tau\alpha \in U$ and the substitution $(\tau\alpha)\alpha'$ is equal to σ modulo E_{PHS}, which justifies the completeness of the set of unifiers U. ♠

Note that in our case the key benefit of the finite variant property is not that it allows us to solve E-unification problems. In fact, solving the resulting $\mathcal{AX}$-unification problems might even be more expensive than solving the original E-unification problem. In our case, the key benefit is that the finite variant property allows us to specify normal-form conditions on dependency graphs that depend on whether a cancellation rule was successfully applied. We also exploit the finite variant property in the following theorem, which allows us to switch from validity claims modulo E to validity claims modulo $\mathcal{AX}$.

Theorem 9 (Changing the Equational Theory)**.** *For every multiset rewriting system R and every guarded trace property φ,*

$$R \vDash_E^{\forall} \varphi \quad \textit{iff} \quad \{\mathit{trace}(dg) \mid dg \in \mathit{dgraphs}_{\mathcal{AX}}(\lceil R \rceil_{\mathcal{AX}}^{\mathcal{R}}),\ dg \text{ is } \downarrow_{\mathcal{AX}}^{\mathcal{R}}\textit{-normal}\} \vDash_{\mathcal{AX}}^{\forall} \varphi$$

Proof. We first show that the satisfaction of the guarded trace property φ is invariant under $\downarrow_{\mathcal{AX}}^{\mathcal{R}}$-normalization; i.e., $(tr, \theta) \vDash_E \varphi$ iff $(tr\downarrow_{\mathcal{AX}}^{\mathcal{R}}, \theta) \vDash_{\mathcal{AX}} \varphi$, for every trace tr and

every valuation θ. This holds because φ is closed and all its terms are either variables or public names, which implies that φ cannot distinguish between normal and non-normal terms. Then, we show that

$$dgraphs_E(R)\downarrow^{\mathcal{R}}_{\mathcal{AX}} =_{\mathcal{AX}} \{dg \mid dg \in dgraphs_{\mathcal{AX}}(\lceil R \rceil^{\mathcal{R}}_{\mathcal{AX}}),\ dg \text{ is } \downarrow^{\mathcal{R}}_{\mathcal{AX}}\text{-normal}\}\ .$$

This holds because $\lceil R \rceil^{\mathcal{R}}_{\mathcal{AX}}$ is a complete set of $\mathcal{R}$,$\mathcal{AX}$-variants. These two propositions imply the claimed equivalence. □

Example 30 ($\downarrow^{PHS}_{\varnothing}$-normal Dependency Graphs)**.** To normalize the dependency graph $dg = (I, D)$ in Figure 8.1 on page 92 with respect to $\downarrow^{PHS}_{\varnothing}$, it suffices to replace I_7 with $ri := [\mathsf{K}(\mathrm{enc}(\mathsf{a},\mathsf{k})), \mathsf{K}(\mathsf{k})] \text{–}[]\text{→}[\mathsf{K}(\mathsf{a})]$. We call the result dg'. Since ri is not an instance of the decryption rule $rdec := [\mathsf{K}(x), \mathsf{K}(y)] \text{–}[]\text{→}[\mathsf{K}(\mathrm{dec}(x, y))]$ or any other rule in $P \cup MD \cup \{\textsc{Fresh}\}$, dg' is not in $dgraphs_{\varnothing}(P \cup MD)$. However, ri is an instance of $\mathsf{K}(\mathrm{enc}(x, y)), \mathsf{K}(y) \text{–}[]\text{→} \mathsf{K}(x)$, which is one of the two $PHS,\varnothing$-variants of $rdec$. Note that

$$\lceil rdec \rceil^{PHS}_{\varnothing} = \{[\mathsf{K}(\mathrm{enc}(x, y)), \mathsf{K}(y)] \text{–}[]\text{→}[\mathsf{K}(x)],\ [\mathsf{K}(x), \mathsf{K}(y)] \text{–}[]\text{→}[\mathsf{K}(\mathrm{dec}(x, y))]\}$$

and $\lceil rdec \rceil^{PHS}_{\varnothing} \subseteq \lceil MD \rceil^{PHS}_{\varnothing}$. Thus, $dg' \in dgraphs_{\varnothing}(\lceil P \cup MD \rceil^{PHS}_{\varnothing})$. ♠

8.4.2. Normal Form Dependency Graphs

In this section, we show how to define the set of normal form dependency graphs of a protocol P. This definition depends on the equational theory considered during the execution of P. To simplify our presentation, we first define the normal form dependency graphs for the equational theory E_{PHS} only. Afterwards, we explain how to generalize the definition given for E_{PHS}.

Normal Form Dependency Graphs for E_{PHS}

We define the normal form dependency graphs for E_{PHS} in two steps. In the first step, we define the corresponding normal form message deduction rules ND. These rules allow the adversary to deduce the same messages as the message deduction rules $MD_{\Sigma_{PHS}}$, even when using only syntactic equality instead of equality modulo E_{PHS}. Moreover, the normal form message deduction rules distinguish between messages that may be further deconstructed and messages where deconstruction is forbidden. We exploit this distinction later when formalizing the deconstruction-chain property. In the second step, we define the normal form dependency graphs as the dependency graphs in $dgraphs_{\varnothing}(\lceil P \rceil^{PHS}_{\varnothing} \cup ND)$ that satisfy certain normal form conditions, which exclude additional redundancies.

The *normal form message deduction rules* ND are given in Figure 8.10. All of these rules except the Coerce rule are specialized versions of the $PHS,\varnothing$-variants of the message deduction rules $MD_{\Sigma_{PHS}}$. We use $\mathsf{K}^{\downarrow}$-facts to tag messages that may be further deconstructed and $\mathsf{K}^{\uparrow}$-facts where deconstruction is forbidden. The reasoning behind our specialized versions is the following.

We allow the adversary to convert a $\mathsf{K}^{\downarrow}$-fact to a $\mathsf{K}^{\uparrow}$-fact using the Coerce rule. The action on the Coerce rule logs the message the adversary deduced using it. In Section 8.4.4, we use this and the other actions on the rules with a $\mathsf{K}^{\uparrow}$-conclusion to

Communication rules: $\textsc{IRecv}\ \dfrac{\mathsf{Out}(x)}{\mathsf{K}^{\downarrow}(x)}$ $\quad$ $\textsc{ISend}\ \dfrac{\mathsf{K}^{\uparrow}(x)}{\mathsf{In}(x)}[\mathsf{K}(x)]$

Construction rules:

$$\frac{}{\mathsf{K}^{\uparrow}(x{:}pub)}[\mathsf{K}^{\uparrow}(x{:}pub)] \qquad \frac{\mathsf{Fr}(x{:}fresh)}{\mathsf{K}^{\uparrow}(x{:}fresh)}[\mathsf{K}^{\uparrow}(x{:}fresh)] \qquad \frac{\mathsf{K}^{\uparrow}(x)}{\mathsf{K}^{\uparrow}(\mathrm{h}(x))}[\mathsf{K}^{\uparrow}(\mathrm{h}(x))]$$

$$\frac{\mathsf{K}^{\uparrow}(x) \quad \mathsf{K}^{\uparrow}(y)}{\mathsf{K}^{\uparrow}(\mathrm{enc}(x,y))}[\mathsf{K}^{\uparrow}(\mathrm{enc}(x,y))] \qquad \frac{\mathsf{K}^{\uparrow}(x) \quad \mathsf{K}^{\uparrow}(y)}{\mathsf{K}^{\uparrow}(\mathrm{dec}(x,y))}[\mathsf{K}^{\uparrow}(\mathrm{dec}(x,y))]$$

$$\frac{\mathsf{K}^{\uparrow}(x) \quad \mathsf{K}^{\uparrow}(y)}{\mathsf{K}^{\uparrow}(\langle x,y\rangle)}[\mathsf{K}^{\uparrow}(\langle x,y\rangle)] \qquad \frac{\mathsf{K}^{\uparrow}(x)}{\mathsf{K}^{\uparrow}(\mathrm{fst}(x))}[\mathsf{K}^{\uparrow}(\mathrm{fst}(x))] \qquad \frac{\mathsf{K}^{\uparrow}(x)}{\mathsf{K}^{\uparrow}(\mathrm{snd}(x))}[\mathsf{K}^{\uparrow}(\mathrm{snd}(x))]$$

Deconstruction rules: $\dfrac{\mathsf{K}^{\downarrow}(\langle x,y\rangle)}{\mathsf{K}^{\downarrow}(x)}$ $\quad$ $\dfrac{\mathsf{K}^{\downarrow}(\langle x,y\rangle)}{\mathsf{K}^{\downarrow}(y)}$ $\quad$ $\dfrac{\mathsf{K}^{\downarrow}(\mathrm{enc}(x,y)) \quad \mathsf{K}^{\uparrow}(y)}{\mathsf{K}^{\downarrow}(x)}$

Coerce rule: $\textsc{Coerce}\ \dfrac{\mathsf{K}^{\downarrow}(x)}{\mathsf{K}^{\uparrow}(x)}[\mathsf{K}^{\uparrow}(x)]$

Figure 8.10.: Normal message deduction rules *ND* for the equational theory E_{PHS} computed from the $PHS, \varnothing$-variants of the message deduction rules $MD_{\Sigma_{PHS}}$

formalize invariants about message deduction. For now, we can ignore these actions. The *communication rules* allow the adversary to communicate with the protocol. Received messages may be further deconstructed, while the sent messages must have been constructed. The *construction rules* allow the adversary to compose constructed messages further. Note that all their $\mathsf{K}^{\uparrow}$-premises are smaller than their $\mathsf{K}^{\uparrow}$-conclusion. This allows us to effectively solve $\mathsf{K}^{\uparrow}$-premises in a backwards-fashion during constraint solving. The *deconstruction rules* allow the adversary to deconstruct messages. The adversary may need to construct additional message for a deconstruction rule to be applicable. For example, the deconstruction rule for decryption requires the adversary to have constructed the decryption key.

Note that the IRecv rule is the only rule in *ND* that provides a $\mathsf{K}^{\downarrow}$-conclusion, but does not require a $\mathsf{K}^{\downarrow}$-premise. This implies the deconstruction-chain property, which informally states that every $\mathsf{K}^{\downarrow}$-premise must be provided by a chain of deconstruction rules starting from an instance of the IRecv rule. We formalize this property as follows. Let $dg = (I, D)$ be a dependency graph in $dgraphs_{\varnothing}(\lceil P \rceil^{PHS}_{\varnothing} \cup ND)$. Its *deconstruction-chain relation* $\dashrightarrow_{dg}$ is the smallest relation such that $c \dashrightarrow_{dg} p$ if c is a $\mathsf{K}^{\downarrow}$-conclusion in dg and (a) $c \rightarrowtail p \in D$ or (b) there is a premise (j, u) such that $c \rightarrowtail (j, u) \in D$ and $(j, 1) \dashrightarrow_{dg} p$.

Example 31. Let dg be a dependency graph containing Graph (c) from Figure 8.11. We have $(j_1, 1) \dashrightarrow_{dg} (j_2, 1)$, $(j_2, 1) \dashrightarrow_{dg} (j_4, 1)$, $(j_1, 1) \dashrightarrow_{dg} (j_4, 1)$, $(j_1, 1) \dashrightarrow_{dg} (j_3, 1)$, $(j_3, 1) \dashrightarrow_{dg} (j_5, 1)$, and $(j_1, 1) \dashrightarrow_{dg} (j_5, 1)$, but not $(j_1, 1) \dashrightarrow_{dg} (i, 2)$ because $(i, 2)$ is not a $\mathsf{K}^{\downarrow}$-premise. ♠

Lemma 5 (Deconstruction-Chain Property)**.** *Let P be a protocol that does not use any $\mathsf{K}^{\uparrow}$- or $\mathsf{K}^{\downarrow}$-facts. For every dependency graph $dg \in dgraphs_{\varnothing}(\lceil P \rceil_{\varnothing}^{PHS} \cup ND)$ and every premise p with fact $\mathsf{K}^{\downarrow}(t)$ of dg, there is a node i in dg such that $I_i \in ginsts(\text{IRECV})$ and $(i, 1) \dashrightarrow_{dg} p$.*

Proof. By induction over the length of the rewriting sequences of the dependency graphs in $dgraphs_{\varnothing}(\lceil P \rceil_{\varnothing}^{PHS} \cup ND)$. □

In the next section, we explain how we use the deconstruction-chain property during constraint solving. Now, we define the normal form dependency graphs.

Let P be a protocol that does not use any $\mathsf{K}^{\uparrow}$- or $\mathsf{K}^{\downarrow}$-tags. A *normal form dependency graph for P* is a dependency graph $dg \in dgraphs_{\varnothing}(\lceil P \rceil_{\varnothing}^{PHS} \cup ND)$ that satisfies the following four normal form conditions.

N1 The dependency graph dg is $\downarrow_{\varnothing}^{PHS}$-normal.

N2 No instance of the CoERCE rule deduces a pair.

N3 If there are two conclusions $(i, 1)$ and $(j, 1)$ deducing the same $\mathsf{K}^{d}(m)$ fact with $d \in \{\uparrow, \downarrow\}$, then $i = j$.

N4 If there is a conclusion $(i, 1)$ with fact $\mathsf{K}^{\downarrow}(m)$ and a conclusion $(j, 1)$ with fact $\mathsf{K}^{\uparrow}(m)$, then $i < j$.

We denote the set of all normal form dependency graphs of P by $ndgraphs_{ND,PHS,\varnothing}(P)$. We define the corresponding *normal form traces* as $ntraces_{ND,PHS,\varnothing}(P) := \{trace(dg) \mid dg \in ndgraphs_{ND,PHS,\varnothing}(P)\}$.

In the following, we first state and prove the theorem formalizing that the normal form traces are a faithful model of the execution of P when considering guarded trace properties. Afterwards, we explain the normal form conditions and give examples of the redundancies eliminated in normal form dependency graphs.

Theorem 10 (Normal Form Traces for E_{PHS})**.** *For every guarded trace property φ that does not use any $\mathsf{K}^{\uparrow}$-facts,*

$$P \cup MD_{\Sigma_{PHS}} \vDash^{\forall}_{E_{PHS}} \varphi \qquad \textit{iff} \qquad ntraces_{ND,PHS,\varnothing}(P) \vDash^{\forall}_{\varnothing} \varphi \ .$$

Proof. We first use Theorem 9 to switch from validity modulo E_{PHS} to validity modulo the empty equational theory. Then, we show that we can convert every $\downarrow_{\varnothing}^{PHS}$-normal dependency graph $dg \in dgraphs_{\varnothing}(\lceil P \cup MD \rceil_{\varnothing}^{PHS})$ to a dependency graph $dg' \in ndgraphs_{ND,PHS,\varnothing}(P)$ (and vice versa) such that the traces $trace(dg)$ and $trace(dg')$ are equivalent modulo the insertion and removal of elements of the form $\varnothing$ or $\{\mathsf{K}^{\uparrow}(m)\}$. This conversion is a special case of the proof given in [SMCB12b, Appendix D.2]. Schmidt details this proof further in his thesis [Sch12]. The validity of φ is invariant under the insertion and removal of elements of the form $\varnothing$ or $\{\mathsf{K}^{\uparrow}(m)\}$, as temporal variables are interpreted over $\mathbb{Q}$ and φ does not contain any $\mathsf{K}^{\uparrow}$-facts. This concludes the proof. □

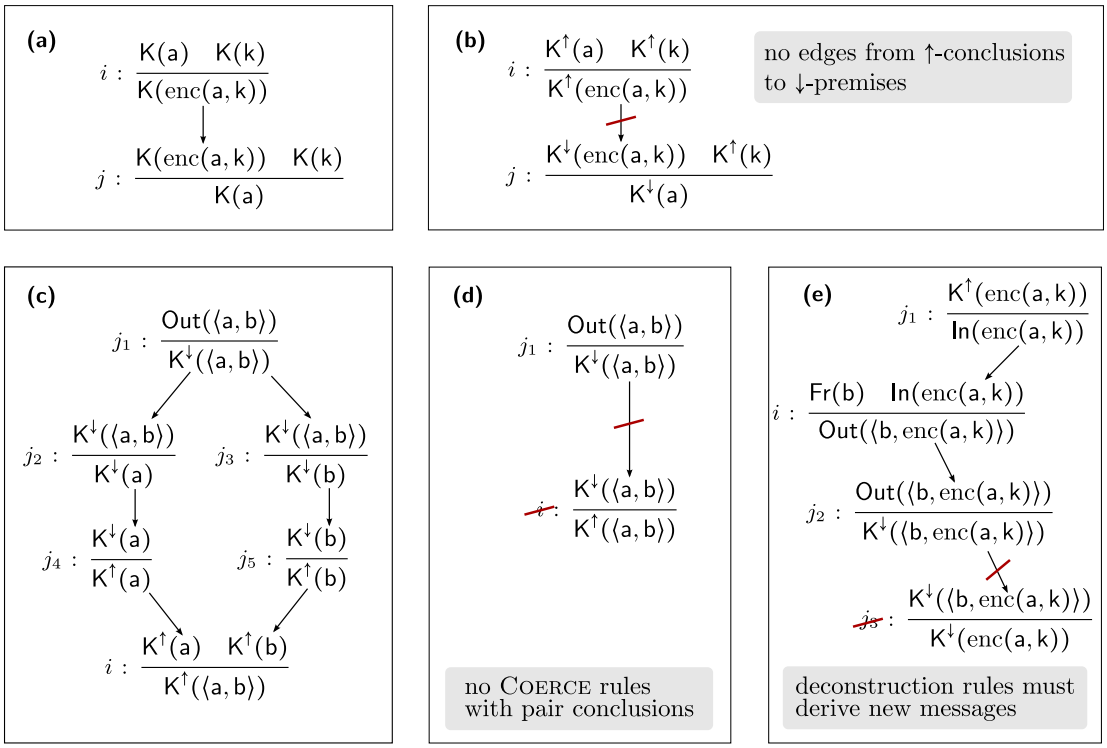


Figure 8.11.: Examples of redundancies eliminated in normal form dependency graphs. Disallowed nodes and edges are crossed out. We assume $i, j, j_1, \ldots, j_5 \in \mathbb{N}$ and $\mathsf{a}, \mathsf{b}, \mathsf{k} \in \mathit{FN}$. We do not depict the actions of construction rules, as they are irrelevant for these examples.

Figure 8.11 depicts some of the redundancies eliminated in normal form dependency graphs. Graph (a) depicts a redundant decryption step that is allowed when using the $\mathit{PHS}, \varnothing$-variants of the message deduction rules. This redundancy is eliminated by the $\uparrow\downarrow$-tags used in the normal form message deduction rules, as illustrated in Graph (b).

Condition **N1** excludes the redundancy of considering instances of different $\mathit{PHS}, \varnothing$-variants that are equal modulo E_{PHS}. As we use a complete set variants, every instance of a variant that is not $\downarrow_{\varnothing}^{\mathit{PHS}}$-normal is an instance of another variant. It thus suffices to consider $\downarrow_{\varnothing}^{\mathit{PHS}}$-normal rule instances only.

Condition **N2** requires that pairs are always constructed using the construction rule for pairs, which renders solving $\mathsf{K}^{\uparrow}(\langle t_1, t_2 \rangle) \blacktriangleright_v i$ premises trivial. The graphs (c) and (d) in Figure 8.11 illustrate the redundancy eliminated by this condition. They depict two alternative deductions of the pair $\mathsf{K}^{\uparrow}(\langle \mathsf{a}, \mathsf{b} \rangle)$ from $\mathsf{Out}(\langle \mathsf{a}, \mathsf{b} \rangle)$. Condition **N2** excludes Graph (d).

Condition **N3** excludes multiple deductions of the same $\mathsf{K}^{\uparrow}$- or $\mathsf{K}^{\downarrow}$-conclusion. This does not exclude any traces because $\mathsf{K}^{\uparrow}$- and $\mathsf{K}^{\downarrow}$-facts are persistent and they do not occur jointly with other conclusions. During constraint solving we exploit Condition **N3** to merge $\mathsf{K}^{\uparrow}$- and $\mathsf{K}^{\downarrow}$-conclusions deriving the same message.

Condition **N4** requires that, if the same message is deduced using a deconstruction and a construction rule, then the deconstruction must happen before the construction.

Conditions **N3** and **N4** together entail that all messages deconstructed from a message sent by the protocol must be new in the sense that they were neither constructed nor deconstructed before. Graph (e) in Figure 8.11 depicts a redundancy excluded by Condition **N4**. In Graph (e), the adversary forwards the encryption $\mathsf{enc}(\mathsf{a},\mathsf{k})$ to himself via an instance of the protocol rule $\mathsf{Fr}(x),\mathsf{In}(y)\text{–}[]\text{→}\mathsf{Out}(\langle x,y\rangle)$. This forwarding is redundant because the adversary either constructed $\mathsf{K}^{\uparrow}(\mathsf{enc}(\mathsf{a},\mathsf{k}))$ by himself and therefore already derived $\mathsf{K}^{\uparrow}(\mathsf{a})$ or he derived $\mathsf{K}^{\uparrow}(\mathsf{enc}(\mathsf{a},\mathsf{k}))$ using the Coerce rule and therefore already derived $\mathsf{K}^{\downarrow}(\mathsf{enc}(\mathsf{a},\mathsf{k}))$ before obtaining it from node j_3. Note that such forwarding opportunities arise from all protocol rules that use a variable v such that v's contents were received from the public channel and v occurs in an extractable position of a message sent on the public channel. Such forwarding is therefore quite common and we require Condition **N4** to prune it during constraint solving.

Normal Form Dependency Graphs for More General Equational Theories

The definition of normal form dependency graphs given in the previous section does not especially exploit the structure of the equational theory E_{PHS}. It is thus straightforward to generalize it to further equational theories provided that we find a suitable set of normal form message deduction rules and normal form conditions.

In [SMCB12b], we show how to construct normal form message deduction rules for the disjoint combination of a subterm-convergent rewriting theory and an equational theory modeling Diffie-Hellman exponentiation whose exponents form an Abelian group. We also introduce further normal form conditions to exclude additional redundancies stemming from Diffie-Hellman exponentiation. In his thesis [Sch12], Schmidt explains the corresponding constructions in detail. He also provides normal form message deduction rules and normal form conditions that additionally allow reasoning about bilinear pairings and multisets of messages.

8.4.3. Additional Constraint-Reduction Rules

In the previous section, we defined the normal form dependency graphs for the equational theory E_{PHS} and formalized their properties. We now give a constraint-reduction relation, which exploits these properties to effectively reason about message deduction. More precisely, we define the constraint-reduction relation $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$, which we use to search for normal form dependency graphs of a protocol P. To make the definition of $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ independent of the concrete definition of normal form dependency graphs, we base it on the following assumptions.

Assumptions about the Normal Form Dependency Graphs

Without loss of generality, let P be a protocol that does not use any $\mathsf{K}^{\uparrow}$- or $\mathsf{K}^{\downarrow}$-facts. Let E be an equational theory that has the finite variant property for the rewriting theory $\mathcal{R}$ and the equational theory $\mathcal{AX}$. Moreover, let ND be a set of multiset rewriting rules. Overloading notation, we call ND the *normal form message deduction rules*. We assume given a definition of the set of *normal form dependency graphs of* P for this choice of ND, $\mathcal{R}$, and $\mathcal{AX}$. We use $ndgraphs_{ND,\mathcal{R},\mathcal{AX}}(P)$ to denote this set of normal form dependency graphs. We define the corresponding set of *normal form traces of* P as

$ntraces_{ND,\mathcal{R},\mathcal{AX}}(P) := \{trace(dg) \mid dg \in ndgraphs_{ND,\mathcal{R},\mathcal{AX}}(P)\}$. We make the following eight assumptions.

A1 The rules IRECV, ISEND, and COERCE from Figure 8.10 are contained in ND.

A2 All rules in ND other than the ISEND rule are of the form $l -[\ a\ ]\to \mathsf{K}^d(m)$ for $d \in \{\uparrow, \downarrow\}$.

A3 All rules in ND with a $\mathsf{K}^\uparrow$-conclusion are of the form $l -[\ \mathsf{K}^\uparrow(m)\ ]\to \mathsf{K}^\uparrow(m)$.

A4 The only rule in ND with a $\mathsf{K}^\downarrow$-conclusion, but no $\mathsf{K}^\downarrow$-premise is the rule IRECV.

A5 $ndgraphs_{ND,\mathcal{R},\mathcal{AX}}(P) \subseteq_{\mathcal{AX}} dgraphs_{\mathcal{AX}}(\lceil P \rceil^{\mathcal{R}}_{\mathcal{AX}} \cup ND)$

A6 Every normal form dependency graph $dg \in ndgraphs_{ND,\mathcal{R},\mathcal{AX}}(P)$ satisfies conditions **N1-4** with respect to $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normalization and equality modulo $\mathcal{AX}$.

A7 For every guarded trace property φ that does not use any $\mathsf{K}^\uparrow$-facts,

$$P \cup MD \vDash^{\forall}_{E} \varphi \quad \text{iff} \quad ntraces_{ND,\mathcal{R},\mathcal{AX}}(P) \vDash^{\forall}_{\mathcal{AX}} \varphi\ .$$

A8 The set of normal form traces $ntraces_{ND,\mathcal{R},\mathcal{AX}}(P)$ is prefix closed.

Note that the definition of the normal form dependency graphs for E_{PHS} given in the previous section satisfies these assumptions for $\mathcal{R} = PHS$ and $\mathcal{AX} = \varnothing$. These assumptions are also satisfied by the definitions of the normal form dependency graphs given in [SMCB12a] and [Sch12]. They therefore provide a solid foundation for defining the constraint-reduction relation $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$.

Assumptions **A1-4** characterize the properties of the normal form message deduction rules that we exploit during constraint solving. Assumption **A5** ensures that normal form dependency graphs are also $(\lceil P \rceil^{\mathcal{R}}_{\mathcal{AX}} \cup ND), \mathcal{AX}$-dependency graphs. Together, the first five assumptions imply the deconstruction-chain property, as formalized in Lemma 5. Assumption **A6** ensures that we can exploit the normal form conditions **N1-4** during constraint solving. Assumption **A7** ensures that we can restrict our search for models of guarded trace properties to normal form dependency graphs, Assumption **A8** ensures that it is sound to use trace induction to reason about properties of the normal form traces. We use this later in Section 8.4.4 to prove the soundness of type assertions.

Normal Form Solutions

We adapt the notions related to constraint solving as follows. A constraint system Γ may additionally contain *deconstruction chain* constraints, written $(i, u) \dashrightarrow (j, v)$. Overloading notation, we say that Γ is *wellformed* if it satisfies the conditions **WF1-4** from Section 8.2.3 modulo $\mathcal{AX}$ and the condition **WF5**.

WF5 For every deconstruction chain $c \dashrightarrow p$ in Γ, there exist terms t and s such that $(c, \mathsf{K}^\downarrow(t)) \in_{\mathcal{AX}} cs(\Gamma)$ and $(p, \mathsf{K}^\downarrow(s)) \in_{\mathcal{AX}} ps(\Gamma)$.

For a structure (dg, θ), we extend the satisfaction relation $\Vdash_{\mathcal{AX}}$ defined in Section 8.2.2 such that

$$(dg, \theta) \Vdash_{\mathcal{AX}} (i, u) \dashrightarrow (j, v) \qquad \text{iff } (\theta(i), u) \dashrightarrow_{dg} (\theta(j), v)\ .$$

We say that (dg, θ) is a *normal form model of* Γ, if dg is a normal form dependency graph and $(dg, \theta) \Vdash_{Ax} \Gamma$. A *normal form solution* of Γ is a normal form dependency graph dg such that there is a valuation θ with $(dg, \theta) \Vdash_{Ax} \Gamma$. We write $nsols_{P,ND,\mathcal{R},\mathcal{AX}}(\Gamma)$ to denote the set of all normal form solutions of Γ. The following theorem formalizes that we can verify validity and satisfiability claims for the protocol P by searching for normal form solutions.

Lemma 6 (Searching for Normal Form Solutions)**.** *For every guarded trace property φ that does not use any $\mathsf{K}^{\uparrow}$-facts, $P \cup MD \models^{\exists}_{E} \varphi$ iff $nsols_{P,ND,\mathcal{R},\mathcal{AX}}(\{\varphi\}) \neq \emptyset$.*

Proof. Follows from Assumption **A7** and the definition of normal form solutions. □

The Constraint-Reduction Relation $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$

In the following, we first define the constraint-reduction relation $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$. Then, we prove its correctness and completeness and show that we can construct a normal form solution from every wellformed, $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$-irreducible constraint system. Finally, we illustrate the use of $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ on an extended example.

Definition The *constraint-reduction relation* $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ is defined in Figure 8.12. The rule $\mathtt{R}_{\mathtt{induct}}$ states that we can use the $\rightsquigarrow^{\mathtt{last}}_{([P]^{\mathcal{R}}_{\mathcal{AX}} \cup \{\textsc{Isend}\}),\mathcal{AX}}$ constraint-reduction relation for solving all constraints except $\mathsf{K}^{\uparrow}$-actions, $\mathsf{K}^{\uparrow}$-premises, $\mathsf{K}^{\downarrow}$-premises, and deconstruction chains. The rule $\mathtt{N1}$ ensures that all rule instances in node constraints are in $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normal form, as required by the normal form condition **N1**.

The four rules $\mathtt{S}_{\mathsf{K}^{\uparrow}_{\blacktriangleright}}$, $\mathtt{S}_{\mathsf{K}^{\uparrow}_{@}}$, $\mathtt{N2}$, and $\mathtt{N3}_{\uparrow}$ work together to reason about messages deduced using construction rules and the Coerce rule. Exploiting **A3**, they use $\mathsf{K}^{\uparrow}$-actions to represent $\mathsf{K}^{\uparrow}$-conclusions whose corresponding rule has not yet been determined using the rule $\mathtt{S}_{\mathsf{K}^{\uparrow}_{@}}$. We use this construction to ensure that the rule $\mathtt{N4}$, which orders $\mathsf{K}^{\uparrow}$-conclusions and $\mathsf{K}^{\downarrow}$-conclusions, is applicable as soon as possible. These four rules work as follows.

The rule $\mathtt{S}_{\mathsf{K}^{\uparrow}_{\blacktriangleright}}$ solves a $\mathsf{K}^{\uparrow}(t) \blacktriangleright_v i$ premise that is not yet being derived before node i. It works by introducing a $\mathsf{K}^{\uparrow}(t)@j$ constraint for a fresh timepoint j together with the constraint that $j \lessdot i$. The introduced action constraint is then either merged with an existing node deriving $\mathsf{K}^{\uparrow}(t)$ using the rule $\mathtt{N3}_{\uparrow}$ or it is solved using the rule $\mathtt{S}_{\mathsf{K}^{\uparrow}_{@}}$. Coerce nodes that derive pairs are pruned using the rule $\mathtt{N2}$. We delay solving $\mathsf{K}^{\uparrow}(v)@i$ actions for $v \in \mathcal{V}_{msg} \cup \mathcal{V}_{pub} \cup PN$, as they can always be solved trivially using an instance of the construction rule for public names.

The rules $\mathtt{N3}_{\downarrow}$, $\mathtt{S}_{\mathsf{K}^{\downarrow}_{\blacktriangleright}}$, and $\mathtt{S}_{\dashrightarrow}$. work together to reason about messages deduced using deconstruction rules. Rule $\mathtt{N3}_{\downarrow}$ merges duplicate derivations of the same $\mathsf{K}^{\downarrow}$-conclusion. Rule $\mathtt{S}_{\mathsf{K}^{\downarrow}_{\blacktriangleright}}$ solves an open $\mathsf{K}^{\downarrow}$-premise by introducing an arbitrary instance of the Irecv rule from which the adversary deduced the $\mathsf{K}^{\downarrow}$-premise using a chain of deconstruction rules. Such an instance of the Irecv rule must exist due to the deconstruction-chain property. The possible messages sent by the protocol from which the chain could start are enumerated using the rule $\mathtt{DG}_{\blacktriangleright}$ on the newly introduced $\mathsf{Out}(t)$ premise. Rule $\mathtt{S}_{\dashrightarrow}$ refines a deconstruction chain by unfolding its definition by one step. It makes a case distinction on whether a deconstruction chain consists of (1) a single edge or (2) an

Rule $\mathtt{S}_{\mathsf{K}^{\uparrow}@}$ uses the definition $kn^{\uparrow}_{<i}(\Gamma) := \{m \mid \exists j.\ j \lessdot_\Gamma i \wedge (\mathsf{K}^{\uparrow}(m)@j) \in as(\Gamma) \cup \Gamma\}$, which denotes the set of messages constructed before timepoint i. We write $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ as $\rightsquigarrow$ and the empty set of constraint systems $\varnothing$ as $\bot$.

$\mathtt{R}_{\mathtt{induct}}$: $\Gamma \rightsquigarrow \Gamma_1 \parallel \ldots \parallel \Gamma_n$ if $\Gamma \rightsquigarrow^{\mathtt{induct}}_{([P]^{\mathcal{R}}_{\mathcal{AX}} \cup \{\textsc{IsEnd}\}),\mathcal{AX}} \Gamma_1 \parallel \ldots \parallel \Gamma_n$, except for solving $\mathsf{K}^{\uparrow}$-actions, $\mathsf{K}^{\uparrow}$-premises, and $\mathsf{K}^{\downarrow}$-premises

$\mathtt{N1}$: $\Gamma \rightsquigarrow \bot$ if $(i : ri) \in \Gamma$ and ri not $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normal

$\mathtt{S}_{\mathsf{K}^{\uparrow}\blacktriangleright}$: $\Gamma \rightsquigarrow (\mathsf{K}^{\uparrow}(t)@j, j \lessdot i, \Gamma)$
if $\underline{(\mathsf{K}^{\uparrow}(t) \blacktriangleright_v i)} \in \Gamma$ and j fresh; $\boxed{t \notin_{Ax} kn^{\uparrow}_{<i}(\Gamma)}$

$\mathtt{S}_{\mathsf{K}^{\uparrow}@}$: $\Gamma \rightsquigarrow \parallel_{ri \in ND} \parallel_{f \in acts(ri)} (i : ri, \mathsf{K}^{\uparrow}(t) \approx f, \Gamma)$
if $\underline{(\mathsf{K}^{\uparrow}(t)@i)} \in \Gamma$; $\boxed{\mathsf{K}^{\uparrow}(t)@i \notin_{Ax} as(\Gamma) \text{ and } t \notin \mathcal{V}_{msg} \cup \mathcal{V}_{pub} \cup PN}$

$\mathtt{S}_{\mathsf{K}^{\downarrow}\blacktriangleright}$: $\Gamma \rightsquigarrow (i : \mathsf{Out}(y) \dashv\!\!\mid\!\!\mapsto \mathsf{K}^{\downarrow}(y), (i,1) \dashrightarrow (j,v), \Gamma)$
if $\underline{(\mathsf{K}^{\downarrow}(t) \blacktriangleright_v j)} \in \Gamma$ and i, y fresh; $\boxed{\text{not } \exists c.\ (c \twoheadrightarrow (j,v)) \in \Gamma \vee (c \dashrightarrow (j,v)) \in \Gamma}$

$\mathtt{S}_{\dashrightarrow}$: $\Gamma \rightsquigarrow (c \twoheadrightarrow p, \Gamma) \parallel \parallel_{ri \in ND} \parallel_{u \in idx(prems(ri))} (i : ri, c \twoheadrightarrow (i,u), (i,1) \dashrightarrow p, \Gamma)$
if $\underline{c \dashrightarrow p \in \Gamma}$ and i fresh; $\boxed{\forall p'.\ (c \twoheadrightarrow p') \notin \Gamma \text{ and } \forall x \in \mathcal{V}_{msg}.\ (c, \mathsf{K}^{\downarrow}(x)) \notin cs(\Gamma)}$

$\mathtt{N2}$: $\Gamma \rightsquigarrow \bot$ if $(i : \mathsf{K}^{\downarrow}(\langle t_1, t_2 \rangle) \dashv\!\!\mid\!\!\mapsto \mathsf{K}^{\uparrow}(\langle t_1, t_2 \rangle)) \in_{Ax} \Gamma$

$\mathtt{N3}_{\uparrow}$: $\Gamma \rightsquigarrow (i \doteq j, \Gamma)$ if $\{\mathsf{K}^{\uparrow}(t)@i, \mathsf{K}^{\uparrow}(t)@j\} \subseteq_{Ax} as(\Gamma) \cup \Gamma$; $\boxed{i \neq j}$

$\mathtt{N3}_{\downarrow}$: $\Gamma \rightsquigarrow (i \doteq j, \Gamma)$ if $\{((i,1), \mathsf{K}^{\downarrow}(t)), ((j,1), \mathsf{K}^{\downarrow}(t))\} \subseteq_{Ax} cs(\Gamma)$; $\boxed{i \neq j}$

$\mathtt{N4}$: $\Gamma \rightsquigarrow (i \lessdot j, \Gamma)$ if $((i,1), \mathsf{K}^{\downarrow}(t)) \in_{Ax} cs(\Gamma)$ and $\mathsf{K}^{\uparrow}(t)@j \in_{Ax} as(\Gamma) \cup \Gamma$; $\boxed{\text{not } i \lessdot_\Gamma j}$

Figure 8.12.: Rules defining the constraint-reduction relation $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$

edge to a rule ri with a $\mathsf{K}^{\downarrow}$-premise and a further deconstruction chain starting from the $\mathsf{K}^{\downarrow}$-conclusion of ri. We disallow refining chains that start from a message variable v, as this would lead to non-termination.

Note that later constraint-reduction steps will often instantiate this message variable v and allow us to finish refining the chain. This is always the case for protocols that check the types of encrypted or signed data. We can model such type checks using sorts. This is sound for any implementation that ensures that all bitstrings representing terms of one sort are different with overwhelming probability from all bitstrings representing terms of any other incomparable sort. To reason about protocols that receive arbitrary messages as cleartext, we use the rule `N4`, as explained in Example 32. We use type assertions as explained in Section 8.4.4, to reason about protocols that blindly forward encrypted messages; e.g., a protocol containing a rule like $[\mathsf{In}(\mathrm{enc}(x, k_1))] \dashv\!\!\mid\!\!\mapsto [\mathsf{Out}(\mathrm{enc}(x, k_2))]$ for $x \in \mathcal{V}_{msg}$.

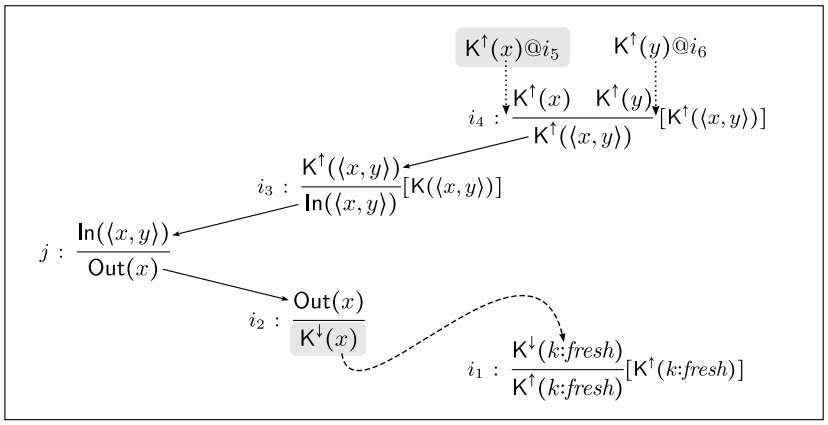


Figure 8.13.: A contradictory constraint system pruned by rule N4.

Example 32 (Forwarding of Messages via Protocol Rules)**.** Consider solving the premise $\mathsf{K}^{\uparrow}(k{:}fresh)$ in the context of a protocol that contains the multiset rewriting rule $[\mathsf{In}(\langle x, y\rangle)] -[]\rightarrow [\mathsf{Out}(x)]$. One of the cases that we must consider is captured by the constraint system depicted in Figure 8.13. It states that the adversary might deduce k using a deconstruction chain starting from the message x sent by node j. This constraint system is contradictory because all its normal form solutions violate condition **N4**. We show that this constraint system is contradictory using rule $\mathtt{DG}_{\lessdot}$ after we used rule N4 to derive $i_5 \lessdot i_2$. ♠

Formal Properties We now prove the properties, which formalize the suitability of the constraint-reduction relation $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ for constraint solving. More precisely, we show that this constraint-reduction relation is wellformed, correct and complete with respect to normal form solutions, and that we can extract a normal form solution from every wellformed, $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$-irreducible constraint system.

Theorem 11 (Wellformedness, Correctness, and Completeness of $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$)**.** *For every constraint-reduction step* $\Gamma \rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}} \mathbf{\Gamma'}$ *of a constraint system to a set of constraint systems* $\mathbf{\Gamma'}$*, it holds that*

(i) if Γ *is wellformed, then all constraint systems in* $\mathbf{\Gamma'}$ *are wellformed and*

(ii) $nsols_{P,ND,\mathcal{R},\mathcal{AX}}(\Gamma) =_{\mathcal{AX}} \bigcup_{\Gamma' \in \mathbf{\Gamma'}} nsols_{P,ND,\mathcal{R},\mathcal{AX}}(\Gamma')$

provided that the assumptions **A1-7** *hold.*

Proof (sketch of the complete proof on page 185). The wellformedness of $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ follows from its definition and **WF1-5**. The correctness and completeness of the rule $\mathtt{R}_{\mathtt{last}}$ follows from Theorem 7 and the assumptions **A1,2,5**. The other rules are trivially correct, as they only add further constraints to Γ. The completeness of the other rules follows from the assumptions **A1-7**, which characterize the structure of the normal form message deduction rules and the normal form conditions. □

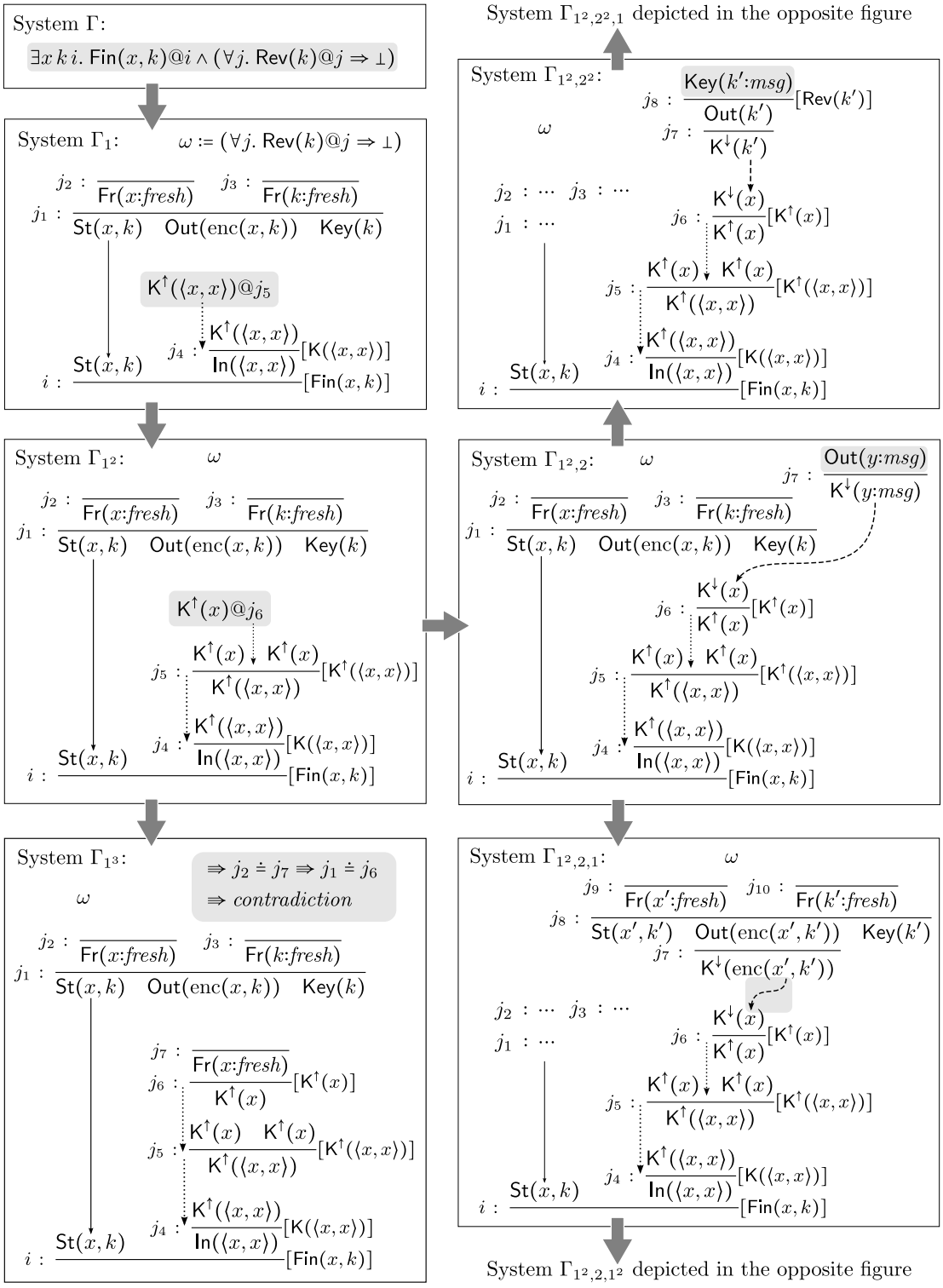


Figure 8.14.: First half of the constraint systems from Example 33, where we show that $nsols_{P_{Ex},ND,PHS,\varnothing}(\Gamma) = \varnothing$.

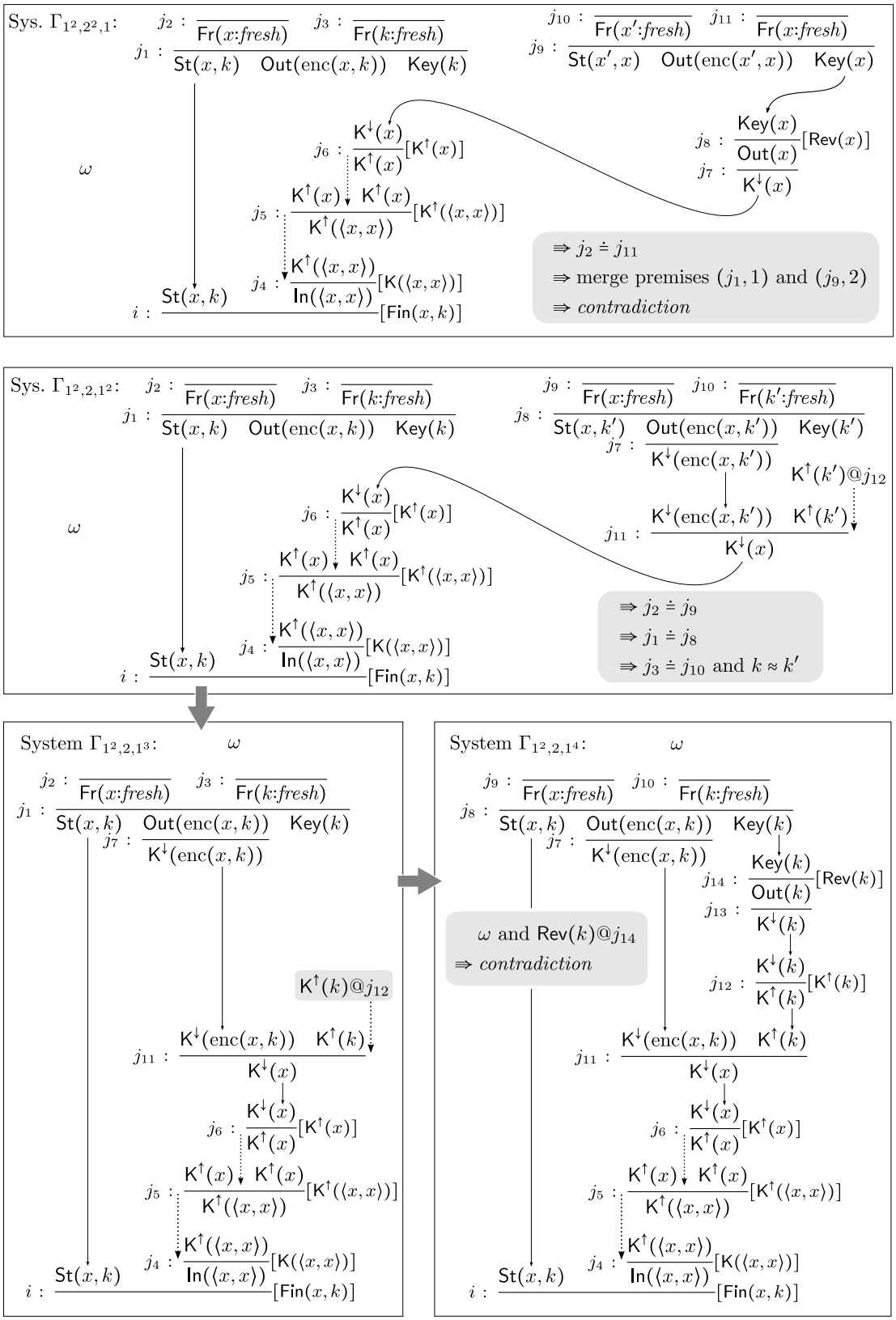


Figure 8.15.: Second half of the constraint systems from Example 33.

Theorem 12 (Solution Construction for $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$)**.** *We can construct a normal form solution for every wellformed and* $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$*-irreducible constraint system* Γ*, provided that the assumption* **A1-7** *hold.*

Proof (sketch of the complete proof on page 187). The main idea is to convert Γ to a $\rightsquigarrow^{\mathtt{last}}_{(\lceil P\rceil^{\mathcal{R}}_{\mathcal{AX}}\cup ND),\mathcal{AX}}$-irreducible constraint system Γ', such that we can use the solution construction method from the proof of Theorem 8. This conversion solves remaining $\mathsf{K}^{\uparrow}(x)@i$ actions with $x \in \mathcal{V}_{msg} \cup \mathcal{V}_{pub} \cup PN$, using the construction rule for public names, inserts edges for all $\mathsf{K}^{\uparrow}$-premises, and removes all deconstruction chains. The proof that the solution constructed from the resulting constraint system Γ' is a normal form solution of Γ exploits the $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$-irreducibility of Γ. □

Extended Example In the following example, we illustrate the use of $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ for reasoning about message deduction. We do this by revisiting Example 28, which we used to illustrate the insufficiency of $\rightsquigarrow^{\mathtt{last}}_{R,E}$ for reasoning about message deduction.

Example 33 (Using $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ for Constraint Solving)**.** We prove the validity claim from Example 28 on page 110,

$$P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\forall}_{E_{PHS}} \forall x\, k\, i.\ \mathsf{Fin}(x,k)@i \Rightarrow \exists j.\ \mathsf{Rev}(k)@j\ ,$$

using the constraint-solving relation $\rightsquigarrow^{\mathtt{msg}}_{P_{Ex},ND,PHS,\emptyset}$ where ND are the normal form message deduction rules for E_{PHS} defined in Section 8.4.2.

Due to Lemma 6 on page 121, this claim holds if and only if the constraint system

$$\Gamma := \{\ \exists x\, k\, i.\ \mathsf{Fin}(x,k)@i \wedge (\forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot)\ \}$$

has no normal form solutions; i.e., $nsols_{P_{Ex},ND,PHS,\emptyset}(\Gamma) = \emptyset$. We prove this using the constraint-solving relation $\rightsquigarrow^{\mathtt{msg}}_{P_{Ex},ND,PHS,\emptyset}$. We depict the constraint systems encountered in this proof in figures 8.14 and 8.15. We explain the corresponding constraint-reduction steps in the following paragraphs.

We start with System Γ and first solve all constraints other than message deduction constraints. We then solve the occurring $\mathsf{K}^{\uparrow}$-premise using the rule $\mathsf{s}_{\mathsf{K}^{\uparrow}_{\blacktriangleright}}$. The resulting system is System Γ_1. In its depiction, we use the following additional notation. We annotate at least one occurrence of every non-temporal and non-message variable with its sort. We abbreviate edges between Fr- and In-facts using inference rule notation. We depict the ordering constraint $j_5 \lessdot j_4$ using a dotted arrow from the action $\mathsf{K}^{\uparrow}(\langle x, x\rangle)@j_5$ to the node j_4.

We proceed by solving the $\mathsf{K}^{\uparrow}(\langle x, x\rangle)@j_5$ action in System Γ_1 using the rule $\mathsf{s}_{\mathsf{K}^{\uparrow}_{@}}$. There are two resulting cases: one where this action is provided by the Coerce rule and one where this action is provided by the pair construction rule. We immediately reduce the first case to a contradiction using rule `N2`, as pairs are never deduced using the Coerce rule. The second case is formalized by System Γ_{1^2}. Note that we always eagerly apply the $\mathsf{s}_{\mathsf{K}^{\uparrow}_{\blacktriangleright}}$ rule to solve $\mathsf{K}^{\uparrow}$-premise constraints.

Our next step is to solve the $\mathsf{K}^{\uparrow}(x)@j_6$ action in System Γ_{1^2} using rule $\mathsf{s}_{\mathsf{K}^{\uparrow}_{@}}$. There are two resulting cases, formalized by systems Γ_{1^3} and $\Gamma_{1^2,2}$. System Γ_{1^3} denotes the case that the adversary might have generated the fresh name denoted by the variable $x{:}\mathit{fresh}$

by himself. This is not possible, as it conflicts with the $\mathsf{Fr}(x{:}\mathit{fresh})$ premise of node j_1. We formalize this by reducing System Γ_{1^3} to the empty set of constraint systems using the rules $\mathtt{DG}_{\mathtt{out}}$, $\mathtt{DG}_{\mathtt{Fr}}$, and $\mathtt{DG}_{\mathtt{lbl}}$.

System $\Gamma_{1^2,2}$ denotes the case that the adversary might deduce $\mathsf{K}^{\uparrow}(x)$ using the Coerce message deduction rule. The premise $\mathsf{K}^{\downarrow}(x) \blacktriangleright_1 j_6$ must be deduced using a chain of deconstruction rules starting from some message sent by the protocol. Hence, we also applied rule $\mathtt{S}_{\mathsf{K}^{\downarrow}\blacktriangleright}$, which introduced node j_7 and the deconstruction chain $(j_7, 1) \dashrightarrow (j_6, 1)$ in System $\Gamma_{1^2,2}$. Note that we cannot yet refine this deconstruction chain using the $\mathtt{S}_{\dashrightarrow}$ rule, as this chain starts from a message variable. We must first enumerate the messages sent by the protocol by applying the rule $\mathtt{DG}_{\blacktriangleright}$ to the premise $j_7 \blacktriangleright_1 \mathsf{Out}(y{:}\mathit{msg})$. There are two resulting cases, formalized by systems $\Gamma_{1^2,2,1}$ and $\Gamma_{1^2,2^2}$. In the depiction of both systems, we suppress the copied node and edge constraints on the nodes j_1, j_2, and j_3 to save space. We also abbreviate edges between Out-facts using inference rule notation. System $\Gamma_{1^2,2,1}$ denotes the case that y might be some encryption $\mathrm{enc}(x', k',)$ sent by some instance of the first protocol rule. System $\Gamma_{1^2,2^2}$ denotes the case that y might be some key k' revealed by the adversary using some instance of the key-reveal rule. We conclude the proof by reducing both of these systems to the empty set of constraint systems as follows.

For System $\Gamma_{1^2,2,1}$, we proceed by refining the deconstruction chain $(j_7, 1) \dashrightarrow (j_6, 1)$ twice using rule $\mathtt{S}_{\dashrightarrow}$. We thereby obtain System $\Gamma_{1^2,2,1^2}$ where x and x' are identified and the deconstruction chain has been replaced with the edges to and from the deconstruction rule for decryption j_{11}. This is the only way to refine this chain, as all other deconstruction rules lead to cases with edges between non-unifiable facts. The uniqueness of Fresh instances leads to the depicted chain of implications and results in System $\Gamma_{1^2,2,1^3}$. The only remaining unsolved constraint in this system is the $\mathsf{K}^{\uparrow}(k)@j_{12}$ action. The constraint-reduction steps required to solve this constraint are similar to the ones used to solve the $\mathsf{K}^{\uparrow}(x)@j_6$ action in System Γ_{1^2}. System $\Gamma_{1^2,2,1^4}$ is the only one of the resulting constraint systems that is not trivially contradictory due to the uniqueness of Fresh instances. In fact, this system could be instantiated to a normal form dependency graph, if it were not for the trace formula $\omega = (\forall j.\ \neg(\mathsf{Rev}(k)@j) \vee \bot)$. Since System $\Gamma_{1^2,2,1^4}$ contains node j_{14} with a $\mathsf{Rev}(k)$ action, we can use the rules $\mathtt{S}_{\neg,@}$ and $\mathtt{S}_{\bot}$ to reduce this system to the empty set of constraint systems.

For System $\Gamma_{1^2,2^2}$, we proceed by first solving the $\mathsf{Key}(k'{:}\mathit{msg}) \blacktriangleright_1 j_8$ premise using the rule $\mathtt{DG}_{\blacktriangleright}$ and then refining the deconstruction chain $(j_7, 1) \dashrightarrow (j_6, 1)$ using the rule $\mathtt{S}_{\dashrightarrow}$. The only resulting system is System $\Gamma_{1^2,2^2,1}$, where x and k' have been unified and the deconstruction chain has been replaced with the edge $(j_7, 1) \rightarrowtail (j_6, 1)$. This system contradicts the uniqueness of the Fresh instances and can therefore be reduced to the empty set of constraint systems using the rules $\mathtt{DG}_{\mathtt{out}}$, $\mathtt{DG}_{\mathtt{Fr}}$, and $\mathtt{DG}_{\mathtt{lbl}}$. This concludes the proof that $\mathit{nsols}_{P_{Ex},ND,PHS,\varnothing}(\Gamma) = \varnothing$ and therefore

$$P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\forall}_{E_{PHS}} \forall x\, k\, i.\ \mathsf{Fin}(x, k)@i \Rightarrow \exists j.\ \mathsf{Rev}(k)@j\ .$$

♠

The above example illustrates the use of $\rightsquigarrow^{\mathtt{msg}}_{ND,PHS,\varnothing}$ for reasoning about message deduction modulo the equational theory E_{PHS}. Reasoning about more complicated equational theories, protocols, and properties works analogously. However, managing

the intermediate constraint systems on paper becomes tedious and does not scale well. We solve this problem in Chapter 9, by implementing the Tamarin prover, which mechanizes the use of our constraint-reduction rules. This prover provides an interactive GUI, which displays the constraint systems using a graphical notation similar to the one used in this thesis. Moreover, it automatically executes the trivial constraint-reduction steps, but lets the user interactively choose the non-trivial steps, e.g., selecting the next message deduction constraint to solve. This enables the user to more easily gain intuition about the protocol and the property of interest by interactively constructing its correctness proofs.

8.4.4. Type Assertions

One of the key steps in a validity proof based on the constraint-reduction relation $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ is to *completely* refine the deconstruction chains occurring in the intermediate constraint systems. We cannot immediately achieve this for protocols that send messages from which the content of a message variable v could be extracted. The problem is that, in general, the variable v could be instantiated with any message. Thus without any further information there are in principle infinitely many ways to refine a deconstruction chain starting from this variable.

In Part I, we already recognized this problem for decryption-chain reasoning and provided a solution. This solution is based on type assertions that characterize the instantiations of all message variables of a protocol such that we can finitely enumerate all decryption-chains starting from their contents. We now adapt this solution to our more general verification theory. Thanks to its generality, we require no additional formalism.

In the following, we show that we can formalize type assertions as invariants of the normal form traces of the protocol of interest. We also show that we can prove the validity of type assertions using trace induction. We explain this first on an example. Afterwards, we describe how to construct type assertions in general.

Example 34 (Using Typing Assertions)**.** Consider the following artificial protocol.

$$P_{ty} := \left\{ \begin{array}{l} \textsc{SetupKey}\ \dfrac{\mathsf{Fr}(k)}{!\mathsf{Key}(k)}, \qquad \textsc{RevealKey}\ \dfrac{!\mathsf{Key}(k)}{\mathsf{Out}(k)}[\mathsf{Rev}(k)], \\ \textsc{Init}\ \dfrac{!\mathsf{Key}(k) \quad \mathsf{Fr}(s) \quad \mathsf{Fr}(p)}{\mathsf{Out}(\mathrm{enc}(\langle s,p\rangle,k))}[\,\mathsf{Out1}(\mathrm{enc}(\langle s,p\rangle,k))\,], \\ \textsc{Resp}\ \dfrac{!\mathsf{Key}(k) \quad \mathsf{In}(\mathrm{enc}(\langle s,p\rangle,k))}{\mathsf{Out}(p)}[\mathsf{Sec}(s,k),\ \mathsf{In1}(p,\mathrm{enc}(\langle s,p\rangle,k))] \end{array} \right\}$$

The SetupKey rule sets up encryption keys, which can be revealed by the adversary using the RevealKey rule. Recall that we use ! to mark the !Key fact as a persistent fact. The Init and Resp rules model an initiator and a responder that use these encryption keys to securely exchange a secret value s and a value p to be published over the public channel controlled by the adversary. Upon receiving an encrypted message of the form $\mathrm{enc}(\langle s,p\rangle,k)$, the responder claims that s is secret provided that k has not been revealed

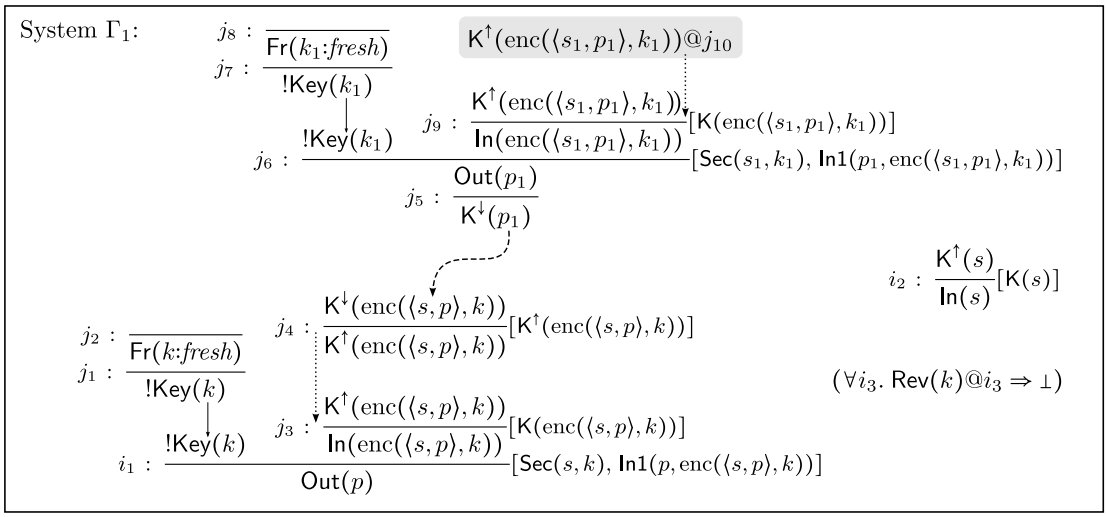


Figure 8.16.: System Γ_1 from Example 34 illustrating the need for type assertions. The variables s, p, s_1, and p_1 are in $\mathcal{V}_{msg}$.

and publishes the value p. We explain the two actions with the gray background later, as we use them only to formalize our type assertion.

Formally, the secrecy claim that we would like to prove is

$$P_{ty} \cup MD_{\Sigma_{PHS}} \vDash^{\forall}_{E_{PHS}} \forall s\ k\ i_1\ i_2.\ \mathsf{Sec}(s,k)@i_1 \wedge \mathsf{K}(s)@i_2 \Rightarrow \exists i_3.\ \mathsf{Rev}(k)@i_3\ .$$

We prove this claim by showing that the constraint system

$$\{\mathsf{Sec}(s,k)@i_1,\ \mathsf{K}(s)@i_2,\ (\forall i_3.\ \mathsf{Rev}(k)@i_3 \Rightarrow \bot)\}$$

has no normal form solutions. We attempt to show this using the constraint-reduction relation $\rightsquigarrow^{\mathtt{msg}}_{P_{ty},ND,PHS,\varnothing}$ analogously to Example 33. As we will see, this direct attempt fails because we fail to reason about what messages can be deconstructed from the content of the message variable p sent by an instance of the Resp rule. We will fix this problem by proving a protocol-specific invariant, which characterizes the possible contents of p such that we can reason about what messages can be extracted from them.

System Γ_1 depicted in Figure 8.16 illustrates the problem of reasoning about what messages can be deconstructed from the content of the message variable p. It results from solving the actions $\mathsf{Sec}(s,k)@i_1$ and $\mathsf{K}(s)@i_2$ and then trying to determine the origin of the encryption $\mathsf{K}^{\uparrow}(\mathrm{enc}(\langle s,p\rangle,k))$ received by Node i_1. System Γ_1 formalizes the case that the encryption $\mathsf{K}^{\uparrow}(\mathrm{enc}(\langle s,p\rangle,k))$ was deconstructed from the message p_1 that is sent to the adversary by some instance of the Resp rule, i.e., Node j_6. Note that System Γ_1 has no normal form solutions, as $\mathsf{K}^{\uparrow}(\mathrm{enc}(\langle s_1,p_1\rangle,k_1))$ was either (1) constructed by the adversary or (2) sent by an instance of the Init rule. Both of these cases are contradictory. The first case implies that the adversary derived $\mathsf{K}^{\uparrow}(p_1)$ before $\mathsf{K}^{\downarrow}(p_1)$, which contradicts condition **N4**. The second case implies that p_1 contains a fresh name. This contradicts the deconstruction chain $(j_5,1) \dashrightarrow (j_4,1)$, as deconstruction chains built from instances of the normal form message deduction rules ND cannot start from a fresh name.

Nevertheless, we fail to reduce system Γ_1 to the empty set of constraint systems. We cannot refine the deconstruction chain $(j_5, 1) \dashrightarrow (j_4, 1)$, as p_1 is a message variable. We can also not solve the $\mathsf{K}^{\uparrow}(s) \blacktriangleright_1 i_2$ premise, as s is a message variable. We are therefore left with trying to determine the origin of the encryption $\mathsf{K}^{\uparrow}(\mathrm{enc}(\langle s_1, p_1\rangle, k_1))$. This will again result in a case with another instance of the Resp rule and an outgoing deconstruction chain that we cannot refine further. We thus fail to show that $nsols_{P_{ty},ND,PHS,\varnothing}(\Gamma_1) = \varnothing$ using the constraint-reduction relation $\rightsquigarrow^{\mathtt{msg}}_{P_{ty},ND,PHS,\varnothing}$.

We fix this problem by proving the invariant

$$\psi_{ty} := \forall p\ m\ i.\ \mathsf{In1}(p, m)@i \Rightarrow (\exists j.\ \mathsf{K}^{\uparrow}(p)@j \wedge j \lessdot i) \vee (\exists j.\ \mathsf{Out1}(m)@j)\ .$$

of the normal form traces of P_{ty}. We call ψ_{ty} a *type assertion* because it characterizes the possible instantiations of the message variable p in the Resp rule of P_{ty}. The intuition behind this type assertion is the following. The action $\mathsf{In1}(p, m)$ denotes that the message variable p was instantiated upon the receipt of the message m. The action $\mathsf{Out1}(m)$ denotes that the message m was sent and is intended to be received by rules marked with an $\mathsf{In1}$-action. In the context of the P_{ty} protocol, ψ_{ty} thus states the following. Whenever the variable p is instantiated upon the receipt of the encryption $\mathsf{K}^{\uparrow}(\mathrm{enc}(\langle s, p\rangle, k))$ by an instance of the Resp rule, then either

(i) the content of p was deduced by the adversary beforehand (because he constructed the encryption by himself) or

(ii) the encryption was sent by an instance of the Init rule (and p thus contains a fresh name).

We exploit ψ_{ty} to reduce Γ_1 to the empty set of constraint systems as follows. Assume that we have proven that ψ_{ty} is an invariant of the normal form traces of P_{ty}, i.e., $ntraces_{ND,PHS,\varnothing}(P_{ty}) \vDash^{\forall}_{\varnothing} \psi_{ty}$. We can thus add ψ_{ty} to the System Γ_1 without changing its normal form solutions; i.e.,

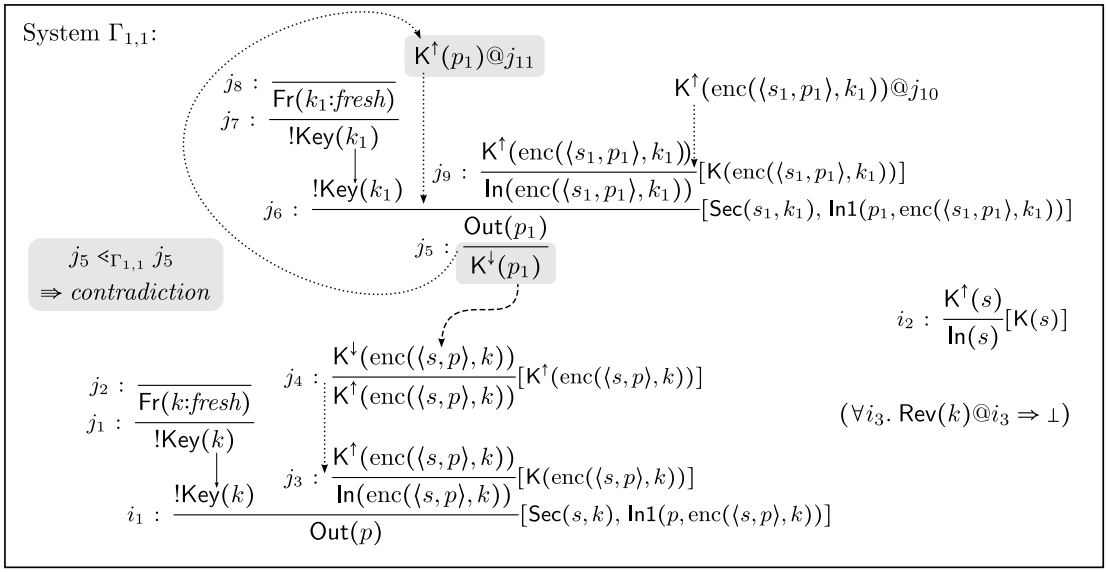
$$nsols_{P_{ty},ND,PHS,\varnothing}(\Gamma_1) =_{\varnothing} nsols_{P_{ty},ND,PHS,\varnothing}(\Gamma_1 \cup \{\psi_{ty}\})\ .$$

Hence, we can add the disjunction

$$(\exists j.\ \mathsf{K}^{\uparrow}(p_1)@j \wedge j \lessdot j_6) \vee (\exists j.\ \mathsf{Out1}(\mathrm{enc}(\langle s_1, p_1\rangle, k_1))@j)$$

to the system $\Gamma_1 \cup \{\psi_{ty}\}$ by instantiating the universally quantified variables p, m, and i in ψ_{ty} with p_1, $\mathrm{enc}(\langle p_1, m_1\rangle, k_1)$, and j_6. This disjunction allows us to reduce System Γ_1 to the empty set of constraint systems as follows. We first perform a case split on the disjunction and then solve the existential quantifiers and the conjunctions. The two resulting constraint systems are depicted in Figure 8.17. System $\Gamma_{1,1}$ formalizes the case that the intruder constructed the encryption. We reduce it to the empty set of constraint systems by applying rule `M4` followed by rule $\mathtt{DG}_{\lessdot}$. System $\Gamma_{1,2}$ formalizes the case that some instance of the Init rule (Node j_{11}) sent the encryption. We reduce it to the empty set of constraint systems by applying rule $\mathtt{S}_{\dashrightarrow}$ followed by rules $\mathtt{DG}_{\rightarrowtail}$ and $\mathtt{S}_{\approx}$.

We prove that $ntraces_{ND,PHS,\varnothing}(P_{ty}) \vDash^{\forall}_{\varnothing} \psi_{ty}$ using trace induction. This is sound, as the normal form traces are prefix closed because of Assumption **A8**. After the application of trace induction, it remains to show that $\mathrm{BC}(\psi_{ty})$ and $\mathrm{IH}(\psi_{ty}) \Rightarrow \psi_{ty}$ hold for all normal



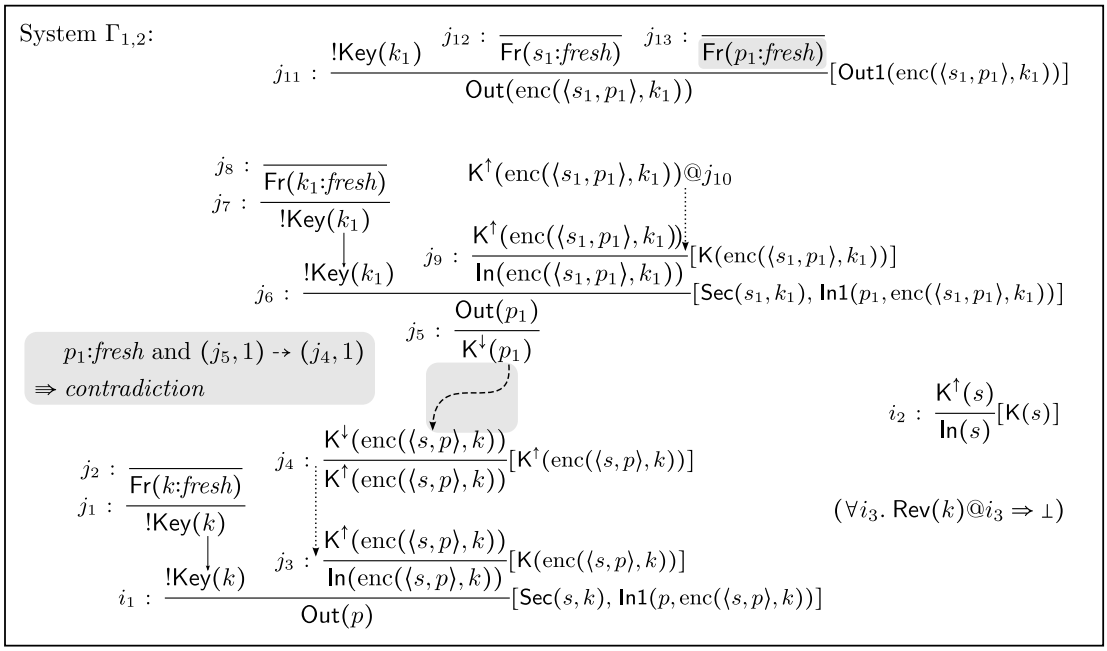


Figure 8.17.: Constraint systems resulting from the use of the type assertion ψ_{ty}.

form traces of P_{ty}. This is trivial for $\text{BC}(\psi_{ty})$. For $\text{IH}(\psi_{ty}) \Rightarrow \psi_{ty}$, we expand $\text{IH}(\psi_{ty})$ and show that

$$\begin{aligned} \mathit{ntraces}_{ND,PHS,\emptyset}(P_{ty}) \vDash^{\forall}_{\emptyset} (\forall p\ m\ i.\ \mathsf{In1}(p,m)@i \Rightarrow\ & \text{last}(i) \\ & \vee (\exists j.\ \mathsf{K}^{\uparrow}(p)@j \wedge j \lessdot i \wedge \neg\text{last}(j)) \\ & \vee (\exists j.\ \mathsf{Out1}(m)@j \wedge \neg\text{last}(j)) \\ ) \Rightarrow \psi_{ty} & \end{aligned}$$

using the constraint-reduction relation $\rightsquigarrow^{\mathtt{msg}}_{P_{ty},ND,PHS,\emptyset}$. This proof is straightforward and we therefore do not detail it here. The key step is to exploit the induction hypothesis $\text{IH}(\psi_{ty})$ analogously to the reduction of System Γ_1 to the empty set of traces. One can inspect this proof in the GUI of the Tamarin prover by loading the `Minimal_Typing_Example.spthy` file, which is distributed with the Tamarin prover. ♠

Constructing Type Assertions

The goal of a type assertion is to enable the complete refinement of all deconstruction chains that may occur in a proof of a property of a protocol P. In general, a type assertion for the protocol P must therefore characterize all possible instantiations of every message variable v whose content is extractable from a message sent by a multiset rewriting rule in P. We construct such a type assertion roughly as follows.

We assume without loss of generality that the variables of different rules in P are disjoint. We add an action $\mathsf{In}_v(v,m)$ to every rule containing a message variable v that is instantiated upon the receipt of the message m. We also add a corresponding $\mathsf{Out}_v(m')$ action to every rule of P that sends a message m' unifying with the received message m. The type assertion for P is then

$$\bigwedge_{v \in vars(P) \cap \mathcal{V}_{msg}} (\forall v\ m\ i.\ \mathsf{In}_v(v,m) \Rightarrow (\exists j.\ \mathsf{K}^{\uparrow}(v)@j \wedge j \lessdot i) \vee (\exists j.\ \mathsf{Out}_v(m)))\ .$$

We prove this type assertion using trace induction and exploit it as in the previous example. Informally, this works if the following two assumptions are valid.

(i) If the adversary constructs the received message m by himself, then he must have deduced the content of the instantiated variable v beforehand.

(ii) The unification of m' to m instantiates the message variable v such that the deconstruction chains starting from it can be completely refined.

For some protocols, we must thus weaken the type assertion such that we consider only the cryptographic components of the sent and received messages instead of the complete messages. More precisely, instead of considering the received message m, we consider the cryptographic component c immediately containing the message variable v of interest. This ensures Assumption (i). Instead of considering the unifying sent messages m', we then consider the unifying cryptographic components c' occurring in sent messages. This adaption is required to account for the fact that the adversary can construct matching messages from partially deconstructed sent messages.

This construction suffices to reason about many protocols. In particular, it suffices to reason about all the case studies from Part I, as it generalizes the construction of type assertions for decryption-chain reasoning. Moreover, we have also used it in our case studies about cryptographic APIs and stream protocols. We did not require type assertions to reason about Diffie-Hellman based AKE protocols because they receive all their message components as cleartext. See Section 9.3, for more information.

9. Automated Protocol Analysis

In this chapter, we explain how we automate the analysis of security protocols using our constraint-reduction rules. Our automation is based on the notion of a constraint solving algorithm, which we define in Section 9.1. On a high level, a constraint solving algorithm consists of two components: (1) a constraint-reduction strategy, which gives rise to a possibly infinite search tree, and (2) a search strategy, which is used to search this tree for a solved constraint system. We implement a constraint solving algorithm based on our constraint-reduction rules in a tool, called the Tamarin prover. The user interface of the Tamarin prover provides both an automatic and an interactive mode. The automatic mode uses iterative deepening as the search strategy and a heuristic to select the next constraint-reduction rule to apply. The interactive mode uses a Graphical User Interface (GUI) to allow the user to determine interactively both the search and the constraint-reduction strategy. We typically use the interactive mode to understand automatically generated proofs and to investigate non-termination issues. Our GUI supports this task further by visualizing the intermediate constraint systems analogously to the examples depicted in this thesis. We discuss these two modes and the implementation of the Tamarin prover in Section 9.2. Afterwards, in Section 9.3, we report on the experimental results that we obtained using Tamarin's automatic mode to analyze a wide range of case studies. Finally, we discuss the strengths and weaknesses of our approach to the automated analysis of security protocols in Section 9.4.

9.1. Constraint Solving Algorithms

A *constraint-reduction strategy* is a partial function r from constraint systems to finite sets of constraint systems satisfying the following two conditions.

(i) The constraint-reduction relation $\{(\Gamma, r(\Gamma)) \mid \Gamma \in dom(r)\}$ is well-formed, correct, and complete.

(ii) Every well-formed constraint system not in the domain of r has a non-empty set of solutions.

Intuitively, r reduces constraint systems in its domain to finite sets of constraint systems covering the same set of solutions, whereas it marks constraint systems not in its domain as solved.

We represent the possibly infinite-depth *search trees* using the coinductive datatype

$$\mathit{SearchTree} ::= \mathsf{Solved}(\mathcal{P}(\mathit{Constraint})) \mid \mathsf{Reduce}(\mathcal{P}(\mathit{Constraint}) \times \mathcal{P}_{\mathit{fin}}(\mathit{SearchTree})) \ .$$

We define the search tree corresponding to a constraint-reduction strategy r as

$$\mathit{searchTree}_r(\Gamma) := \begin{cases} \mathsf{Solved}(\Gamma) & \text{if } \Gamma \notin dom(r) \\ \mathsf{Reduce}(\Gamma, \{\mathit{searchTree}_r(\Gamma') \mid \Gamma' \in r(\Gamma)\}) & \text{if } \Gamma \in dom(r) \end{cases} .$$

A *constraint solving algorithm* checks whether a constraint system Γ has a solution by searching for a Solved-node in $\mathit{searchTree}_r(\Gamma)$. Depending on the constraint-reduction strategy r, the constraint system Γ, and the employed search strategy, this check might not terminate. However, if it terminates then we have either

(i) a proof that Γ has no solution, i.e., the finite and complete search tree without a Solved-node, or

(ii) a proof that Γ has a solution, i.e., the constraint system whose set of solutions is non-empty *and* is contained in the set of solutions of Γ.

We illustrate our notion of constraint-solving algorithms in the following example.

Example 35 (Constraint Solving Algorithm)**.** Consider a constraint solving algorithm that uses the simple constraint-reduction strategy that always selects the first applicable rule of the rules given in Figure 8.3 on page 97, which define $\rightsquigarrow^{\mathtt{basic}}_{R,E}$. Assume moreover that the algorithm uses a breadth-first search strategy.

We use this algorithm to prove the statement $P_{Ex} \cup MD_{\Sigma_{PHS}} \vDash^{\exists}_{E_{PHS}} i \doteq j \vee j \lessdot j$. The search tree considered by the algorithm is

$$
\begin{aligned}
\mathit{searchTree}(\{i \doteq j \vee j \lessdot j\}) = \mathsf{Reduce}(\ & \{i \doteq j \vee j \lessdot j\}, && \text{rule } \mathtt{S}_{\vee} \\
& \{\mathsf{Reduce}(\ \{i \doteq j,\ i \doteq i \vee j \lessdot j\}, && \text{rule } \mathtt{S}_{\doteq} \\
& \qquad \{\mathsf{Solved}(\{i \doteq i,\ i \doteq i \vee i \lessdot i\})\}) \\
& ,\mathsf{Reduce}(\{j \lessdot j,\ i \doteq i \vee j \lessdot j\}, \varnothing) \\
& \}) \ ,
\end{aligned}
$$

where the gray boxes annotate the constraint-reduction steps with the used rules. The algorithm thus finds the constraint system $\{i \doteq i,\ i \doteq i \vee i \lessdot i\}$, whose set of solutions is non-empty due to Theorem 5. The solutions of this constraint system are solutions of $\{i \doteq j \vee j \lessdot j\}$, as $\rightsquigarrow^{\mathtt{basic}}_{R,E}$ is a correct constraint-reduction relation.

If required we can use the proof of Theorem 5 to construct a solution of this constraint system, and thus construct a trace of P_{Ex} satisfying $i \doteq j \vee j \lessdot j$. The constructed solution would be (dg, θ) for $\theta := \{i \mapsto 1\}$ and $dg := ([\,]\dashv[\,]\rightarrow[\mathsf{Fr}(\mathsf{a})]\,], \varnothing)$ with $\mathsf{a} \in \mathit{FN}$. Note that dg contains an instance of the FRESH rule, as the solution construction method ensures that all temporal variables are instantiated with indices of nodes in dg. The trace of this solution is $[\,]$, as expected. ♠

Note the difference in difficulty between proving satisfiability claims and proving validity claims. Validity claims can only be proven if the corresponding search tree is of finite depth. If the search tree is of infinite depth, satisfiability claims can however still be proven, as proving a satisfiability claim just requires the corresponding search tree to contain at least one Solved-node at finite depth.

9.2. The TAMARIN Prover

The TAMARIN prover implements the theory described in this thesis extended with the support for equational theories modeling Diffie-Hellman exponentiation, bilinear

pairings, and multisets from [Sch12]. Note that the development of the Tamarin prover and its theory was joint work with Benedikt Schmidt, as detailed in Section 1.3. We explain the implementation of the Tamarin prover and its use as follows.

In Subsection 9.2.1, we first state the verification problems that are analyzed by the Tamarin prover and then detail their translation to *constraint solving problems*, i.e., determining whether a constraint system has a solution. Efficiently solving these problems using our constraint-reduction rules raises certain implementation concerns, which we state and address in Subsection 9.2.2. Afterwards, in Subsection 9.2.3, we explain the constraint-reduction strategy employed by Tamarin's automatic mode. Finally, we explain Tamarin's interactive mode in Subsection 9.2.4.

9.2.1. Supported Verification Problems

On a high level, Tamarin supports the analysis of satisfiability and validity claims of trace formulas for protocols that possibly make use of a public channel controlled by a Dolev-Yao style adversary. This analysis is performed modulo an equational theory modeling the semantics of the employed cryptographic algorithms. The current implementation [MS12] supports equational theories combined from

- an arbitrary subterm-convergent rewriting theory,
- the equations E_{DH}, given in Section 7.5.1, modeling Diffie-Hellman exponentiation,
- the equations modeling multiset union, given in [Sch12], and
- the equations modeling bilinear pairing, given in [Sch12].

To simplify the specification of security protocol verification problems, the Tamarin prover allows to restrict the set of considered traces using axioms. We can use axioms for example to implement rules with inequality checks as follows. We first add $\mathsf{InEq}(t, s)$ actions to all rules that require the term t to be different from the term s, and then filter out the traces where t and s are instantiated to the same message using the axiom $\forall x\, i.\, \mathsf{InEq}(x, x)@i \Rightarrow \bot$.

The input given to the Tamarin prover specifies the protocol (including the adversary capabilities), the considered equational theory, and the expected properties jointly as a security protocol theory. Formally, a *security protocol theory* is a six-tuple $T = (\Sigma, E, P, \vec{\alpha}, \vec{\varphi}, \vec{\chi})$ of a signature Σ specifying the functions for constructing cryptographic messages, an equational theory E specifying the semantics of the functions in Σ, a set of protocol rules P, and sequences of closed trace formulas $\vec{\alpha}$, $\vec{\varphi}$, and $\vec{\chi}$. We call $\vec{\alpha}$ the *axioms of* T , $\vec{\varphi}$ the *validity claims of* T, and $\vec{\chi}$ the *satisfiability claims of* T . We say that the security protocol theory T is *sound* iff all its validity and satisfiability claims hold for the traces of $P \cup MD_\Sigma$ satisfying the axioms; i.e.,

- $P \cup MD_\Sigma \vDash_E^\forall (\bigwedge_{\alpha \in set(\vec{\alpha})} \alpha) \Rightarrow \varphi$ for each $\varphi \in set(\vec{\varphi})$, and
- $P \cup MD_\Sigma \vDash_E^\exists (\bigwedge_{\alpha \in set(\vec{\alpha})} \alpha) \wedge \chi$ for each $\chi \in set(\vec{\chi})$.

The objective of the Tamarin prover is to support its user in checking the soundness of security protocol theories. We illustrate this in the following example.

input

```
theory Artificial_Example begin

functions: sdec/2, senc/2
equations: sdec(senc(m, k), k) = m

rule Step1:  [Fr(x), Fr(k)  ] --[]-> [St(x,k), Out(senc(x,k)), Key(k)]
rule Step2:  [St(x,k), In(x)] --[Fin(x,k)]-> []
rule Reveal: [Key(k)        ] --[Rev(k)  ]-> [Out(k)]

lemma Fin_reachable: exists-trace
  "Ex k S #i.  Fin(S, k) @ i"

lemma Fin_unique: all-traces
  "All S k #i #j. Fin(S, k) @ i & Fin(S, k) @ j ==> #i = #j"

lemma Keys_must_be_revealed: all-traces
  "All k S #i.  Fin(S, k) @ i ==> Ex #j. Rev(k) @ j & j < i"

end
```

ouput

```
[[...some of the output elided...]]

lemma Keys_must_be_revealed:
  "All k S #i.  Fin(S, k) @ i ==> Ex #j. Rev(k) @ j & j < i"
simplify
solve( St( S, k ) |>[0] #i )
  case Step1
  solve( !KU( ~n ) @ #vk )
    case Step1
    solve( !KU( ~n.1 ) @ #vk.1 )
      case Reveal
      by contradiction // from formulas
    qed
  qed
qed

summary of the analysis:
  Fin_reachable (exists-trace): verified (5 steps)
  Fin_unique (all-traces): verified (8 steps)
  Keys_must_be_revealed (all-traces): verified (5 steps)
```

Sort annotations are prefixed, i.e., `~n` denotes *n:fresh* and `#i` denotes *n:temp*. Variables in subformulas of the form `@ i` or `i < j` are of sort *temp*, all other variables without a sort-prefix are of sort *msg*. The expression `St( S, k ) |>[0] #i` denotes the premise constraint $\mathsf{St}(S,k) \blacktriangleright_0 i$. The fact symbol `!KU` denotes $\mathsf{K}^{\uparrow}$.

Figure 9.1.: Example security protocol theory specified using Tamarin's input language and the corresponding analysis results generated by Tamarin's automatic mode

Example 36 (TAMARIN input and output)**.** Figure 9.1 shows the input file modeling the P_{Ex} protocol from Example 18 on page 79 and the security properties considered in Example 23 on page 95, Example 24 on page 101, and Example 33 on page 126. The listed output was generated using TAMARIN's automatic mode. Note that TAMARIN does not output the Reduce-nodes of a search tree. Instead it outputs for each proof step which constraint was solved. It also uses a simple heuristic for naming the resulting cases. Their corresponding constraint systems can be inspected using TAMARIN's interactive mode. ♠

Translation to Constraint Solving Problems

The TAMARIN prover checks the soundness of security protocol theories by translating their validity and satisfiability claims to corresponding constraint solving problems. These problems are then solved using the appropriate instance of the $\rightsquigarrow^{\text{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ constraint-reduction relation. Schmidt explains the decomposition of the equational theory E into the rewriting system $\mathcal{R}$ and the equational theory $\mathcal{AX}$ in detail in [Sch12]. In general, we try to orient as many equations as possible and use them as term rewriting rules in $\mathcal{R}$. We use the equational theory $\mathcal{AX}$ to handle the remaining equations, e.g., associativity and commutativity of the multiplication in the group of Diffie-Hellman exponents. For more information about the computation of the normal form message deduction rules ND, we also refer the reader to [Sch12].

In the following, we specify the translation of the soundness of a security protocol theory $T = (\Sigma, E, [\alpha_1, \ldots, \alpha_n], [\varphi_1, \ldots, \varphi_m], \vec{\chi})$ to an equivalent set of constraint solving problems. Our translation supports the use of validity claims as *lemmas* in later proofs. This improves both the efficiency (by sharing common subproofs) and the effectiveness (by adding additional protocol-specific inductive invariants) of our translation. We assume that the lemmas are an appropriately marked subset of the validity claims. To ensure the soundness of our translation, we require that a validity claim φ_i is only used as a lemma in the proof of a validity claim φ_j if $i < j$.

To extend the scope of our translation, we assume given a partial function GP that translates a trace property that is trivially equivalent to a guarded trace property to this guarded trace property. For example, we expect this function to rewrite all formulas to negation normal form and perform trivial syntactic manipulations like the reordering of conjunctions to ensure the guardedness of quantified variables. We further assume that all formulas in T are in the domain of GP, i.e., can be converted to equivalent guarded trace properties.

Translating claims proven without trace induction Let $\chi \in \mathit{set}(\vec{\chi})$ be a satisfiability claim of T and let $\lambda_1, \ldots, \lambda_l$ be the lemmas to be used in its proof. We translate the soundness of this claim to the constraint solving problem of determining whether

$$\mathit{nsols}_{P,ND,\mathcal{R},\mathcal{AX}}(\{\mathrm{GP}(\alpha_1), \ldots, \mathrm{GP}(\alpha_n), \mathrm{GP}(\lambda_1), \ldots, \mathrm{GP}(\lambda_l), \mathrm{GP}(\chi)\}) \neq \varnothing \,.$$

For a validity claim φ of T to be proven using the lemmas $\lambda_1, \ldots, \lambda_l$, the corresponding constraint solving problem is to determine whether

$$\mathit{nsols}_{P,ND,\mathcal{R},\mathcal{AX}}(\{\mathrm{GP}(\alpha_1), \ldots, \mathrm{GP}(\alpha_n), \mathrm{GP}(\lambda_1), \ldots, \mathrm{GP}(\lambda_l), \mathrm{GP}(\neg\varphi)\}) = \varnothing \,.$$

Both translations follow straightforwardly from the definition of the soundness of the security protocol theory T and the verification theory developed in Chapter 8.

Translating claims proven using trace induction When using trace induction, we must weaken the induction hypothesis with the precondition that all axioms must hold. To simplify stating the resulting constraint systems, we assume without loss of generality that all axioms and lemmas are guarded trace properties. For a satisfiability claim χ of T to be proven using trace induction, and using the lemmas $\lambda_1, \ldots, \lambda_l$, the corresponding constraint solving problem is to determine whether

$$nsols_{P,ND,\mathcal{R},\mathcal{AX}}(\{\alpha_1, \ldots, \alpha_n, \lambda_1, \ldots, \lambda_m, \mathrm{GP}(\mathrm{IH}(\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \neg\chi))), \mathrm{GP}(\chi)\}) \neq \varnothing \ .$$

For a validity claim φ of T to be proven using trace induction, and using the lemmas $\lambda_1, \ldots, \lambda_l$, the corresponding constraint solving problem is to determine whether

$$nsols_{P,ND,\mathcal{R},\mathcal{AX}}(\{\alpha_1, \ldots, \alpha_n, \lambda_1, \ldots, \lambda_m, \mathrm{GP}(\mathrm{IH}(\alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \varphi))), \mathrm{GP}(\neg\varphi)\}) = \varnothing \ .$$

Note that we actually do not need to weaken the induction hypothesis with axioms that are safety properties, i.e., trace formulas whose counter-examples remain counter-examples for every extension of their trace. This improves the efficiency of the application of the induction hypothesis during constraint solving. In the Tamarin prover, we exploit that a guarded trace property is a safety formula if it does not contain an existentially quantified temporal variable.

9.2.2. Implementation Concerns

We solve the constraint solving problems corresponding to the soundness of a security protocol theory using an optimized constraint-reduction relation derived from $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$. We denote this optimized relation by $\rightsquigarrow^{\mathtt{impl}}$. It implements two optimizations that improve its usability and efficiency.

First, the $\rightsquigarrow^{\mathtt{impl}}$ constraint-reduction relation allows to delay the case distinctions on which variant of a protocol rule is used to solve a premise constraint. This optimization is crucial for reasoning about Diffie-Hellman exponentiation, as otherwise large case distinctions would have to be performed up-front and the proofs would become too large. Schmidt explains this optimization in his thesis [Sch12].

Second, the $\rightsquigarrow^{\mathtt{impl}}$ constraint-reduction relation contracts sequences of trivial constraint-reduction steps. As we will see, this contraction is implemented such that the work of performing trivial constraint-reduction steps is shared between different (sub)proofs. Moreover, this contraction also allows proofs constructed using $\rightsquigarrow^{\mathtt{impl}}$ to be human readable and to highlight the key correctness argument underlying a protocol. Informally, this contraction is the result of (1) a fixed set of constraint-reduction rules, called the simplification rules, that we eagerly apply to all constraint systems and (2) a protocol-specific precomputation of the possible results of solving premise constraints and $\mathsf{K}^{\uparrow}$-actions. We detail this contraction in the following two subsections.

Simplification of Constraint Systems

We partition the constraint-reduction rules of $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ into case distinction rules, contradiction rules, and simplification rules. The *case distinction rules* are all rules that may result in a case distinction and need to be applied with care to avoid unnecessary case distinctions and non-termination. For our purposes the case distinction rules are $\mathtt{S}_{@}$, $\mathtt{S}_{\mathsf{K}^{\uparrow}{}_{@}}$, $\mathtt{S}_{\blacktriangleright}$, $\mathtt{S}_{\mathsf{K}^{\downarrow}{}_{\blacktriangleright}}$, $\mathtt{S}_{\dashrightarrow}$, $\mathtt{S}_{\vee}$, $\mathtt{S}_{\neg,\doteq}$, $\mathtt{S}_{\neg,<}$, and $\mathtt{S}_{\neg,\mathrm{last}}$. Note that the unification rule $\mathtt{S}_{\approx}$ is the only rule that may result in multiple cases, but is not considered a case distinction rule. We make this choice because applying the substitutions resulting from the $\mathtt{S}_{\approx}$ rule is the core means for propagating information between different constraints, and we therefore apply this rule eagerly. The *contradiction rules* are all rules that immediately reduce a constraint system to the empty set of constraint systems, i.e., the rules $\mathtt{S}_{\bot}$, $\mathtt{S}_{\neg,@}$, $\mathtt{S}_{\neg,\approx}$, $\mathtt{DG}_{<}$, $\mathtt{N1}$, $\mathtt{N2}$, and $\mathtt{S}_{\mathrm{last},<}$. The *simplification rules* are all rules of $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ that are neither case distinction rules nor contradiction rules, e.g., $\mathtt{S}_{\approx}$ and $\mathtt{S}_{\forall}$.

The simplification rules correspond to trivial proof steps that we want to contract in $\rightsquigarrow^{\mathtt{impl}}$. We achieve this by requiring that the constraint systems in the range of $\rightsquigarrow^{\mathtt{impl}}$ are always saturated under the application of simplification rules. We implement this by applying simplification rules eagerly to our constraint systems. This may actually result in non-termination, e.g., for a constraint system containing a trace formula of the form $\forall x\ y\ i.\ f(x,y)@i \Rightarrow \exists z\ j.\ f(y,z)@j$ for some fact symbol f. The problem with this formula is that saturation under $\mathtt{S}_{\forall}$ and $\mathtt{S}_{\exists}$ introduces unboundedly many f-actions, as soon as the system contains some f-action. However, such trace formulas are rather artificial and we therefore ignore this problem in our current implementation [MS12].

Note that from an automation perspective there is no difference between contradiction rules and simplification rules. We also apply contradiction rules eagerly to our constraint systems to prune as many of them as early as possible. From a user's perspective there is however a difference. While simplification steps are mostly boilerplate steps, the application of a contradiction rule in a proof often conveys important information about the correctness argument underlying a protocol. We therefore differentiate between these two types of rules and make the application of contradiction rules explicit in the proofs generated by the Tamarin prover.

Precomputed Case Distinctions

Precomputed case distinctions generalize the idea of specializing the Chain rule for a specific protocol, which we explain in Section 4.2 of Part I. Precomputed case distinctions implement two optimizations. First, they allow us to contract constraint-reduction steps that are trivial in the context of a specific protocol, e.g., solving an St-premise in the context of the P_{Ex} protocol. Second, they enable sharing the work of performing these trivial constraint-reduction steps between different (sub)proofs. The following example illustrates the underlying idea.

Example 37 (Precomputed Case Distinction)**.** Consider solving a constraint of the form $\mathsf{K}^{\uparrow}(x{:}\mathit{fresh})@j_6$ in the context of the P_{Ex} protocol from Example 18 on page 79. Intuitively, there are three cases of how the adversary could have learned the fresh name denoted by $x{:}\mathit{fresh}$:

1. he could have generated it by himself,

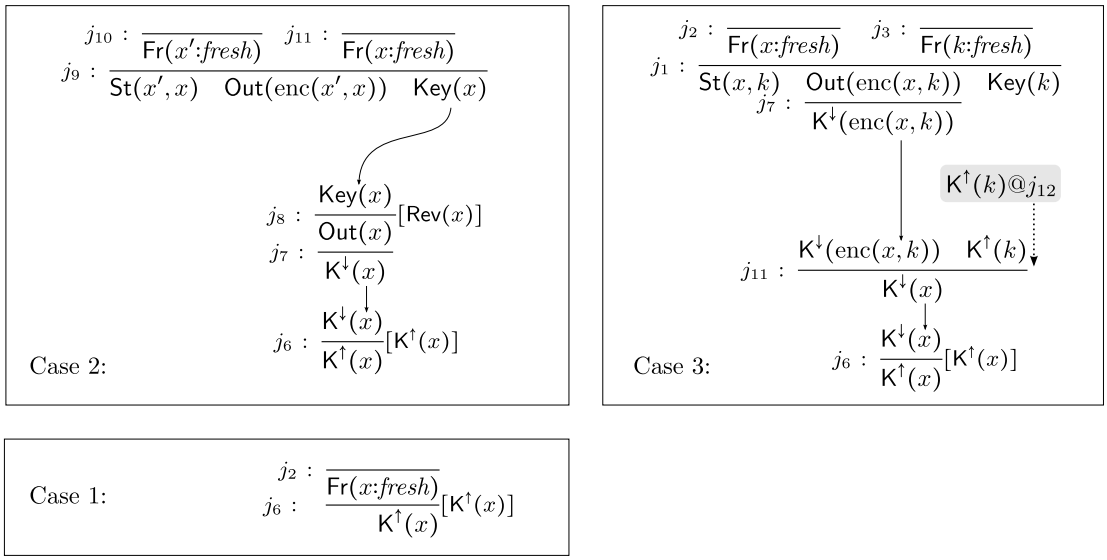


Figure 9.2.: The three precomputed cases that cover all possible ways of solving a constraint of the form $\mathsf{K}^{\uparrow}(x{:}\mathit{fresh})@j_6$ in the context of the P_{Ex} protocol

2. he could have learned it by revealing a key generated by an instance of the first rule of the P_{Ex} protocol, or
3. he could have learned it by decrypting it from the message sent by an instance of the first step of the P_{Ex} protocol.

We can formally compute these cases as follows. We start with the constraint system $\{\mathsf{K}^{\uparrow}(x{:}\mathit{fresh})@j_6\}$ and solve every occurring constraint whose solving does not lead to non-termination. In the context of the P_{Ex} protocol, it suffices to avoid solving more than one $\mathsf{K}^{\uparrow}$-action to avoid non-termination. Figure 9.2 depicts the resulting three constraint systems, whose joint set of solutions is equal to the solutions of $\{\mathsf{K}^{\uparrow}(x{:}\mathit{fresh})@j_6\}$. These constraint systems correspond exactly to the expected three cases. Their only unsolved constraint is the $\mathsf{K}^{\uparrow}(k)@j_{12}$ action in Case (3).

The benefit of these precomputed cases is that they allow us to solve any $\mathsf{K}^{\uparrow}(y{:}\mathit{fresh})$-action constraint in a *big-step fashion* in the context of the P_{Ex} protocol. This works because we can rename the variables in $\mathsf{K}^{\uparrow}(x{:}\mathit{fresh})@j_6$ and the corresponding precomputed cases jointly without changing the relation between their sets of solutions. For example, we use these precomputed cases as follows to solve a constraint $\gamma := \mathsf{K}^{\uparrow}(y{:}\mathit{fresh})@i$ contained in some constraint system Γ. We first rename x to y, j_6 to i, and all other variables in these precomputed cases away from the free variables of Γ. Let the renamed cases be Δ_1, Δ_2, and Δ_3. The three resulting constraint systems covering all possibilities of solving γ in Γ are then $\Gamma' \cup \Delta_1$, $\Gamma' \cup \Delta_2$, and $\Gamma' \cup \Delta_3$. ♠

The above example provides intuition on how to use precomputed case distinctions. For a formal treatment of their usage, see the explanation of constraint rewriting rules in [Sch12]. In the following, we focus on the precomputation of the case distinctions for a protocol P. We precompute a case distinction for every premise constraint built from a fact symbol f occurring in P. We compute such a case distinction by starting with the

constraint system $\{f(\vec{x}) \blacktriangleright_0 i\}$ for $\vec{x} \in \mathcal{V}^*_{msg}$ and $i \in \mathcal{V}_{temp}$, and solving every constraint whose solving does not lead to non-termination. We also precompute case distinctions for each message with a different outermost constructor. More precisely, we also precompute case distinctions for the constraint $\mathsf{K}^{\uparrow}(x{:}\mathit{fresh})@i$ and for all $\mathsf{K}^{\uparrow}(g(\vec{x}))@i$ constraints with $g \in \Sigma$, $\vec{x} \in \mathcal{V}^*_{msg}$, and $i \in \mathcal{V}_{temp}$. The key difficulty of the precomputation is to guarantee termination, while still solving as many constraints as possible in order to share as much work as possible.

Note that the precomputation always starts from a constraint system containing a single trace atom or graph constraint. When solving such a constraint system, non-termination must involve the solving of unboundedly many premise constraints or $\mathsf{K}^{\uparrow}$-actions. A simple criterion to avoid non-termination would thus be to solve at most one premise and at most one $\mathsf{K}^{\uparrow}$-action. This is however overly conservative, as the solving of most premises will terminate. We therefore only exclude the multiple solving of $\mathsf{K}^{\uparrow}$-actions and the multiple solving of a certain set of premises, which we call the *loop breakers*. Their defining property is that every infinite sequence of solving premise constraints must involve an infinite number of loop breakers.

For a protocol P, we use elements in $P \times \mathbb{N}$, which denote premises of rules in P, to mark loop-breakers. During the case distinction precomputation, we avoid solving the corresponding premise constraints. We select the loop-breakers such that they are a feedback-vertex set in the *fact-flow graph* induced by the relation

$$\begin{aligned} \mathit{factFlow}_E(P) := \{&((ru, i), (ru', j) \\ &\mid ru, ru' \in P,\ i \in \mathit{idx}(\mathit{prems}(ru)),\ j \in \mathit{idx}(\mathit{prems}(ru')), \\ &\exists f \in \mathit{concs}(ru).\ \mathit{insts}(f) \cap_E \mathit{insts}(\mathit{prems}(ru')_j) \neq \varnothing\ \}\ , \end{aligned}$$

where a *feedback-vertex set* of a graph G is a set of vertices F such that each cycle of G contains at least one vertex in F. In Tamarin, we first compute the fact-flow relation using E-unification. Then, we compute the set of loop-breakers using a simple, greedy algorithm that repeatedly searches for cycles and marks the first vertex that it finds in a cycle as a loop-breaker. This choice has worked well in our case studies.

To share even more work when using our precomputed case distinctions, we finalize them using the following two additional steps. First, we saturate them jointly under the solving of premise and $\mathsf{K}^{\uparrow}$-action constraints whose corresponding precomputed case distinction contains at most one case. This corresponds to eagerly solving premise and $\mathsf{K}^{\uparrow}$-action constraints that are trivial in the context of a specific protocol. We call the resulting set of precomputed case distinctions the *untyped case distinctions*. In a second step, we extend the individual cases of the untyped case distinctions with the validity claims in a security protocol theory that are marked as type assertions. We saturate the extended cases again under the solving of (trivial) constraints and call the resulting set of precomputed case distinctions the *typed case distinctions*. Provided that the type assertions are strong enough, the typed case distinctions will not contain any deconstruction chains. We use the untyped case distinctions in the inductive proofs of the type assertions. For the proofs of all other claims, we use the typed case distinctions and thereby avoid solving deconstruction chain constraints in the proofs of these claims.

Note that the precomputed case distinctions often provide valuable insights into the workings of a protocol in the context of an adversary. Therefore, our first step after

modeling a protocol in Tamarin is often to inspect its precomputed case distinctions using the GUI.

9.2.3. The Automatic Mode

Selecting the next case distinction rule or precomputed case distinction to apply to a constraint system constitutes the key difficulty in the construction of a security proof using $\rightsquigarrow^{\mathtt{impl}}$. In Tamarin's automatic mode, this selection is performed by a heuristic. To explain this mode, we introduce the notion of a goal of a constraint system.

First, note that all case distinction rules are $\mathtt{S}_\gamma$ rules. Moreover, precomputed case distinctions can be seen as big-step versions of case distinction rules. As explained earlier, a $\mathtt{S}_\gamma$ rule solves the constraint underlined in its side-condition, and the boxed part of its side condition ensures that this constraint is not solved more than once. The *open goals* of a constraint system Γ are all constraints in Γ that may be solved using one of the case distinction rules. The *goals* of Γ are all constraints in Γ that could be solved using one of the case distinction rules if we ignored the boxed part of their side-condition. The goals of Γ are thus a superset the open goals of Γ.

The open goals formalize precisely the choices that can and need to be made when constructing proofs using $\rightsquigarrow^{\mathtt{impl}}$. To implement an automatic constraint-solving mode, we use a heuristic that selects the next open goal to solve. The design rationale underlying Tamarin's heuristic is that it prefers goals that are either trivial to solve or likely to result in a contradiction. Tamarin's heuristic works as follows. It groups the goals into the categories given below and prefers goals from an earlier category to goals from a later category. The goals in a single category are solved in chronological order, i.e., the order in which they were introduced in the constraint systems in the search tree. The employed categories are

1. In-premise, Fr-premise, and K-action goals,
2. $\mathsf{K}^{\downarrow}$-premise goals and deconstruction chain goals,
3. disjunction and negation goals,
4. premise goals that do not correspond to premises marked as loop-breakers,
5. action goals that are no $\mathsf{K}^{\uparrow}$-actions,
6. $\mathsf{K}^{\uparrow}(x)$-action goals with $x \in \mathcal{V}_{fresh}$,
7. $\mathsf{K}^{\uparrow}(m)$-action goals with $St(m) \cap \mathcal{V}_{fresh} \neq \varnothing$ that are unlikely to be deducible by the adversary, defined below
8. premise goals marked as loop breakers and $\mathsf{K}^{\uparrow}(m)$-action goals that are unlikely to be deducible by the adversary
9. the $\mathsf{K}^{\uparrow}(m)$-actions that are likely to be deducible by the adversary.

We consider a $\mathsf{K}^{\uparrow}(m)@i$ goal of a constraint system Γ *likely to be deducible* by the adversary, if $vars(m) \cap \mathcal{V} = \mathcal{V}_{pub}$, or, if m occurs as a top-level pair-component of an Out-conclusion or a $\mathsf{K}^{\downarrow}$-conclusion of a node j not after i, i.e., a node j that does not satisfy $i \lessdot_\Gamma j$. In our case studies, delaying goals that are likely to be deducible by the adversary significantly improved the verification speed.

9.2.4. The Interactive Mode

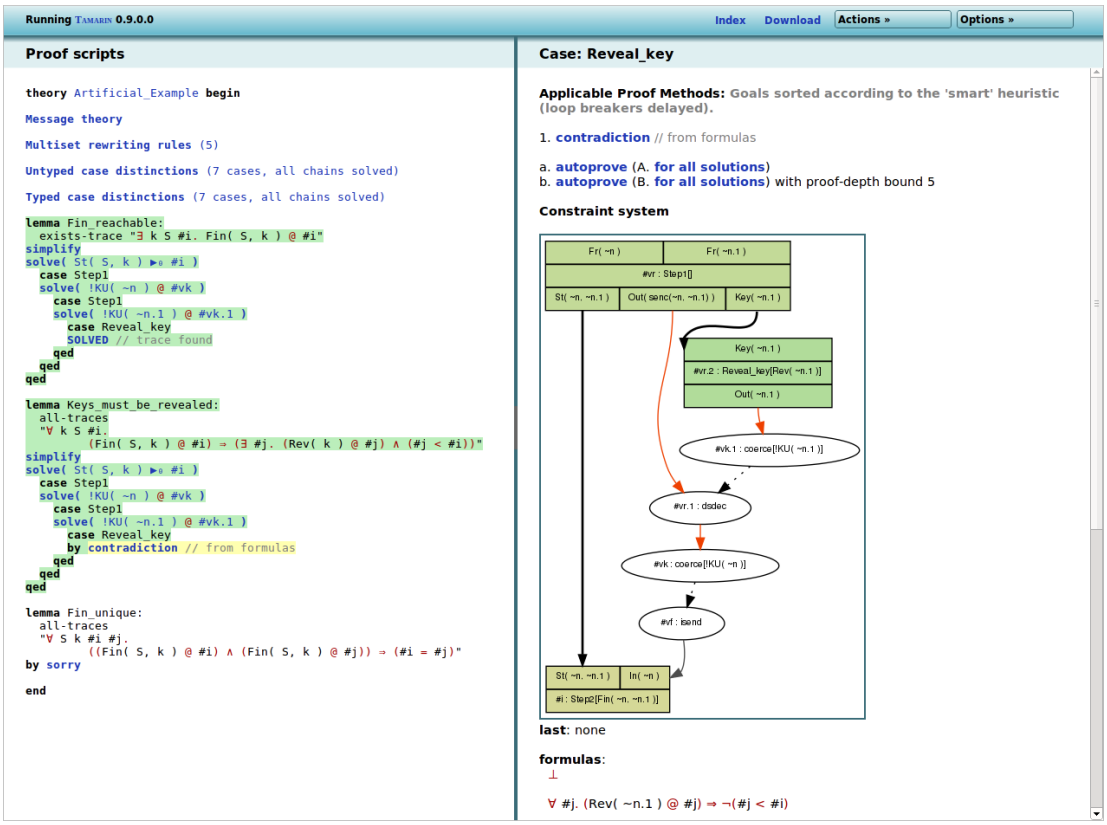
The interactive mode of the Tamarin prover provides a web-based Graphical User Interface (GUI) that allows the user to interactively explore and construct protocol security proofs using the $\rightsquigarrow^{\mathtt{impl}}$ constraint-reduction relation. The first version of this GUI was implemented by Cedric Staub as part of his bachelor's thesis [Sta11]. Figure 9.3 on the following page provides screenshots illustrating the two main use cases of Tamarin's interactive mode.

In screenshot (a), we use the GUI to explore the automatically generated proof of the property of the P_{Ex} protocol that we prove in Example 33 on page 126. On the left-hand-side, the GUI displays an overview of the security protocol theory considered and the constraint-reduction steps used in the proofs of its claims. Note that for historical reasons the claims of a security protocol theory are called lemmas in Tamarin. We refer the reader to Tamarin's user manual for the explanation of how claims are marked for reuse or as type assertions. Items on the left-hand-side of the GUI that have associated information are marked in blue, and the right-hand-side of the GUI displays the information associated to the currently selected item. In this screenshot, we selected the one constraint-reduction step that applies a contradiction rule in the proof of the `Keys_must_be_revealed` claim. This step corresponds to the reduction of System $\Gamma_{1^2,2,1^4}$ from Figure 8.15 on page 125 to the empty set of constraint systems. The graph on the right-hand-side depicts all graph constraints of constraint system associated to this step. The trace formulas in this system are listed in textual form below the depicted graph constraints. Note that these graph constraints actually correspond to an execution of the P_{Ex} protocol. This system is nevertheless no counter-example to the `Keys_must_be_revealed` claim, as it violates the trace formula $\forall j.\, \mathsf{Rev}(n_1{:}\mathit{fresh})@j \Rightarrow \neg(j \lessdot i)$ listed below the graph constraints.

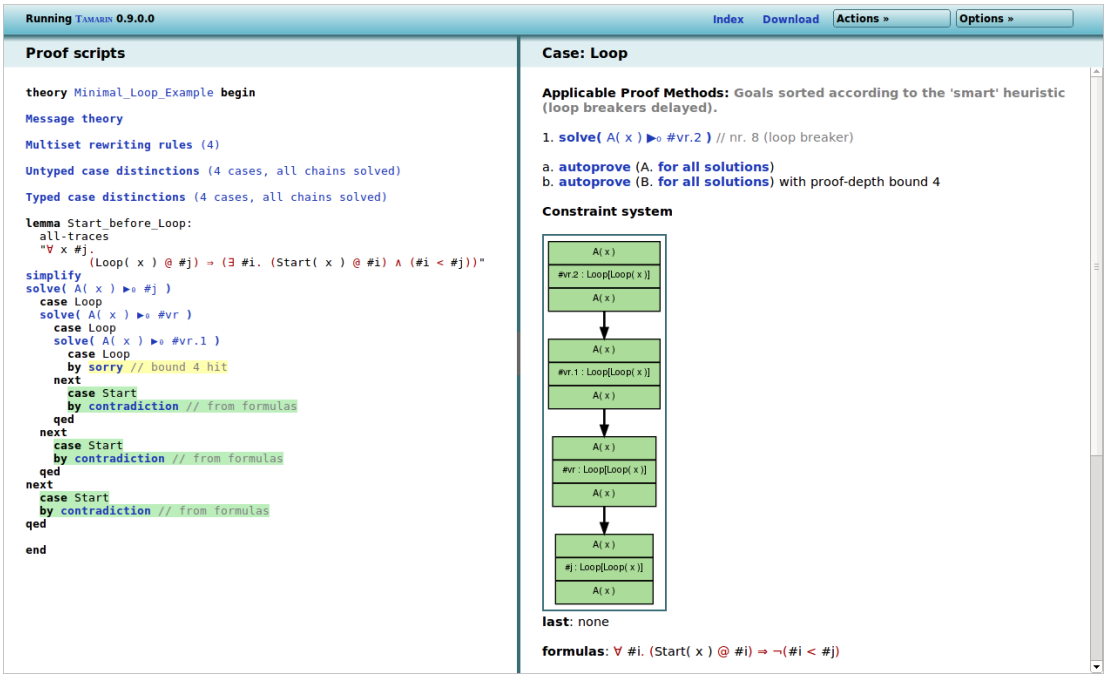
Note that proofs containing "sorry" steps are incomplete. The user can complete these proofs using the proof methods listed above the constraint system associated to a constraint-reduction step. A proof method either solves one of the constraint system's open goals using $\rightsquigarrow^{\mathtt{impl}}$ or it uses Tamarin's automatic mode to try to solve the currently selected constraint system. The depth of the proofs generated by the automatic mode can be bounded to guarantee termination. We used this in screenshot (b) to investigate the non-terminating proof attempt considered in Example 25 on page 104. The structure of the resulting incomplete proof and the depicted constraint system highlight that we are missing an argument (i.e., the induction hypothesis) about why the solving of an $\mathsf{A}(x)$ premise with the conclusion of the $\mathsf{A}(x)$–[$\mathsf{Loop}(x)$]→$\mathsf{A}(x)$ rule does not lead to a solution.

9.3. Experimental Results

We validate the practical applicability of the Tamarin prover on three classes of security protocols: classic security protocols, protocols with loops and non-monotonic state, and Diffie-Hellman based authenticated key exchange protocols. In the following, we present the experimental results that we obtained using the automatic mode of version `0.8.2.0` of the Tamarin prover. We measured the reported runtimes on an Intel i7 Quad Core laptop with 4GB RAM. All our protocol models are distributed together with the source



(a) inspecting a contradiction step of an automatically generated proof



(b) constructing a proof interactively to investigate the reasons for non-termination

Figure 9.3.: Screenshots of Tamarin's GUI

code of the Tamarin prover [MS12].

9.3.1. Classic Security Protocols

Note that the verification theory and the heuristic employed by the Tamarin prover generalize the theories and heuristics employed by the `scyther-proof` and the Scyther tools, which are specialized to the analysis of classic security protocols. Results obtained using these tools (e.g., the ones given in Table 5.1 on page 52) are therefore also obtainable using the Tamarin prover. Due to its generality, the Tamarin prover is however slower than these specialized tools.

We estimate the loss in efficiency with respect to `scyther-proof` on the three benchmarks given in Table 9.1. The protocols considered in these benchmarks are the three-message version of the Needham-Schroeder Public-Key protocol [NS78] analyzed (NSPK3) and fixed (NSLPK3) by Lowe in [Low96] and the TLS handshake protocol as modeled by Paulson in [Pau99]. For each of these protocols, we analyze secrecy and injective agreement. The analysis results of `scyther-proof` on these benchmarks are given in Table 5.1 on page 52. We observe that `scyther-proof`'s proof generation is six times faster on NSLPK3 and three times faster on the TLS handshake. We attribute the larger slowdown on NSLPK3 to the protocol-specific precomputation of the case-distinctions performed by the Tamarin prover. Amortizing the precomputation time is hard for a protocol as small as NSLPK3.

Note that these three benchmarks are also used in [CLN09] to compare the performance of several (sometimes bounded) security protocol verification tools. A direct comparison to the results presented in [CLN09] is not possible due to the difference in the properties analyzed and the processor used. We observe however that the bounded verification of these protocols quickly takes longer than a few seconds when increasing the number of concurrent sessions. We thus consider the few seconds taken by Tamarin to perform an unbounded verification of these benchmarks to be well acceptable in practice.

One drawback of Tamarin is that it sometimes requires type assertions to effectively analyze a protocol. They are required because we analyze protocols in an untyped model, which ensures that we can also find typeflaw attacks. We expect that we can alleviate the need for type assertions by adapting the type inference heuristic described in Section 5.2.5 to multiset rewriting based protocol models. Moreover when not considering typeflaw attacks, one can annotate all variables in the protocol specification with more precise sorts than *msg*, which ensures that all deconstruction chains can be refined completely. Type assertions are then not required.

	Protocol	Properties	Result	Time [s]	Type Assertion
1.	NSPK3 [NS78, Low96]	secrecy, inj. agree	attack	1.8	yes
2.	NSLPK3 [Low96]	secrecy, inj. agree	proof	1.5	yes
3.	TLS handshake [Pau99]	secrecy, inj. agree	proof	2.6	no

Table 9.1.: Results of analyzing classic security protocols using Tamarin

	Verification Problem	Features	Time [s]	Invariants
1.	TESLA 1 [PCTS00]	loops	3.3	–
2.	TESLA 2 (lossless) [PCTS00]	loops	16.6	2
3.	TESLA 2 [PCTS00]	loops	non-termination	–
4.	Security Device [ARR11]	non-mono	0.4	1
5.	Contract Signing [ARR11]	non-mono	0.5	–
6.	Keyserver [Möd10a]	loops, non-mono	0.2	–
7.	Exclusive Secrets [DKRS11]	loops, non-mono	1.6	2
8.	Envelope [DKRS11]	loops, non-mono	74.4	2
	(interactively constructed proof)		0.5	
9.	Envelope (no reboot) [DKRS11]	loops, non-mono	19.6	2
10.	YubiKey [KS12]	loops, non-mono	15.0	2
11.	YubiHSM [KS12]	loops, non-mono	11.7	3

Table 9.2.: Results of using Tamarin to automatically analyze protocols with loops or non-monotonic state.

9.3.2. Protocols with Loops and Non-Monotonic State

Our security protocol model is well-suited to the modeling of protocols with loops and non-monotonic state. The results presented in Table 9.2 demonstrate that our verification theory and Tamarin's heuristic are also strong enough to automatically prove their correctness in many cases. In the following, we describe the protocols and properties that we verified and discuss the results we obtained.

Verification Problems 1-3 Problem 1 is the verification of our model of Scheme 1 of the TESLA protocol [PCTS00] that we present in Section 7.5.2. We verify both that there is an honest execution in our model and that stream authentication is guaranteed. Note that we include sanity checks like proving the existence of an honest execution in all our problems to reduce the chance of modeling errors. Tamarin proves these two properties together in 3.3 seconds. The proof of the stream authentication property requires the use of trace induction, but no additional invariants must be specified.

Problems 2 and 3 both consider Scheme 2 of the TESLA protocol. This scheme uses the same delayed authentication construction as Scheme 1, but it replaces the repeated generation of fresh MAC keys with a single inverted hash chain. The use of an inverted hash chain makes this scheme resilient against the loss of intermediate messages. The authenticity of a disclosed MAC key is checked by checking whether its n-th hash corresponds to the last authenticated MAC key, where n is estimated from the key disclosure schedule.

Problem 2 models the message exchange of TESLA Scheme 2, but only claims authentication, if the disclosed key hashes to its *immediate* predecessor in the hash chain; i.e., it verifies that Scheme 2 guarantees authentication in a lossless setting. Using two invariants, one formalizing the uniqueness of the keys in a hash chain and one formalizing that the adversary only knows expired keys, Tamarin verifies this problem in 16.6 seconds.

Problem 3 models the message exchange of TESLA Scheme 2 and the n-th hash check to provide authentication in the case of lost intermediate messages. Tamarin does not terminate on this problem. We found that our current verification theory must be extended with support for reasoning about iterated function application to allow for the verification of this problem. We expect this to be possible by building on the support for reasoning about multisets developed in Schmidt's thesis [Sch12].

Verification Problems 4-9 These problems are case studies considered in the work on extending the Horn-theory based approach with support for reasoning about protocols with non-monotonic state [Möd10a, DKRS11, ARR11].

Problem 4 verifies the exclusivity property of the security device from [ARR11]. Its non-monotonic state is easy to reason about, as the model requires only branching, but no loops. Its verification requires one invariant (a type assertion) stating that the messages decrypted by the device correspond to the secrets that were encrypted. In contrast to [ARR11], we model an unbounded number of security devices.

Problem 5 verifies the fairness property of the contract signing protocol by Garay *et al.* [GJM99] as modeled in [ARR11]. We model the employed private contract signatures as a subterm-convergent rewriting theory. The correctness argument for the fairness property requires the Trusted Third Party (TTP) to store the status of all contracts it was involved with. More precisely, the TTP must answer at most one abortion or resolution request per contract. We model this by adding an action $\mathsf{Answered}(c)$ to each multiset rewriting rule that answers a request for a contract c, and verifying the fairness property for the traces that satisfy $\forall c\, i\, j.\, \mathsf{Answered}(c)@i \wedge \mathsf{Answered}(c)@j \Rightarrow i \doteq j$, i.e., the traces where there was not more than one answer per contract. The proof of the fairness property deals mostly with the different options of combining the cryptographic messages sent by the TTP. Tamarin constructs this proof in 0.5 seconds. In contrast to [ARR11], we do not require abstracting the model such that only a single contract is considered.

Problem 6 corresponds to the keyserver example from [Möd10a], as distributed in [Möd10b]. It models a keyserver storing a database of public keys. Both the adversary and honest clients may register keys with this server. The server moreover accepts requests to update a registered key with a new public key. Upon such a request, the server updates his database and revokes the old public key. In the model distributed in [Möd10b], the server also publishes the private key corresponding to the revoked public key. This is not faithful to a real world implementation, as the update requests do not contain the private keys. We therefore model that the server confirms update requests and the clients publish the corresponding private keys themselves upon confirmation of their request. The property verified in the model from [Möd10b] is that all keys registered to honest clients and known to the adversary have been revoked beforehand. Tamarin verifies this property in 0.2 seconds. In contrast to [Möd10a], we do not require the abstraction to a fixed number of agent identities.

Problem 7 is the running example from [DKRS10a], a protocol that implements exclusive access to secrets using the Trusted Platform Module (TPM). The protocol's three design objectives are: (1) Alice commits to two secrets s_1 and s_2 before knowing Bob's choice; i.e., Alice cannot choose the secrets depending upon Bob's choice. (2) Bob

can learn one of the secrets, but not both. (3) once Alice has committed to the secrets, Bob can open one of them without any interaction or help from Alice. The protocol works by coupling the two secrets to different, exclusive states of Bob's TPM. It thereby guarantees exclusive access to Alice's secrets, provided that Bob, which we identify with the adversary, cannot reboot his TPM. We require two invariants to verify this property. The first invariant is a type assertion characterizing what messages the adversary can learn using the TPM's "unbind" command, which decrypts messages with keys internal to the TPM. The second invariant characterizes the state changes of the TPM. It enables the backwards reasoning to skip over multiset rewriting rules that only read the TPM's state. Tamarin verifies these two invariants and the exclusivity property in 1.6 seconds. In contrast to [DKRS10a], we do not require a protocol-specific manual proof justifying bounding the size of the TPM's state.

Problems 8 and 9 are the variants of the Envelope protocol [AR10] that were analyzed using ProVerif in [DKRS10a]. This protocol implements a virtual escrow of data, i.e., committing to a secret such that the recipient can independently choose whether he wants to access it or not. If the recipient decides not to access the secret, then he gets cryptographic proof that he did not access it and that he will never be able to access it. As in Problem 7, the protocol is implemented using the recipients TPM as a trusted third party. In contrast to Problem 7, the Envelope protocol is secure under the rebooting of the recipient's TPM. Using two invariants, Tamarin succeeds to verify both the variant of the protocol that considers reboots (Problem 8) and the one that does not consider reboots (Problem 9). The two invariants are adapted versions of the ones from Problem 7. Note that ProVerif did not terminate for the protocol that considers reboots and its verification was therefore left as future work in [DKRS10a]. Moreover, Tamarin verifies the version without reboots in 20 seconds, while it takes ProVerif 35 minutes to verify this version [DKRS10b]. This corresponds to a speedup of more than two orders of magnitude.

Note further that there actually exists a short, 28 step long proof which justifies the correctness of our model of the Envelope protocol with reboots (Problem 8). We constructed this proof interactively using Tamarin's GUI and then copied it into the Tamarin input file specifying the Envelope protocol. Checking this proof and the two invariants takes Tamarin only 0.5 seconds. The brevity of this proof demonstrates that our verification theory is well-suited for the formalization of the correctness argument of the Envelope protocol. The size difference between our manual proof (28 steps) and the one found by our heuristic (7136 steps) also highlights that there is significant potential for optimizing our heuristic for protocols like Envelope.

Verification Problems 10 and 11 These two problems were analyzed by Künnemann and Steel in [KS12]. The authors use the Tamarin prover to analyze the Yubikey hardware device, which is used by more than a million users [Yub12] to authenticate themselves against network-based services. We refer the reader to [KS12] for an in-depth discussion of these two verification problems. We chose to include these two problems here because (1) they further demonstrate the expressivity of our protocol model and (2) they demonstrate that the Tamarin prover is ready to be used for independent research.

	Protocol	Security Model	Result	Time [s]
1.	DH2 [CMU11]	weakened eCK [CMU11]	proof	13.7
2.	KAS1 [Nat09]	KI+KCI [CMU11]	proof	0.6
3.	KAS2 [Nat09]	weakened eCK [CMU11]	proof	5.1
4.	KAS2 [Nat09]	eCK	attack	2.0
5.	KEA+ [LM06]	KI+KCI	proof	0.8
6.	KEA+ [LM06]	KI+KCI+wPFS	attack	1.0
7.	NAXOS [LLM07]	eCK	proof	7.2
8.	NAXOS [LLM07]	eCK+PFS	attack on PFS	4.8
9.	SIG-DH	PFS	proof	0.7
10.	SIG-DH	eCK	attack	9.8
11.	STS-MAC [BWM99b]	KI, reg-PK	UKS-attack	3.4
12.	STS-MAC-fix1 [BWM99b]	KI, reg-PK (with PoP)	proof	7.2
13.	STS-MAC-fix2 [BWM99b]	KI, reg-PK	proof	1.8
14.	TS1-2004 [JKL04]	KI	UKS-attack	0.3
15.	TS1-2008 [JKL08]	KI	proof	0.4
16.	TS2-2004 [JKL04]	KI+wPFS	attack on wPFS	0.7
17.	TS2-2008 [JKL08]	KI+wPFS	proof	1.3
18.	TS3-2004/08 [JKL04, JKL08]	KI+wPFS	non-termination	-
19.	UM [BWM99a]	wPFS	proof	0.9
20.	UM [BWM99a]	PFS	attack	0.8

Table 9.3.: Results of using Tamarin to automatically analyze recent AKE protocols

9.3.3. Diffie-Hellman Based Authenticated Key Exchange Protocols

In [SMCB12a], we used the Tamarin prover to analyze several recent Authenticated Key Exchange (AKE) protocols that make use of Diffie-Hellman (DH) exponentiation to achieve security against strong adversaries. See Section 7.5.1 for an overview of AKE protocols and the considered adversaries. We include these case studies here to demonstrate the strength of the verification theory underlying the Tamarin prover. For an in-depth treatment of AKE protocols and reasoning about DH exponentiation, we refer the reader to Schmidt's thesis [Sch12].

Table 9.3 lists the results that we obtained. We modeled the Signed Diffie-Hellman (SIG-DH) protocol, the STS protocol and two fixes [BWM99b], the UM [BWM99a], KEA+ [LM06], and NAXOS [LLM07] protocols, and the TS1, TS2, and TS3 protocols [JKL04] and their updated versions [JKL08]. We also modeled NIST's KAS1 and KAS2 protocols [Nat09] and the related DH2 protocol by Chatterjee *et al.* [CMU11]. For each protocol, we formalized its intended and related security models and analyzed them using Tamarin. For example, to verify Key Independence (KI) for STS, we model that the adversary can reveal certain session keys. Additionally, the adversary can register public keys for himself, even if those keys have been previously registered for another identity. In this example, we find the UKS attack reported in [BWM99b]. The first fix from [BWM99b] requires a Proof-of-Possession (PoP) of the private key for registering a public key. The second fix includes the identities of the participants in

the signatures. We model and successfully verify both fixes. For NIST's KAS1 and KAS2 protocols [Nat09], our analysis confirms both the security proof and the informal statements made in [CMU11].

Our results indicate that, in general, Tamarin is effective and efficient for the analysis of Diffie-Hellman based AKE protocols. For example, it requires 7.2 seconds to verify NAXOS in the eCK model, which we explained in Section 7.5.1.

The non-termination of Tamarin for the TS3-2004/08 protocols is an instance of a general problem that may occur with our approach for reasoning about message deduction. Namely, if a protocol can serve as a generic message deduction oracle, then our normal-form conditions may fail to eliminate sufficiently many redundant steps. For the TS-2004/08 protocols, it is the session key reveal rule that can serve as a generic message deduction oracle. It remains future work to develop normal-form conditions that improve reasoning about such protocols.

9.4. Discussion

On a high level, the Tamarin models of our case studies [MS12] demonstrate that the protocol model presented in Chapter 7 is expressive and well-suited to the specification of a wide range of security protocol verification problems. Our experimental results demonstrate that the verification theory presented in Chapter 8 suffices to analyze most of the resulting constraint-solving problems *and* that this theory can be automated well. Note that many of our case studies were previously out of scope for automatic security protocol verification tools or required manually proving additional protocol-specific statements. Overall, our results are very encouraging and we expect the Tamarin prover to succeed in the automatic analysis of many other case studies of a similar type.

However, the problems that we consider are undecidable in general, and *there is no silver bullet for such problems.* In the case of the Tamarin prover, the undecidability manifests itself as non-termination. In our case studies, supplying additional invariants proven using trace induction often suffices to avoid non-termination. Trace induction is however not always strong enough to prove the properties of interest. The main limitation of trace induction is that trace formulas can only relate the actions of a trace with each other. Hence, we cannot use trace induction to reason about the inductive structure of the terms occurring during the execution of a multiset rewriting system. This is the reason why we had to develop special support for reasoning about message deduction, and it is the reason why our attempt to verify Scheme 2 of the TESLA protocol failed. Trace induction is also not sufficient to reason about the mutual exclusion of multiset rewriting rules using the same linear fact in the context of loops. For example, our current constraint-reduction rules do not suffice to prove that the trace property $\forall x\, i\, j.\, \mathsf{Stop}(x)@i \wedge \mathsf{Stop}(x)@j \Rightarrow i \doteq j$ is valid for the multiset rewriting system $R := R_{loop} \cup \{\mathsf{A}(x) \text{–}[\ \mathsf{Stop}(x)\ ]\!\!\rightarrow [\,]\}$. The missing invariant for a successful proof is that no reachable state of R contains more than one $\mathsf{A}(m)$ fact for each $m \in \mathcal{M}$. We can obviously not express this invariant as a trace formula. However, we can manually formalize this concept of facts with exclusive instances and introduce corresponding constraint-reduction rules. We actually did this and implemented preliminary support for reasoning about multiset rewriting systems like R in Tamarin. See its user manual

for more information.

Due to the undecidability of the problems considered, the introduction of additional constraint-reduction rules for reasoning about new types of protocol mechanisms is unavoidable in general. The key benefit of our approach is its compositionality. We can build support for the different mechanisms individually and gain, for free, support for reasoning about protocols that exploit several types of mechanisms jointly. Our combination of $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ and trace induction to reason about cryptographic protocols with loops and non-monotonic state is a prime example of this approach.

Meta-Theoretical Properties of $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$

We have not yet managed to prove meta-theoretical properties of the $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ constraint-reduction relation that explain its practically demonstrated strength. We believe that proving the following two conjectures would provide such an explanation.

Conjecture 1 (Finding Existing Solutions)**.** *There exists a constraint-reduction strategy r derived from the $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ relation such that, for every wellformed constraint system Γ with a non-empty set of solutions, $searchTree_r(\Gamma)$ contains a Solved-node at finite depth.*

Intuitively, this conjecture states that if there exists a solution, then we can find it. We expect that one can prove this conjecture using a constraint-reduction strategy r that ensures that no open goal is delayed for an indefinite number of constraint-reduction steps, i.e., ensures fairness of goal solving. We moreover expect r to consist of the rules from $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ and a rule that allows to perform case distinctions on whether two temporal variables are equal. The following example illustrates the need for such case distinctions.

Example 38 (Solutions Missed by $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$)**.** Consider, the satisfiability claim

$$\left\{\frac{\mathsf{In}(x)\quad \mathsf{In}(y)}{}[\mathsf{A}(x,y)]\right\} \cup MD \vDash^{\exists}_{\emptyset} \begin{array}{l}(\exists x\ y\ i.\ \mathsf{A}(x,y)@i) \wedge \\ (\forall x\ y\ i.\ \mathsf{A}(x,y)@i \Rightarrow \exists z\ j.\ \mathsf{A}(y,z)@j)\ ,\end{array}$$

which obviously holds. An example of a satisfying dependency graph is

$$\left(\left[\frac{}{\mathsf{K}^{\uparrow}(\mathsf{a})}[\mathsf{K}^{\uparrow}(\mathsf{a})],\ \frac{\mathsf{K}^{\uparrow}(\mathsf{a})}{\mathsf{In}(\mathsf{a})},\ \frac{\mathsf{K}^{\uparrow}(\mathsf{a})}{\mathsf{In}(\mathsf{a})},\ \frac{\mathsf{In}(\mathsf{a})\quad \mathsf{In}(\mathsf{a})}{}[\mathsf{A}(\mathsf{a},\mathsf{a})]\right], D\right)$$

for $\mathsf{a} \in PN$ and $D := \{(1,1) \nrightarrow (2,1), (1,1) \nrightarrow (3,1), (2,1) \nrightarrow (4,1), (3,1) \nrightarrow (4,2)\}$. However, the $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ constraint-reduction relation is not sufficient to reduce the constraint-system corresponding to this satisfiability claim to a solved constraint system. The problem is that the $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ relation always introduces new node constraints and does not consider the option that two node constraints might be instantiated to the same node in a solution. ♠

The following conjecture relates our approach for the unbounded verification of security protocols to the known fact that the bounded verification of security protocols is decidable. We expect that only minimal changes of Tamarin's automatic mode are required to convert it into a decision procedure for bounded protocol verification.

Conjecture 2 (Bounded Decision Procedure)**.** *Let b be a bound on the number of instances of protocol rules allowed in a solution. There exists a constraint-reduction strategy r derived from the $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ relation such that exploring a bounded part of $\mathit{searchTree}_r(\Gamma)$ is sufficient to determine whether Γ has a solution with less than b instances of protocol rules.*

Said differently, this conjecture states that there exists a constraint-reduction strategy r that uses the rules of $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ and is guaranteed to always increase in a bounded number of constraint-reduction steps, the minimal number of instances of protocol rules in the solutions of the resulting constraint systems. Informally, this corresponds to showing that the constraint-reduction strategy r cannot get stuck with only solving message deduction constraints. We expect this conjecture to hold at least for subterm-convergent rewriting theories, as their construction and deconstruction rules only introduce goals smaller than the terms sent by protocol rule instances. The key difficulty in the proof of this conjecture is to find a well-founded measure on constraint systems that accounts for all the flexibility allowed by $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$.

10. Related Work

We first discuss other automated methods for the construction of protocol security proofs. Afterwards, we discuss work related to our case studies.

10.1. Automated Construction of Protocol Security Proofs

In this section, we compare related methods for the automated construction of protocol security proofs to our approach. By "our approach" we mean the Tamarin prover in general, and `scyther-proof` in the particular case of constructing machine-checked proofs. We believe that the following four aspects of an automated protocol verification method are important to consider in such a comparison:

1. the expressivity of its specification language,
2. the expressivity of its proof calculus,
3. its automation support, and
4. the trustworthiness of the resulting analyses.

We discuss these aspects in the following four paragraphs. Note that we do not have formal results relating different methods with respect to these aspects. We however use them as guidelines in our comparison.

The specification language determines the verification problems to which a method can be applied. A method's specification language therefore determines the method's maximal scope. The specification language of the Tamarin prover is given by the notion of validity claims, which builds on the notions of labeled multiset rewriting, equational theories, and two-sorted first-order logic. The specification language of `scyther-proof` is given by its notion of a judgment.

The proof calculus is the set of inference steps available to prove a verification problem. In the case of security protocol verification, proof calculi whose proofs are recursively enumerable are necessarily incomplete, as protocol security is undecidable and attacks are recursively enumerable. In our approach, the proof calculus is the set of constraint-reduction rules. In a type-based method, the proof calculus is the type system used to type-check a security protocol. In a method like ProVerif, the proof calculus consists of the available abstractions and the resolution rules implemented in ProVerif. The expressivity of a proof calculus provides a lower-bound on the effectiveness of an automated method, as only verification problems derivable in its calculus can be proven. It also provides a lower-bound on the efficiency of a method, as the time to verify a problem (i.e., construct a proof for it) is at least as long as the time it takes to check size of the smallest proof for this problem constructible in the method's proof calculus. Note that in our approach proofs are made explicit. They correspond to finite search

trees without a Solved-node, as defined in Section 9.1. By comparing the proof calculus of Tamarin to the calculi of other methods, we can see whether there are methods that can construct proofs that we cannot represent (efficiently) in Tamarin's proof calculus. Such comparisons allow us therefore to check whether there are obviously missing constraint-reduction rules.

The automation support of a method determines its practical applicability. Good automation support allows an approach to be applied by practitioners with less training in formal reasoning. This is important for our long-term goal of making the verification of security protocols a routine engineering task. Good automation support also allows hiding trivial proof steps and focus on the key correctness arguments *without* risking to miss corner cases. Note that the extreme case of a fully automatic approach allows considering even the validity of a whole verification problem a trivial proof step, and a missed corner case then corresponds to an unexpected attack.

There are two main factors involved in increasing the trustworthiness of a security protocol analysis:

(i) the faithfulness of the verification problem modeled with respect to the intended statement about the real world and

(ii) the chance that the proof of the verification problem is flawed.

The first factor is obviously subjective. In general, we believe that an expressive specification language with a straightforward semantics simplifies ensuring the faithfulness of a verification problem. In our approach, we reduce the chance of flawed proofs by formally *deriving* our proof calculus from the semantics of our verification problems in a standard, well-understood logic. We further reduce the chance of flawed proofs by providing strong automation (in the form of the Tamarin prover) for the construction of proofs using our calculus. In Part I, we even construct *machine-checked* proofs for a smaller set of verification problems, whose correctness is independent of any implementation errors in the corresponding automatic proof construction tool, i.e., `scyther-proof`.

10.1.1. Computational Protocol Security Proofs

Computational security proofs are mostly manually constructed in the form of pen-and-paper proofs. For a long time, their construction was considered to be too hard to automate. Recently, there has however been significant progress towards their automation. The CryptoVerif tool [Bla08] provides a (semi-)automated way to construct game-based protocol security proofs of industrial scale protocols, e.g., Kerberos [BJST08]. The soundness proofs of the verification theory underlying CryptoVerif have not yet been formalized in a theorem prover. A possible foundation for such a formalization are CertiCrypt [BGZ09] or EasyCrypt [BGHZ11]. CertiCrypt formalizes the theory required for machine-checking computational security proofs of cryptographic primitives (e.g., ElGamal encryption) in the interactive theorem prover Coq. EasyCrypt leverages on SMT solvers to (partially) automate the construction of such proofs. An alternative approach to automate at the checking of computationally sound proofs is to use a type system whose soundness has been proven with respect to a computational semantics, e.g., [Lau05, FKS11]. Note that automatically constructing machine-checked, computational protocol security proofs is an active research area.

10.1.2. Symbolic Protocol Security Proofs

We first discuss automated methods that analyse security protocols for a bounded number of sessions. Afterwards, we discuss the main three kinds of methods for the automated construction of unbounded symbolic protocol security proofs: methods based on backwards-search, abstraction based methods, and type based methods. Finally, we discuss other approaches for the construction of symbolic machine-checked protocol security proofs.

Automatic Protocol Analysis for a Bounded Number of Sessions

Protocol security is NP-complete for a bounded number of sessions, but undecidable for an unbounded number of sessions [DLM04, TEB05]. Many tools therefore analyze protocols only with respect to a bounded number of sessions, e.g., Millen and Shmatikov's constraint solver [MS01], OFMC [BMV05], and SATMC [AC08]. In contrast to Tamarin, these tools can therefore only find attacks on protocols, but not verify protocols for an unbounded number of sessions. Several of these tools are also based on constraint solving. Below we relate our approach to the approach by Millen and Shmatikov [MS01]. A study of the relation to further constraint-solving based approaches is future work.

To simplify our presentation, we use MS to refer to Millen and Shmatikov's approach. This approach considers a message algebra with hashing, signatures, and symmetric and asymmetric encryption. It defines $\mathcal{F}(T)$ to denote the set of all messages deducible from a set of messages T. In MS, there is only one type of constraints. A constraint $m : T$ denotes that the term m must be deducible from the set of terms T. A solution to such a constraint is a substitution σ such that $m\sigma$ is deducible from $T\sigma$, i.e., $m\sigma \in \mathcal{F}(T\sigma)$. In MS, a constraint system is a sequence of constraints $[m_1 : T_1, \ldots, m_n : T_n]$. The sequence of constraints is used to track the dependencies between deduced messages, i.e., that message m_i must be deduced before message m_j for every $1 \leq i < j \leq n$. A solution of a constraint system is a substitution σ that solves all constraints in the system.

In our approach, a constraint $m : T$ can be interpreted as a $\mathsf{K}^{\uparrow}(m)$ premise that must be constructed, and the only allowed deconstruction chains are the ones starting from $\mathsf{K}^{\downarrow}(t)$ conclusions with $t \in T$. The relation between the constraint reduction rules in MS and our constraint-reduction rules is quite direct. It is however a bit hidden by the fact that in MS the necessary case distinctions are implicit in the quantification over all applicable rules, see the reduction procedure **P** given in [MS01, Figure 2]. In our approach, each constraint-reduction rule explicitly lists the required case distinctions. Thus each of the rules given in MS corresponds to one case in one of our constraint-reduction rules.

The *(un)* rule in MS, which unifies m to some term $t \in T$, corresponds to the case of solving a deconstruction chain by inserting an edge between its $\mathsf{K}^{\downarrow}(t)$-conclusion and the premise of the Coerce rule instance providing $\mathsf{K}^{\uparrow}(m)$. The decomposition rules from MS correspond to the case of solving a $\mathsf{K}^{\uparrow}(m)$ premise using a construction rule other than Coerce. The analysis rules from MS correspond to the case of extending a deconstruction chain with a deconstruction rule. A hidden symmetric encryption $\lceil x \rceil_y$ in MS tracks that the corresponding $\lceil x \rceil_y^{\leftrightarrow}$ has already been decomposed to avoid decomposing it multiple times. In our approach, a $\mathsf{K}^{\downarrow}$-conclusion has been decomposed if it has an outgoing edge. We therefore do not need to introduce such a special term

constructor. We also track all dependencies between the deduction of different messages explicitly. Therefore, we do not have to maintain the origination and monotonicity invariants, which are required in MS to ensure the soundness of the application of the *(elim)* rule. The *(elim)* rule replaces constraints of the form $m_j : T_j \cup \{v\}$ with $m_j : T_j$. Note that this rule is only applied if there is an earlier constraint $v : T_i$ with $i < j$ and $T_i \subseteq T_j$, which formalizes that the adversary must have constructed v by himself and thus he does not gain anything by analyzing v. In our approach, the application of the *(elim)* rule corresponds to a combined application of the rules `N4` and $\texttt{DG}_{<}$. The rule `N4` ensures that every $\mathsf{K}^{\downarrow}(m)$ conclusion occurs before a corresponding $\mathsf{K}^{\uparrow}(m)$ conclusion. See Example 32 for further intuition.

Summarizing, the difference between MS and our approach of solving message deduction constraints is that: we explicitly track all dependencies, we enumerate protocol steps lazily in a demand driven fashion, and we support the use of type assertions to enable unbounded verification in many cases. The explicit tracking of dependencies is key to our approach, as it enables reasoning about partially ordered sets of constraints. This allows us to avoid performing unnecessary case distinctions on the order of events, e.g., an up-front enumeration of an interleaving of protocol steps. It also allows us to enumerate the interleavings of protocol steps in a goal-directed fashion and it obviates the need for partial-order reduction techniques.

Security Protocol Verification Based on Backwards-Search

In the following, we compare to the main approaches for the construction of protocol security proofs based on backwards-search: Athena [SBP01], Scyther [Cre08b], CPSA [RG09], and Maude-NPA [EMM07]. We also discuss strand spaces and multiset rewriting.

Athena and Scyther The verification algorithm implemented in `scyther-proof` represents a substantial extension of the Scyther algorithm [Cre08b], which in turn is a descendant of the Athena algorithm [SBP01]. `scyther-proof`'s algorithm extends the Scyther algorithm with support for a larger range of security properties, automatic generation of type assertion soundness proofs, lemma instantiation, proof minimization, and the generation of machine-checked proofs. Like Athena and Scyther, `scyther-proof` assumes a free term algebra and represents protocols as linear role scripts. The Tamarin prover builds on the ideas underlying these three tools and extends them further. Compared to these tools, Tamarin supports both a more expressive specification language and a more expressive proof calculus. An example of this additional expressivity is Tamarin's support for specifying and reasoning about loops and Diffie-Hellman exponentiation. Another example is that Tamarin supports notions like compromising adversaries [BC10] without changing its specification language or proof calculus. The Tamarin prover moreover provides a user-friendly GUI for the interactive inspection and construction of protocol security proofs.

Strand Spaces and CPSA Similar to our approach, strand spaces [GT02] provide a security model that also explicitly represents the message deduction steps performed by the adversary. The main reasoning technique for the analysis of security protocols in the

strand space model are authentication tests [Gut04], whose use is automated by the CPSA tool [RG09]. Authentication tests exploit knowledge about the secrecy of the honest agents' long-term keys to avoid reasoning about the intruder's intermediate message deduction steps. An interesting application of authentication tests is to characterize the possible executions of a security protocol, as described in [DGT07a, DGT07b]. In our approach, we can similarly characterize the execution of a protocol by searching for all Solved-nodes in the search tree of the satisfiability claims that check the reachability of the last step of a protocol role. In some cases, authentication tests provide a stronger reasoning technique than our constraint-solving rules without trace induction. It would be interesting to investigate the increase in the expressivity of our proof calculus when incorporating authentication tests.

Multiset Rewriting and Extensions of Strand Spaces There has been work on extending strand spaces with support for session-key compromise [Gut01] and on extending them with non-monotonic state and guaranteed progress [Gut12]. Both of these extensions are special cases of our notion of labeled multiset rewriting combined with trace formulas. Note that in [Gut12], Guttman actually uses labeled multiset rewriting to model the internal state changes of protocol participants. He however does not model the adversary behavior using multiset rewriting, which leads to a more complicated structure than our dependency graphs. Note also that, in [CDL+02], Cervesato *et al.* compare the (unlabeled) multiset rewriting formalism developed in [DLMS99, CDL+99] to strand spaces. Interestingly, they state in [CDL+02, Section 6] that "strand space bundles appear to be a better notion of computation trace than rewrite sequences, and therefore analogs could be fruitfully adopted in multiset rewrite systems". Our notion of dependency graphs formalizes such an adaption, and our work confirms this statement.

Maude-NPA The Maude-NPA tool [EMM07] uses backwards narrowing to verify the unreachability of a set of symbolic attack states. In theory, Maude-NPA can therefore prove the security of a protocol for an unbounded number of sessions. The security model of Maude-NPA is inspired by strand spaces and also models protocols as linear role scripts and assumes a Dolev-Yao adversary with access to the long-term keys of a fixed set of agents. Security properties are encoded as the unreachability of a set of symbolic attack states. In contrast to many other tools, Maude-NPA however supports a rich set of equational theories, e.g., XOR, homomorphic encryption, and a Diffie-Hellman theory similar to ours. Maude-NPA also exploits the finite variant property to reason about these equational theories. The technique of folding variant narrowing [ESM12], which we use in Tamarin to compute variants, was first implemented in Maude-NPA.

Although our work is based on the Scyther tool, it turns out that the resulting approach is also closely related to the backwards narrowing approach used by Maude-NPA. The main differences between our approach and Maude-NPA are that

1. our approach supports fewer equational theories,
2. our approach uses partially ordered sets of constraints to avoid unnecessary case distinctions,
3. our approach uses deconstruction chains to reason about message deduction, and

4. our approach allows exploiting protocol-specific invariants.

As we will see below, the ability of our approach to delay the solving of constraints allows a uniform treatment of several of the state-space reduction techniques exploited in Maude-NPA [EMM11]. More precisely, the relations of the techniques listed in [EMM11, Table 1] to our approach are the following.

- *Grammars*: Maude-NPA precomputes grammars describing sets of terms that can never be known by the adversary. The resulting pruning rule is often required to achieve termination. In our approach, grammars can be seen as secrecy invariants proven by induction over the normal form traces. In our case studies, we rarely require protocol-specific secrecy invariants, but we do use invariants over the normal form traces to specify type assertions. In contrast to grammars, our invariants allow one to specify more complex conditions for the secrecy of a message, e.g., the secrecy invariant used in the verification of Scheme 2 of the TESLA protocol, which states that a key in a hash chain is secret until it expires.
- *Input*: Maude-NPA eagerly performs backwards transitions that enable receive steps of threads. Tamarin's heuristic does the same by eagerly solving In- and $\mathsf{K}^{\uparrow}$-premises.
- *Inconsistent*: Maude-NPA prunes inconsistent states as soon as possible by exploiting that (1) the adversary learns each message at most once and that (2) nonces must be sent at least once before they can occur in the adversary's knowledge. The pruning rules for exploiting (1) correspond to the rules that we use to exploit normal form condition **N3**, which states that $\mathsf{K}^{\uparrow}$- and $\mathsf{K}^{\downarrow}$-conclusions are unique. The corresponding constraint-reduction rules $\mathtt{N3}_{\uparrow}$ and $\mathtt{N3}_{\downarrow}$ are simplification rules. Together with the eager solving of In- and $\mathsf{K}^{\uparrow}$-premises, the simplification with rule $\mathtt{N3}_{\uparrow}$ ensures the early detection of contradictions stemming from condition **N3**. Note that these contradictions manifest themselves as cyclic dependencies, which are pruned using the rule $\mathtt{DG}_{\ll}$. Maude-NPA's pruning rules for exploiting (2) are in some cases stronger than our constraint-reduction rules. We could also exploit these rules in our approach, but we have not had the need for it. The effect of these stronger rules is mostly covered by our eager solving of deconstruction-chain constraints and the preference of solving $\mathsf{K}^{\uparrow}$-actions with messages containing variables of sort *fresh*.

  Note that, for subterm-convergent theories consisting only of rewriting rules $l \to r$ where r is a subterm in l at depth at most 2, requiring that each message is learned at most once seems to be sufficient to reason effectively about message deduction, i.e., might suffice for a proof of Conjecture 2. We however noticed that further normal form conditions are necessary to reason effectively about message deduction modulo subterm-convergent theories with more deeply nested terms and theories like Diffie-Hellman exponentiation, see [Sch12].
- *Subsumption*: Maude-NPA stores all states encountered during its search, and it prunes new states subsumed by states that it already encountered. In our terminology, Maude-NPA's subsumption relation under-approximates the problem of determining whether the set of solutions of one constraint system is contained

in the set of solutions of another constraint system. In our approach, we do not exploit such a subsumption relation. In our case studies, we also did not observe constraint systems in one branch of the search tree being subsumed by constraint systems in another branch. It seems that our support for partial order reasoning reduces the effectiveness of a subsumption check. Note that, if required, our approach allows sharing subproofs using protocol-specific lemmas.

- *Super-lazy Intruder*: Maude-NPA delays solving messages that are trivially deducible by the adversary in a search state. Later narrowing steps may instantiate these messages further and invalidate their property of being trivially deducible. In this case, Maude-NPA must perform a roll-back of the search. This roll-back ability is implemented using a quite complex construction. In our approach, we do not need any special support for delaying the solving of constraints. Our heuristic also implements a super-lazy intruder, as it delays the solving of $\mathsf{K}^{\uparrow}$-actions whose messages are likely to be deducible.

Abstraction Based Security Protocol Verification

There are several methods for the construction of protocol security proofs based on abstraction, e.g., [BLP06, Bla09, MV09, Möd10a, DKRS11, ARR11]. On a theoretical level, these methods are incomparable to our approach. On a practical level, we note that the scope of our approach covers many of the case studies analyzed using abstraction based methods. The hypothesis underlying abstraction-based methods is that the security property of interest is also satisfied for a simpler definition of the set of all executions of a protocol *and* that proving the security property for this more liberal definition is simpler than proving it for the precise definition. The success of the ProVerif tool [Bla09] validates this hypothesis for the verification of secrecy and authentication properties of classic security protocols.

There are three existing approaches [Möd10a, DKRS11, ARR11] for the analysis of protocols that contain loops and whose security relies on non-monotonic state. All of them are based on abstraction. The approaches in [Möd10a, DKRS11] only work for a bounded number of mutable state variables. In our approach, we do not have such a limitation. The approach in [DKRS11] only works under the assumption that the PCR register is extended at most a bounded number of times between resets, and the soundness of this assumption is proven manually. In our approach, we do not require such a manual proof, and therefore also manage to verify their model of the Envelope protocol with reboots, a problem left as future work in [DKRS11]. A key benefit of our approach over these abstraction-based approaches is that our approach allows to precisely model non-monotonic state. It therefore does not suffer from false attacks, i.e., an abstract execution that does not correspond to a concrete execution. In the cases where our proof calculus is not expressive enough, one can specify further invariants or extend the set of constraint-reduction rules to allow to express the employed correctness arguments.

The abstraction based methods [GLRV05, BAF08, KT09] support verification for an unbounded number of sessions of protocols that use Diffie-Hellman exponentiation. Blanchet *et al.* [BAF08] extend ProVerif [Bla01] to handle the property that $(x \hat{} y) \hat{} z \simeq (x \hat{} z) \hat{} y$. Goubault-Larrecq [GLRV05] accounts for this property using a Horn-

theory approach and resolution modulo AC. Küsters and Truderung [KT09] give a transformation that, given a Horn theory modeling secrecy and simple authentication properties modulo a Diffie-Hellman theory with inverses, produces a Horn theory in the free algebra, which they analyze using ProVerif. Their reduction is similar to our reduction from E_{DH} to AC, but works only for Horn clauses with ground exponents. As stated in [KT09], stronger security properties often violate this restriction. Since our approach allows for non-ground exponents, we can also find attacks where the adversary sends products, e.g., a protocol that receives a message x and leaks a secret if $\mathsf{g} \hat{} (\mathsf{a} * \mathsf{b} * x^{-1}) = \mathsf{g}$.

Using abstraction can improve the efficiency of verification and even guarantee termination for restricted classes of protocols, e.g., [BP05]. Moreover, for the unbounded verification of properties like trace equivalence, only abstraction-based methods are known, e.g., [BAF08]. The drawback of using abstraction is that one might encounter false attacks. In the context of the Tamarin prover, one could integrate abstraction based methods to automatically derive protocol-specific invariants about the messages derived by the adversary or the instances of protocol rules. These invariants might allow pruning more constraint systems, and thereby improve the effectiveness of the Tamarin prover. A starting point for such an integration is [Bla05], where ProVerif's abstraction method is explained in terms of the abstract interpretation of the execution of a multiset rewriting system.

Type Based Protocol Verification

There are several approaches that use type checking to verify security protocols, e.g., [Aba99, GJ04, BFG10]. They often require a significant amount of user-specified type annotations, which are then checked automatically. The applicability of security protocol verification by type checking depends on the difficulty of specifying the right types and on the expressivity of the type system (i.e., the verification theory). A distinguishing feature of type checking is that it allows for the modular verification of security protocols. An example of such a modular verification is [BFG10], where Bhargavan *et al.* show how to verify the source code of implementations of security protocols using a type system based on refinement types and delegating proof obligations to an SMT solver. They use type assertions as a means to specify the invariants that the protocol is expected to satisfy. Some of these invariants correspond to expected security properties, while the remaining ones are auxiliary invariants required to discharge the proof obligations that arise during type-checking. Note that there is a strong relation between type systems and abstraction [Cou97]. The relation between ProVerif's secrecy verification method and the secrecy type system of [Aba99] is for example described in [AB05].

Soundness of Type Assumptions Many security protocol verification methods assume a typed execution model where variables are always instantiated with messages according to the variable's type. It is well-known that this assumption is not always sound and might lead to missed type-flaw attacks. Heather *et al.* [HLS03] were the first to formally address the question of when a type assumption is sound. They propose a protocol transformation that ensures the soundness of type assumptions by reifying the types as tags in the messages. Li *et al.* [LYH05] optimize this tagging scheme so that it

requires fewer tags. Arapinis *et al.* [AD07] improve these results further. They propose a syntactic well-formedness condition that ensures the soundness of a type assumption for a more fine-grained type system for security protocols. Similarly, Delaune *et al.* [DKS08] propose a syntactic condition for ensuring the well-typedness of models of cryptographic APIs.

In contrast to the above approaches, our approach does not rely on a syntactic criterion or a fixed protocol transformation to prove type assertion soundness. It also does not rely on modeling the cryptographic messages in a free term algebra. Our use of type assertions is therefore more widely applicable. The price to pay for this increased scope is that we must prove type assertion soundness for each protocol individually. Fortunately, proving type assertion soundness turned out to be simple in all our case studies.

Constructing Machine-Checked Security Proofs

In the following, we compare our approach to the existing methods for the construction of machine-checked symbolic security proofs. We differentiate between interactive and automatic methods. In general, we prefer automatic methods due to their efficiency. The advantage of interactive methods is their wider scope. For example, there is no automatic approach that allows the verification of a multi-party protocol for an arbitrary number of parties, as proven interactively in [Pau97]. A distinguishing feature of our approach is that it supports the efficient construction of machine-checked protocol security proofs using automatic proof generation, where possible, and interactive proof construction, where required.

Interactive Methods for Machine-Checked Proofs The Inductive Method is one of the most successful approaches for interactively constructing machine-checked symbolic security proofs. It was initially developed by Paulson [Pau98] and later extended by Bella [Bel07] and Blanqui [Bla06]. A further extension is Sprenger and Basin's framework for the verification of families of protocols using refinement [SB10, SB12]. One of the key difficulties in the construction of a protocol security proof is reasoning about the adversary's message deduction capabilities. Several of the theorems used by the Inductive Approach to automate reasoning about message deduction crucially rely on its assumption that encryption keys are atomic. The Chain does not rely on this assumption and can be seen as a means to reason about message deduction in the more general setting that allows for composed keys. In the Inductive Approach, security properties are verified by formulating corresponding (possibly strengthened) protocol-specific invariants and proving them by induction. Formulating and proving these invariants constitutes the main effort when using this approach. In contrast, our fixed set of inference rules combined with type assertions suffices for verifying all protocols in our case studies. This is the main reason for the reduction of almost two orders of magnitude in the interactive proof construction times that we reported on in Section 5.1. Paulson reports that several days were needed for each of the three protocols analyzed in [Pau98] and the analysis of the TLS handshake protocol took six weeks [Pau99]. Note that these six weeks also include building the formal model. However, even if we assume the actual verification took only half this time, then our approach still reduces verification time by almost two orders of magnitude.

Two other approaches for the interactive proof construction of symbolic security proofs were developed using the PVS theorem prover. The first approach was developed by Evans and Schneider [ES05] based on a formalization of rank-functions [Sch97]. Our improvement in proof construction time also applies to their work, as they state that their approach requires more interaction than Paulson's inductive approach. The second approach was developed by Jacobs and Hasuo [JH09]. It is based on a variant of BAN logic [BAN90] whose inference rules are derived from a formalization of strand spaces. They do not provide proof construction times.

Automatic Generation of Machine-Checked Proofs There are two existing approaches for automatically generating machine-checked protocol security proofs. Both of them are based on abstraction. The first approach is by Goubault-Larrecq [GL10]. He models a protocol and its properties as a Horn theory T whose consistency implies that the protocol satisfies its properties. A finite model finder is then used to find a certificate (i.e., a model) for T's consistency. This certificate is machine-checked using a model checker embedded in Coq.

The secrecy properties of protocols 1-5 from Table 5.1 were also analyzed by Goubault-Larrecq in [GL10]. Note that what is referred to in [GL10] as the Kerberos protocol is in fact the simpler Denning-Sacco shared key protocol (Protocol 4 in Table 5.1). The times reported in [GL10] are in the same range as ours. The approach can be used directly with equational theories, but currently cannot handle the authentication properties considered in our work. Moreover, the approach in [GL10] does not provide machine-checked proofs of the soundness of the (non-trivial) abstractions required to model security protocols as a Horn theory. In contrast, the proofs generated by `scyther-proof` are machine-checked with respect to straightforward protocol model.

Brucker and Mödersheim describe an approach for the automatic generation of machine-checkable proofs in [BM10]. They use the OFMC model checker [MV09] to compute a fixpoint of an abstraction of the transition relation of the protocol P of interest. This fixpoint overapproximates the set of reachable states of the protocol P. It is then translated to an Isabelle proof script certifying both that this fixpoint (and hence the protocol P) does not contain an attack and that the abstraction is sound with respect to an automatically generated trace-based execution model of P formalized in the style of the Inductive Approach. This execution model is typed and uses a non-standard intruder who can only send messages matching patterns occurring in the protocol. In our previous work on decryption-chain reasoning [MCB10], we provide a detailed timing comparison to their approach. We found that proof generation times are similar, but our proof checking times are orders of magnitude faster, ranging from a factor 10 to a factor 1700.

10.2. Work Related to our Case Studies

In the following, we discuss work related to our analysis of the ISO/IEC 9798 standard presented in Chapter 6 and work related to the case studies presented in Section 9.3.

Previous Analyses of the ISO/IEC 9798 Protocols Chen and Mitchell [CM10] reported attacks based on parsing ambiguities on protocols from several standards. They identify two types of ambiguities in parsing strings involving concatenation: (1) recipients wrongly parse an encrypted string after decryption, or (2) recipients wrongly assume that a different combination of data fields was input to the digital signature or MAC that they are verifying. They show that such errors lead to attacks, and they propose modifications to the standards. Their analysis resulted in a technical corrigendum to the ISO/IEC 9798 standard [Int10b, Int09a, Int09b].

Some of the ISO/IEC 9798 protocols have been used as case studies for security protocol analysis tools. In [DNL99], the Casper/FDR tool is used to discover weaknesses in six protocols from the ISO/IEC 9798 standard. The attacks discovered are similar to our reflection and role-mixup attacks. They additionally report so-called multiplicity attacks, but these are prevented by following the specification of the time-variant parameters in Part 1 of the standard. Contrary to our findings, their analysis reports "no attack" on the 9798-2-5 and 9798-2-6 protocols as they do not consider type-flaw attacks. A role-mixup attack on the 9798-3-3 protocol was also discovered by the SATMC tool [AC08]. Neither of these two works suggested how to eliminate the detected weaknesses.

In [DDMP04], the authors verify the three-pass mutual authentication protocols that use symmetric encryption and digital signatures, i.e., 9798-2-4 and 9798-3-4. Their findings are consistent with our results.

Protocols Related to the ISO/IEC 9798 Protocols The SASL authentication mechanism from RFC 3163 [ZN01] claims to be based on Part 3 of the ISO/IEC 9798 standard. However, the SASL protocol is designed differently than the ISO/IEC protocols and is vulnerable to a man-in-the-middle attack similar to Lowe's well-known attack on the Needham-Schroeder public-key protocol. Currently, the SASL protocol is not recommended for use (as noted in the RFC). The SASL protocol only provides authentication in the presence of an eavesdropping adversary, which can also be achieved using only plaintext messages.

In the academic literature on key exchange protocols, one finds references to a Diffie-Hellman-based key exchange protocol known as "ISO 9798-3". This protocol seems to be due to [CK01, p. 464-465], where a protocol is given that is similar in structure to the three-pass mutual authentication ISO/IEC 9798 protocol based on digital signatures, where each random value n is replaced by ephemeral public keys of the form g^x. However, in the actual ISO/IEC 9798 standard, no key exchange protocols are defined, and no protocols use Diffie-Hellman exponentiation.

Unbounded Analysis of Diffie-Hellman based AKE Protocols Our analysis results for the individual protocols confirm the known results from the literature, as reported in Subsection 9.3.3. There are no alternative approaches that can automatically verify the protocols that we consider for an unbounded number of sessions with respect to the model of Diffie-Hellman (DH) exponentiation and the strong adversary models that we consider. Alternative approaches that work for subclasses of our verification problems are [LM04, Möd11, NBN11, DG12]. Lynch and Meadows [LM04] and Mödersheim [Möd11] give reductions for DH reasoning without inverses to reasoning modulo a simpler

equational theory for a restricted class of protocols. Both require that all exponents used by a protocol remain secret forever. This excludes modeling ephemeral key reveals and thus verifying recent AKE protocols. Ngo *et al.* [NBN11] propose a method for the automated construction of computational proofs for a restricted class of DH-based protocols that use signing and exponentiation with random numbers only, which excludes many recent AKE protocols. Dougherty and Guttman [DG12] propose a (manual) proof technique to verify protocols with respect to an equational theory that models the Diffie-Hellman exponents as a field. They apply their technique to construct a pen-and-paper proof of the security of the UM protocol [BWM99a]. The equational theory that they consider is a more precise model of DH exponentiation than the equational theories considered by our and other approaches. It is not known whether the unification problem is decidable for the equational theory considered in [DG12]. It is known that, for a theory that models the exponents as a ring, unification is undecidable in general [KNW03].

Unbounded Analysis of the TESLA Protocol [Arc02, BL02, BCSS11] also analyze Scheme 1 or Scheme 2 of the TESLA protocol [PCTS00]. In [Arc02], Archer analyzes a timed model of Scheme 1 in the interactive theorem prover TAME. Note that her proof required manually specifying and proving 18 additional invariants, while Tamarin proves the authentication property automatically using induction. In [BL02] Broadfoot and Lowe analyze Scheme 1 and 2 modeled using CSP and the finite state model checker FDR. They provide protocol-specific pen-and-paper proofs justifying the soundness of their reduction of the infinite state-space to the finite state-space that they analyze. In contrast to our approach, they also succeed to verify the stream authentication for Scheme 2 and a lossy channel. In [BCSS11], Basin *et al.* verify the lossless version of Scheme 2 as a case study for their framework for the interactive verification of physical protocols. They precisely model asynchronous clocks and verify the corresponding proof obligations.

11. Conclusions

Our long-term vision is that every security protocol design is accompanied by a formal correctness proof; and we consider the theory and the tools developed in this thesis a significant step towards this vision. In particular, we consider the theory underlying the Tamarin prover a promising foundation for tackling a wide range of protocol verification problems in future work. In the following, we highlight the benefits of the Tamarin prover both from a practical and a theoretical viewpoint. We also highlight the benefits of `scyther-proof` and discuss future work.

From a practical viewpoint, the Tamarin prover provides an efficient security protocol verification tool in the line of Athena [SBP01] and Scyther [Cre08b]. The Tamarin prover is the first tool that automatically analyzes, without requiring any bounds,

1. protocols whose security relies on non-monotonic state and
2. Diffie-Hellman based AKE protocols with respect to advanced adversary models, e.g., eCK security [LLM07].

The Tamarin prover supports both automatically finding attacks and performing unbounded verification. It also provides an interactive GUI that allows the user to gain further insights on a protocol of interest by inspecting and constructing proofs of its properties. Due to the undecidability of the considered problems, the Tamarin prover is not guaranteed to terminate in general. Nevertheless, in practice, it terminates for a wide range of case studies, as shown in this thesis and in Schmidt's thesis [Sch12].

From a theoretical viewpoint, our notions of satisfiability and validity claims provide an expressive formalism to specify a wide range of protocol verification problems. The simplicity of these notions benefits the development of meta-theory, as demonstrated in this thesis by the development of the verification theory underlying the Tamarin prover. Their reliance on the standard concepts of multiset rewriting, equational theories, and two-sorted first-order logic moreover allows us to exploit previous results about these concepts. In particular, we generalize the concept of strand-spaces [GT02] to multiset rewriting, and exploit the finite variant property [CLD05] and ideas from proof normal forms [Pra65] to reason about message deduction modulo equational theories. The key technique that enables Tamarin to perform unbounded verification is the use of constraint solving to enumerate the interleavings of protocol steps in a goal-directed fashion *jointly* with the solutions to the arising message deduction problems. A further benefit of the use of constraint solving is that it simplifies the extension of Tamarin. To adapt it to a new type of verification problems, one can simply add new constraint-reduction rules that formalize the additionally required correctness arguments. In this thesis, we use this approach for example to extend our basic constraint reduction rules with support for trace induction and reasoning about message deduction.

Our development of the `scyther-proof` tool [Mei12], a proof generating version of the Scyther tool, has several benefits. Its theory highlights the key ideas for constructing

unbounded security proofs using backwards search, and served as the basis for developing the theory underlying the Tamarin prover. Moreover, `scyther-proof`'s implementation demonstrates the viability of a generic approach for constructing proof generating versions of security protocol verification algorithms, i.e., the approach to prove soundness theorems for all steps that an algorithm may take and to represent the generated proofs as the composition of these theorems. In our case, we formally derive these theorems in Isabelle/HOL and thereby enable the `scyther-proof` tool to generate machine-checked proofs. As a last, and very practical benefit, we see the resulting tool set, which consists of Scyther or Tamarin for protocol analysis and `scyther-proof` for the generation of machine-checked proofs. It supports standardization committees with both falsification, for analysis in the early phases of standardization, and verification, providing objective and verifiable security guarantees in the end phases. This tool set is especially interesting for the certification of protocols at the highest evaluation levels, as it is currently pursued in several countries [MMOB10, GL10]. We demonstrate the practicality of this tool set on the ISO/IEC 9798 standard for entity authentication [Int10a], which is used as a core building block in numerous other standards. Surprisingly, we find that its most recent version still exhibits both known and new weaknesses. We therefore propose fixes and use `scyther-proof` to certify our repaired protocols. We note that the ISO/IEC working group responsible for the 9798 standard has released an updated version of the standard based on our proposed fixes.

Future Work

Intuitively, our verification theory can be seen as a language that allows protocol designers to formalize the correctness arguments underlying their protocols. To make our theory more widely applicable, we would therefore like to cover a wider range of both correctness definitions and arguments justifying a protocol's correctness. With respect to correctness arguments, we consider building support for reasoning about further classes of equational theories and improving the support for reasoning about loops and state. With respect to correctness definitions, we consider building support for the unbounded verification of process equivalences (e.g., trace equivalence) and investigating computational soundness results for variants of our theory. We discuss the first three of these directions below.

Reasoning about More Equational Theories We consider three extensions of increasing difficulty.

1. Finding normal form message deduction rules for reasoning about message deduction modulo equational theories like XOR that have the finite variant property, but are not yet supported by Tamarin.
2. Building support for reasoning about equational theories like homomorphic encryption that do not have the finite variant property, but whose unification problem is finitary.
3. Building support for reasoning about equational theories whose unification problem is undecidable in general, e.g., an equational theory that models the Diffie-Hellman

exponents as a ring.

A first step towards these extensions is proving Conjecture 2 for subterm-convergent theories. We expect this proof to highlight the relation of our constraint-reduction rules to the known decision procedures for bounded security protocol analysis, and thereby simplify the adaption of known results from the bounded case to the unbounded case. With respect to the third extension, the key observation is that well-designed security protocols use the terms built over the undecidable equational theory in very specific ways, see for example [DG12]. There is thus a chance that the concrete instances of the unification problems occurring in the verification of a concrete protocol are actually tractable. Moreover, for a well-designed protocol, there usually exists an informal correctness argument. The problem is thus how to formalize and automate the use of this correctness argument despite the undecidability of the equational theories involved. A promising approach towards such an automation is to use constraint solving to jointly enumerate the interleavings of the protocol steps, the solutions to the arising message deduction problems, *and* the solutions to the arising unification problems. The basic idea is that this may allow focusing first on the consequences from the use of the standard cryptographic operators, and only afterwards require reasoning about the solutions to the possibly undecidable unification problems.

Reasoning about Loops and Shared State To avoid missing flaws like the renegotiation attack on TLS [Far10], it is crucial to model protocols as faithfully as possible. For many protocols, such faithful models feature loops and shared state, e.g., due to key renegotiation or the use cryptographic APIs like the TPM. We show in this thesis that trace induction suffices to reason about some properties of such models. As we explain in Section 9.4, trace induction does however not always suffice. We therefore consider building special support for reasoning about common constructions like hash chains or modeling a key-value store using linear facts. We also consider extending the term algebra and the trace formulas with support for sequence numbers and timestamps. These extensions will probably require the introduction of additional constraint reduction rules. Note that for some of these extensions we should be able to adapt results from the rich field of program verification, e.g., techniques for loop invariant inference.

Unbounded Verification of Process Equivalences Process equivalences [AF01] like observational equivalence or trace equivalence cover a wider range of security properties than trace properties. Examples of properties in this increased scope are coercion resistance in electronic voting [DKR10], unlinkability in RFID protocols [ACRR10], or bidding-price-secrecy in auction protocols [DJP10]. Apart from ProVerif, which focuses on a strongly restricted set of processes [BAF08], the state-of-the art approaches for verifying such equivalences bound the number of sessions considered, e.g., [CCLD11, CCK12]. As reported in both [CCLD11] and [CCK12], the interleaving step to enumerate the bounded set of traces is costly and limits the scalability of these approaches. One approach to solve this problem is to jointly enumerate the interleavings and the solutions to the equivalence constraints, analogously to what we do in Tamarin. Thereby, one may not only increase the efficiency of these approaches, but possibly even achieve unbounded verification of process equivalences. Another option is to translate the

process equivalence notions from the applied π calculus to equivalences on multiset rewriting systems, and develop a constraint solver for proving these equivalences based on our constraint-reduction rules. This is especially interesting, as our approach, in contrast to abstraction-based approaches, allows proving both the absence and the existence of solutions, which one requires when searching for counter-examples to process equivalences.

Appendix

A. Additional Examples from Part I

In Section 5.1, we explained how we extended Isabelle's proof language to construct machine-checked security proofs based on decryption-chain reasoning. We also showed the formalization of the secrecy proof from Example 6. In this appendix, we also give the formalizations of the authentication proof from Example 7 and the type assertion soundness proof from Example 11. Both of these formalizations were generated automatically using the `scyther-proof` tool. We only took minor liberties in pretty printing them to improve their readability. These examples demonstrate not only that our mechanization of decryption-chain reasoning allows for succinct machine-checked proofs, but also that our automatic algorithm generates human-readable proofs.

Example 39. The statement and the proof of the non-injective synchronization property φ_{auth} from Example 7 are formalized in Isabelle/HOL by the proof script given in Figure A.1.

Lines 2–11 are a direct translation of the security property φ_{auth}. Note that Isabelle stores the conclusion stated in lines 7–11 under the name "?thesis" for later reference. The proof begins in line 12 by applying the TypChain rule to the message received in the

```
1:  lemma (in CR-state) client-nisynch:
2:    assumes
3:      "(i, C₂) ∈ steps(tr)"
4:      "role_th(i) = C"
5:      "σ(AV(''s''), i) ∉ Compr"
6:    shows
7:      "∃ j. role_th(j) = S ∧
8:            σ(AV(''s''), i) = σ(AV(''s''), j)  ∧
9:            NO(''k'', i)    = σ(MV(''v''), j) ∧
10:           St(i, C₁) ≺ St(j, S₁)             ∧
11:           St(j, S₂) ≺ St(i, C₂)"
12:   proof(sources "inst_σ,i(C₂-pt)")
13:     case fake thus ?thesis
14:       by (auto dest!: client-k-secrecy[OF known])
15:   next
16:     case (S₂-hash j) thus ?thesis
17:       proof(sources "inst_σ,j(S₁-pt)")
18:         case fake thus ?thesis
19:           by (auto dest!: client-k-secrecy[OF known])
20:       next
21:         case (C₁-enc i) thus ?thesis by auto
22:       qed
23:   qed
```

Figure A.1.: Proof script formalizing the non-injective synchronization proof from Example 7

second step of the client role C. This message is the instantiation in the thread i of the pattern of the role step C_2, which is available under the name "C_2-pt". The "fake" case in Line 13 corresponds to Case **(1)** from Example 7. This case is discharged by calling Isabelle's built-in tactic "auto" configured to use the previously proven secrecy lemma "client-k-secrecy" and the KNOWN rule. The "S_2-hash" case in Line 16 corresponds to Case **(2)** and denotes that some server role j sent the hash that thread i received in step C_2. In Line 17, as in the pen-and-paper proof, the TYPCHAIN rule is applied to the message received by the first message of server j. The necessary applications of the INPUT and ROLE rules are handled automatically. The "fake" case in Line 18 corresponds to Case **(2.1)** and is dealt with as before. The case "C_1-enc" in Line 21 corresponds to Case **(2.2)** and denotes that some client i sent the encryption received by the server j. In this case, the premises directly imply the conclusion, which corresponds to *syncWith(j)*. Hence, calling "auto" solves this case. ♠

Example 40. The Isabelle/HOL proof script given in Figure A.2 formalizes the definition of the type assertion CR'_{ty} for the CR' protocol together with its corresponding soundness proof from Example 11. The type assertion soundness proof is more complicated than the previous proofs due to the required bookkeeping expressed using Isabelle's locale infrastructure [Bal06].

In Lines 1-3, the "type_invariant" command that we implemented is used to define the constant CR'_typing and to create a locale CR'_typing_state that contains all the pre-instantiated variants of the TYPCHAIN rule for the CR' protocol under the assumption that the CR'_typing assertion is sound. The definition of CR'_typing only states the

```
1:  type_invariant CR'_typing for CR'
2:  where "CR'_typing =
3:    [((S',''v''), SumT (NonceT C' ''k'') (KnownT S'_1))]"

4:
5:  sublocale CR'_state ⊆ CR'_typing_state
6:  proof -
7:    have "(tr,th,σ) ∈ well-typed CR'_typing"
8:    proof(type_soundness "CR'_typing")
9:      case(S'_1-v tr' th' σ' i)
10:       note facts = this
11:       then interpret state: CR'_typing_state tr' th'
σ'
12:         by unfold_locales auto
13:       show ?case using facts
14:         by (sources "inst_σ',i(S'_1-pt)") auto
15:   qed
16:   thus "CR'_typing_state tr th σ"
17:     by unfold_locales auto
18: qed
```

Figure A.2.: Proof script formalizing the definition of the type assertion CR'_{ty} for the CR' protocol and its corresponding soundness proof from Example 11

type assertion $(S', v) \to C'.k \cup \mathsf{kn}(S'_1)$, as our formalization of type assertions always assigns the type Ag to agent variables.

In Line 5, the "sublocale" command is used to claim that every lemma proven under the assumption that the type assertion CR'_typing is sound is also a valid lemma without this assumption. This claim depends on the lemma given in Line 7, which states that every reachable state of the CR' protocol is well-typed. The corresponding proof given in Lines 8-15 relies on the tactic "type_soundness" that simplifies applications of Theorem 2. The only non-trivial case is given on Line 9. It handles the instantiation of the variable v in the first step of some thread i executing the server role in the context of some arbitrary well-typed reachable state $(\mathrm{tr}', \mathrm{th}', \sigma')$. Lines 10-12 ensure that the "sources" tactic is also applicable in this state. In Line 13-14, we discharge the proof obligations for this case by enumerating all sources for the first message received by thread i and delegating the resulting cases to Isabelle's built-in tactic "auto". In Lines 16-17, we use the lemma from Line 7 to discharge the proof obligation stemming from the "sublocale" command. ♠

B. Additional Proofs from Part II

B.1. Proofs Concerning $\rightsquigarrow^{\mathsf{basic}}_{R,E}$

In the proof that $\rightsquigarrow^{\mathsf{basic}}_{R,E}$ is correct and complete, we require the following additional lemma about the temporal order of a constraint system.

Lemma 7. *For every R,E-model (dg,θ) of a constraint system Γ, it holds that $i \lessdot_\Gamma j$ implies $\theta(i) < \theta(j)$.*

Proof. This follows from **DG1** and the definition of $\Vdash_E$. We prove it by induction over the transitive closure in the definition of $\lessdot_\Gamma$. □

Theorem (Justification of Theorem 4 on page 99)**.** *The constraint-reduction relation $\rightsquigarrow^{\mathsf{basic}}_{R,E}$ is R,E-correct and R,E-complete.*

Proof. We first prove the completeness of all rules from Figure 8.3 on page 97 and then their correctness. Note that several constraint-reduction rules require freshly renamed sets of rules. Formally, a *renaming of a rule ru away from a constraint system Γ* is a well-sorted bijection $\rho : \mathcal{V} \to \mathcal{V}$ such that $vars(ru)\rho \cap vars(\Gamma) = \varnothing$. Identifying ρ with its homomorphic extension, the freshly renamed rule is then $(ru)\rho$.

Completeness of $\rightsquigarrow^{\mathsf{basic}}_{R,E}$: Assume $\Gamma \rightsquigarrow^{\mathsf{basic}}_{R,E} \mathbf{\Gamma'}$ for an arbitrary constraint system Γ and an arbitrary set of constraint systems $\mathbf{\Gamma'}$. Let $((I,D),\theta)$ be an arbitrary R,E-model of Γ and define $dg := (I,D)$ and $M := (trace(dg),\theta)$. We must show that there exists a constraint system $\Gamma' \in \mathbf{\Gamma'}$ such that $(dg,\theta') \Vdash_E \Gamma'$ for some valuation θ'. We perform a case distinction on the constraint-reduction rules defining $\rightsquigarrow^{\mathsf{basic}}_{R,E}$, given in Figure 8.3 on page 97.

$\mathtt{S}_@$ From the rule's side-condition, we have $M \vDash_E f@i$. Hence, $\theta(i) \in idx(I)$ and there is k with $f\theta =_E acts(I_{\theta(i)})_k$. Recall that $I \in ginsts(R \cup \{\text{FRESH}\})^*$. As the FRESH rule has no action, there exists $ru \in R$ and a grounding substitution σ such that $I_{\theta(i)} = (ru)\sigma$. We define

$$\Gamma' := \{i : (ru)\rho, f \approx acts((ru)\rho)_k\} \cup \Gamma$$

where ρ is a renaming of ru away from Γ. Note that $\Gamma' \in \mathbf{\Gamma'}$ and that (dg,θ') with

$$\theta' := \theta[\rho(x) \mapsto \sigma(x)]_{x \in vars(ru)}$$

is a R,E-model of Γ as only fresh variables are updated. It is also a R,E-model of $i : (ru)\rho$ and $f \approx acts((ru)\rho)_k$. Thus, it is a R,E-model of Γ', which concludes this case.

$\mathtt{S}_{\approx}$ From the rule's side-conditions, we have $M \models_E t_1 \approx t_2$. Hence, $t_1\theta =_E t_2\theta$. Therefore, there exists $\sigma \in \mathit{unify}_E^{t_1 \approx t_2}()$ and a valuation ξ such that $\theta(x) =_E (\sigma(x))\xi$ for every $x \in \mathit{dom}(\sigma)$ and $\theta(x) = \xi(x)$ for every $x \in \mathit{vars}(\Gamma) \setminus \mathit{dom}(\sigma)$. Such a ξ exists because fresh variables in σ are not in $\mathit{vars}(\Gamma)$ and $\mathit{unify}_E^{t_1 \approx t_2}()$ is a complete set of unifiers for t_1 and t_2. The structure (dg, ξ) is a R, E-model of $\Gamma\sigma \in \mathbf{\Gamma'}$, as $\Vdash\!\models_E$ ensures that the valuation is applied to every variable.

$\mathtt{S}_{\doteq}$ From the rule's side-conditions, we have $M \models_E i \doteq j$. Hence, $\theta(i) = \theta(j)$. Thus, (dg, θ) is also a R, E-model of $\Gamma\{i/j\} \in \mathbf{\Gamma'}$, as $\Vdash\!\models_E$ ensures that the valuation is applied to every variable.

$\mathtt{S}_{\bot}$ From the rule's side condition, we have $(\mathit{dg}, \theta) \models_E \bot$, which is a contradiction and concludes this case.

$\mathtt{S}_{\neg,@}$ From the rule's side-condition, we have $M \models_E \neg(f@i)$. Moreover, from $(f@i) \in_E \mathit{as}(\Gamma)$, we obtain ri such that $(i : \mathit{ri}) \in \Gamma$ and $f \in_E \mathit{acts}(\mathit{ri})$. Hence, $\theta(i) \in \mathit{idx}(I)$ and $I_{\theta(i)} =_E \mathit{ri}\theta$ and $f\theta \in_E \mathit{acts}(I_\theta(i))$. Thus $M \models_E f@i$, which is a contradiction and concludes this case.

$\mathtt{S}_{\neg,\approx}$ From the rule's side-condition, we have $M \models_E \neg(t \approx t)$ and $t_1 =_E t_2$. Hence, $t\theta \neq_E t\theta$ and $t\theta =_E t\theta$, which is a contradiction and concludes this case.

$\mathtt{S}_{\neg,\doteq}$ From the rule's side-condition, we have $M \models_E \neg(i \doteq i)$. Hence, $\theta(i) \neq \theta(i)$, which is a contradiction and concludes this case.

$\mathtt{S}_{\neg,\lessdot}$ From the rule's side-condition, we have $M \models_E \neg(j \lessdot i)$. Hence, $\theta(j) < \theta(i)$ does not hold, which implies that either $\theta(i) < \theta(j)$ or $\theta(i) = \theta(j)$. These two cases are covered by the two constraint systems in $\mathbf{\Gamma'}$, which concludes this case.

$\mathtt{S}_{\vee}$ Completeness follows trivially from the definition of $\Vdash\!\models_E$ and the definition of $\models_E$ for $\vee$.

$\mathtt{S}_{\wedge}$ Completeness follows trivially from the definition of $\Vdash\!\models_E$ and the definition of $\models_E$ for $\wedge$.

$\mathtt{S}_{\exists}$ From the rule's side-condition, we have $M \models_E \exists x{:}s.\, \varphi$. Hence, there is $w \in \mathbf{D}_s$ such that $(\mathit{trace}(\mathit{dg}), \theta[x \mapsto w]) \models_E \varphi$. It holds that y is fresh and of sort s. Thus, $(\mathit{dg}, \theta[y \mapsto w])$ is a R, E-model of $(\{\varphi\{y/x\}\} \cup \Gamma) \in \mathbf{\Gamma'}$, which concludes this case.

$\mathtt{S}_{\forall}$ From the rule's side-condition, we have $M \models_E (\forall \vec{x}. \neg(f@i) \vee \psi)$, $(f@i)\sigma \in_E \mathit{as}(\Gamma) \cup \Gamma$, and $\mathit{dom}(\sigma) = \mathit{set}(\vec{x})$. It is easy to see that, for every trace formula χ, $M \models_E (\forall \vec{x}.\chi)$ implies $M \models_E \chi\alpha$ for every well-sorted substitution α with $\mathit{dom}(\alpha) = \mathit{set}(\vec{x})$. Note that that σ is well-sorted and $\mathit{dom}(\sigma) = \mathit{set}(\vec{x})$. Thus, we have that $M \models_E \neg(f@i)\sigma$ or $M \models_E \psi\sigma$. The first case contradicts $(f@i)\sigma \in \mathit{as}(\Gamma) \cup \Gamma$. In the second case, M is a R, E-model of $(\{\psi\sigma\} \cup \Gamma) \in \mathbf{\Gamma'}$, which concludes the completeness proof for the $\mathtt{S}_{\forall}$ rule.

$\mathtt{DG}_{\mathtt{lbl}}$ From the rule's side-condition, we have $\theta(i) \in \mathit{idx}(I)$, $\mathit{ri}\theta =_E I_{\theta(i)} =_E \mathit{ri'}\theta$. Hence, $(\mathit{dg}, \theta) \Vdash\!\models_E \mathit{ri} \approx \mathit{ri'}$, which concludes this case.

$\mathtt{DG}_{\lessdot}$ From the rule's side condition, we obtain an i such that $i \lessdot_\Gamma i$. Due to Lemma 7, this implies $\theta(i) < \theta(i)$, which is a contradiction and concludes this case.

$\mathtt{DG}_{\rightarrowtail}$ From the rule's side-condition, we have $c\theta \rightarrowtail p\theta \in D$, $i : ri \in \Gamma$, $j : ri' \in \Gamma$, $u \in idx(concs(ri))$, and $v \in idx(prems(ri'))$ such that $c = (i,u)$, $p = (j,v)$, $concs(ri)_u = f$ and $prems(ri')_v = f'$. Hence, $\theta(i), \theta(j) \in idx(I)$, $ri\theta =_E I_{\theta(i)}$, and $ri'\theta =_E I_{\theta(j)}$. Due to **DG1**, $f\theta =_E concs(I_{\theta(i)})_u =_E prems(I_{\theta(j)})_v =_E f'\theta$. Thus $(dg,\theta) \models_E f \approx f'$, which concludes this case.

$\mathtt{DG}_{\blacktriangleright}$ From the rule's side-condition, we have $i : ri \in \Gamma$, $v \in idx(prems(ri))$ such that $prems(ri)_v = f$. Hence, $\theta(i) \in idx(I)$, $ri\theta =_E I_{\theta(i)}$, and $prems(I_{\theta(i)})_v =_E f\theta$. Thus, $(dg,\theta) \models_E f \blacktriangleright_v i$, which concludes this case.

$\mathtt{S}_{\blacktriangleright}$ From the rule's side-condition, we have $\theta(j) \in idx(I)$ and $f\theta =_E prems(I_{\theta(j)})_v$. From **DG1-2**, we obtain $k \in idx(I)$ and $u \in idx(concs(I_k))$ such that $(k,u) \rightarrowtail (\theta(j),v) \in D$ and $concs(I_k)_u =_E f\theta$. As $I_k \in ginsts(R \cup \{\text{FRESH}\})$, there is $ru \in R \cup \{\text{FRESH}\}$ and a grounding substitution σ such that $I_k = ru\sigma$.

We define

$$\Gamma' := \{i : ru\rho,\ (i,u) \rightarrowtail (j,v)\} \cup \Gamma$$

where ρ is a fresh renaming of ru away from Γ. Note that $\Gamma' \in \mathbf{\Gamma'}$ and that (dg,θ') with

$$\theta' := \theta[i \mapsto k][\rho(x) \mapsto \sigma(x)]_{x \in vars(ru)}$$

is a R,E-model of Γ, as only the valuation of fresh variables is changed. Moreover, (dg,θ') is also a R,E-model of $i : ru\rho$ and $(i,u) \rightarrowtail (j,v)$. Thus, (dg,θ') is a R,E-model of Γ', which concludes this case.

$\mathtt{DG}_{\mathtt{in}}$ From the rule's side-condition, we have $(\theta(i),v) \rightarrowtail p\theta \in D$ and $(\theta(j),u) \rightarrowtail p\theta \in D$. Due to **DG2**, incoming edges are unique. Hence, $(\theta(i),v) = (\theta(j),u)$, which concludes this case.

$\mathtt{DG}_{\mathtt{out}}$ From the rule's side-condition, we have $c\theta \rightarrowtail (\theta(i),v) \in D$ and $c\theta \rightarrowtail (\theta(j),u) \in D$. Moreover, there is $i : ri \in \Gamma$ such that $u \in idx(concs(ri))$ and ri_u is a linear fact. Hence, $\theta i \in idx(I)$, $I_{\theta(i)} =_E ri\theta$, and $(ri\theta)_u$ is also a linear fact. Due to **DG3**, linear conclusions have at most one outgoing edge in dg. Hence, $(\theta(i),v) = (\theta(j),u)$ which concludes this case.

$\mathtt{DG}_{\mathtt{Fr}}$ From the rule's side-condition, we have $\theta(i), \theta(j) \in idx(I)$, $I_{\theta(i)} =_E (\multimap\!\!\rightarrow \mathsf{Fr}(m\theta))$, and $I_{\theta(j)} =_E (\multimap\!\!\rightarrow \mathsf{Fr}(m\theta))$. Due to **MSR2**, we have $I_{\theta(i)}, I_{\theta(j)} \in_E ginsts(\text{FRESH})$. Due to **DG4**, we have $\theta(i) = \theta(j)$, which concludes this case.

Correctness of $\rightsquigarrow^{\text{basic}}_{R,E}$: Assume $\Gamma \rightsquigarrow^{\text{basic}}_{R,E} \mathbf{\Gamma'}$ for an arbitrary, well-formed constraint system Γ and an arbitrary set of constraint systems $\mathbf{\Gamma'}$. Let (dg,θ) be an arbitrary R,E-model of some constraint system $\Gamma' \in \mathbf{\Gamma'}$. We must show that (dg,θ') is a R,E-model of Γ for some valuation θ'. We perform a case distinction on the constraint-reduction rules defining $\rightsquigarrow^{\text{basic}}_{R,E}$, given in Figure 8.3 on page 97.

The only non-trivial cases are the $\mathtt{S}_{\doteq}$ and $\mathtt{S}_{\approx}$ rules, as for all other rules it holds that $\Gamma \subseteq \Gamma'$.

$\mathtt{S}_{\doteq}$ Assume that $(dg, \theta) \vDash_E \Gamma\{i/j\}$. Then, $(dg, \theta[j \mapsto \theta(i)])$ is a R, E-model of Γ, as $\Vdash\!\vDash_E$ applies the valuation to every variable.

$\mathtt{S}_{\approx}$ Assume that $(dg, \theta) \vDash_E \Gamma\sigma$ for some $\sigma \in \mathit{unify}_E^{t_1 \approx t_2}()$. Then,

$$(dg, \theta[x \mapsto \sigma(x)\theta]_{x \in dom(\sigma)})$$

is a model of Γ, as $\Vdash\!\vDash_E$ applies the valuation to every variable.

This concludes the proof that the constraint-reduction relation $\rightsquigarrow_{R,E}^{\mathsf{basic}}$ is R, E-correct and R, E-complete. □

Theorem (Justification of Theorem 5 on page 100)**.** *We can construct an R, E-model for every wellformed, $\rightsquigarrow_{R,E}^{\mathsf{basic}}$-irreducible constraint system Γ.*

Proof. We construct the structure (dg, θ) as detailed in the proof sketch of Theorem 5 on page 100. In the following, we formally prove that this structure is an R, E-model of Γ. We first prove that dg is a dependency graph; i.e., satisfies conditions **DG1-4**. Then, we prove that (dg, θ) satisfies all constraints in Γ.

DG1 Let $(i, u) \rightarrowtail (j, v) \in D$. By construction, $i < j$ holds. Moreover due to our construction and **WF3**, there are facts f and f' such that $((i, u), f) \in \mathit{concs}(dg)$ and $((j, v), f') \in \mathit{prems}(dg)$. It holds that $f =_E f'$, as rules $\mathtt{DG}_{\rightarrowtail}$ and $\mathtt{S}_{\approx}$ are not applicable.

DG2 holds, as rules $\mathtt{DG}_{\blacktriangleright}$, $\mathtt{S}_{\blacktriangleright}$, and $\mathtt{DG}_{\mathtt{in}}$ are not applicable.

DG3 holds, as rule $\mathtt{DG}_{\mathtt{out}}$ is not applicable.

DG4 holds, as rule $\mathtt{DG}_{\mathtt{Fr}}$ is not applicable.

The structure (dg, θ) satisfies all node and edge constraints in Γ by construction. It satisfies the premise constraints in Γ because of condition **WF3**. We prove that $M := (\mathit{trace}(dg), \theta)$ is a model of all trace formulas in Γ by induction over their size.

- $M \vDash_E f@i$. As $\mathtt{S}_{@}$ is not applicable, $f@i \in_E \mathit{as}(\Gamma)$. Hence, there is $i : l -[\ a\ ]\!\!\rightarrow r \in \Gamma$ such that $f \in \mathit{set}(a)$. By construction, $\theta(i) \in \mathit{idx}(\mathit{trace}(dg))$ and $\mathit{trace}(dg)_{\theta(i)} = \mathit{set}(a\theta)$. Thus, $f\theta \in_E \mathit{trace}(dg)_{\theta(i)}$, which concludes this case.
- $M \vDash_E i \lessdot j$ holds by construction.
- $M \vDash_E i \doteq j$. As $\mathtt{S}_{\doteq}$ is not applicable, $i = j$. Thus, $\theta(i) =_E \theta(j)$, which concludes this case.
- $M \vDash_E t_1 \approx t_2$. As $\mathtt{S}_{\approx}$ is not applicable, $t_1 =_E t_2$. Thus, $t_1\theta =_E t_2\theta$, which concludes this case.
- $M \vDash_E \bot$. There is no such constraint in Γ, as otherwise rule $\mathtt{S}_{\bot}$ would be applicable.
- $M \vDash_E \neg(f@i)$. There is no such constraint in Γ, as otherwise rule $\mathtt{S}_{\neg,@}$ would be applicable.
- $M \vDash_E \neg(t_1 \approx t_2)$. There is no such constraint in Γ, as otherwise rule $\mathtt{S}_{\neg,\approx}$ would be applicable.

- $M \vDash_E \neg(i \doteq j)$. There is no such constraint in Γ, as otherwise rule $\mathtt{S}_{\neg,\doteq}$ would be applicable.
- $M \vDash_E \neg(i \lessdot j)$. As $\mathtt{S}_{\neg,\lessdot}$ is not applicable, we have $j \lessdot_\Gamma i$ or $j = i$. In both cases, $M \vDash_E \neg(i \lessdot j)$ holds by construction.
- $M \vDash_E \varphi_1 \wedge \varphi_2$. Rule $\mathtt{S}_\wedge$ is not applicable. Hence $\varphi_1, \varphi_2 \in \Gamma$. As φ_1 and φ_2 are smaller than $\varphi_1 \wedge \varphi_2$, the induction hypothesis applies.
- $M \vDash_E \varphi_1 \vee \varphi_2$. Rule $\mathtt{S}_\vee$ is not applicable. Hence $\varphi_1 \in \Gamma$ or $\varphi_2 \in \Gamma$. As φ_1 and φ_2 are smaller than $\varphi_1 \vee \varphi_2$, the induction hypothesis applies in either case.
- $M \vDash_E \exists x{:}s.\, \varphi$. Rule $\mathtt{S}_\exists$ is not applicable. Hence there exists a $w{:}s$ such that $\varphi\{w/x\} \in \Gamma$. Hence, we have $M \vDash_E \varphi\{w/x\}$ from the induction hypothesis. Thus, $(\mathit{trace}(dg), \theta[x \mapsto w\theta]) \vDash_E \varphi$, which concludes this case.
- $M \vDash_E \forall \vec{x}.\, \neg(f@i) \vee \psi$. Intuitively, this case holds because our construction of (dg, θ) ensures that (1) the range of θ is disjoint from the subterms of Γ and (2) all actions in $\mathit{trace}(dg)$ also exist in Γ. Thus, we can translate every substitution σ of $\vec{x}$ with $M \vDash_E (f@i)\sigma$ back to a substitution σ' of $\vec{x}$ such that $(f@i)\sigma' \in \mathit{as}(\Gamma) \cup \Gamma$, which contradicts the $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$-irreducibility of Γ. Formally, the proof works as follows.

  Let θ' be a valuation θ' such that $\theta'(x) = \theta(x)$ for all $x \notin \vec{x}$. It suffices to show that $(\mathit{trace}(dg), \theta') \vDash_E f@i$ implies $(\mathit{trace}(dg), \theta') \vDash_E \psi$. We will show this by constructing a substitution α from θ' such that $\mathit{dom}(\alpha) = \mathit{set}(\vec{x})$ and $(f@i)\alpha \in \mathit{as}(\Gamma)$. Hence, $M \vDash_E \psi\alpha$ because the reduction rule $\mathtt{S}_\forall$ is not applicable. We will show that our construction of α ensures that $M \vDash_E \psi\alpha$ implies $(\mathit{trace}(dg), \theta') \vDash_E \psi$. This will conclude this case.

  Note that **WF4** ensures that $\vec{x} \in (\mathcal{V}_{msg} \cup \mathcal{V}_{temp})^*$. Here, we focus on the special case that $\vec{x} \in \mathcal{V}^*_{msg}$. The general case can be proven analogously. We require an auxiliary definition and an auxiliary claim to prove this special case. We define $\hat{\theta} : \mathcal{T} \to \mathcal{T}$ as

$$\hat{\theta}(t) = \begin{cases} \theta^{-1}(t) & \text{if } t \in \mathit{ran}(\theta) \\ t & \text{if } t \in (PN \cup FN \cup \mathcal{V}) \setminus \mathit{ran}(\theta) \\ f(\hat{\theta}(t_1), \ldots, \hat{\theta}(t_k)) & \text{if } t = f(t_1, \ldots, t_k) \text{ for } f \in \Sigma^k \end{cases} .$$

  Intuitively, $\hat{\theta}$ lifts θ^{-1} to terms. This works because θ is injective and has a range in $PN \cup FN$. We define the substitution α as

$$\alpha := \{\hat{\theta}(\theta'(x))/x\}_{x \in \vec{x}} .$$

  Note that α is a well-sorted substitution because $\vec{x} \in \mathcal{V}^*_{msg}$. The key property of α is formalized by the following claim.

  Claim 1. *For all $t, s \in \mathcal{T}$ with $(St(t) \cup St(s)) \cap \mathit{ran}(\theta) = \varnothing$, it holds that $t\theta' =_E s\theta$ implies $t\alpha =_E s$.*

  Proof. Intuitively, this holds because $\hat{\theta}$ only changes the leaves of the terms. Formally, we prove this claim by showing that we can transform any sequence of E-equalities equating $t\theta'$ with $s\theta$ to a sequence of E-equalities equating $t\alpha$ with s. □

We now show that $(trace(dg), \theta') \vDash_E f@i$ implies $(trace(dg), \theta') \vDash_E \psi$. Assume that $(trace(dg), \theta') \vDash_E f@i$. Due to the construction of dg, we hence obtain $j : l -[\ a\ ]\!\!\rightarrow r \in \Gamma$ and $f' \in a$ such that $(i, f)\theta' =_E (j, f')\theta$. As we assume that $\vec{x} \in \mathcal{V}^*_{msg}$, this implies that $i = j$, as $\theta'(i) = \theta(i) = \theta(j)$ and θ is injective. Note that all subterms of f and f' are subterms of Γ. Moreover, $ran(\theta) \subseteq (PN \cup FN) \setminus St(\Gamma)$, which implies that $(St(f) \cup St(f')) \cap ran(\theta) = \varnothing$. From $f\theta' =_E f'\theta$, we thus have $f\alpha =_E f'$ due to Claim 1. Note that $f'@j \in as(\Gamma)$ and hence $(f@i)\alpha \in_E as(\Gamma)$. Thus, $(trace(dg), \theta) \vDash_E \psi\alpha$, as the reduction rule $\mathtt{S}_\forall$ is not applicable. Hence, $(trace(dg), \theta[x \mapsto (\alpha(x))\theta]_{x \in \vec{x}}) \vDash_E \psi$. Note that $(\alpha(x))\theta = (\hat{\theta}(\theta'(x)))\theta = \theta'(x)$ due to the definition of $\hat{\theta}$. Thus, $(trace(dg), \theta') \vDash_E \psi$, which concludes this case.

Thus, we can construct an R, E-model for every wellformed, $\rightsquigarrow^{\mathtt{basic}}_{R,E}$-irreducible constraint system Γ. □

B.2. Proofs Concerning Trace Induction

The proofs in this section use two additional definitions. For a set X, we define the *identity function* $\mathbb{I}_X : X \to X$ such that $\mathbb{I}_X(x) = x$ for all $x \in X$. For rational numbers u, a, and b, we define the function $splice_{u,a,b} : \mathbb{Q} \to \mathbb{Q}$ such that

$$splice_{u,a,b}(k) := \begin{cases} k & \text{if } k \leq u \\ u + \frac{b}{a}(k-u) & \text{if } u < k < u + a \\ k - a + b & \text{if } u + a \leq k \end{cases} .$$

Intuitively, $splice_{u,a,b}$ stretches or shrinks the interval from u to $u + a$ to the interval from u to $u + b$.

Lemma 8. *Let $u, a, b \in \mathbb{Q}$ with $|tr| \leq u$, $0 < a$, and $0 < b$. For every trace formula φ, every trace tr, and every valuation θ, we have*

$$(tr, \theta) \vDash_E \varphi \quad \textit{iff} \quad (tr, (\mathbb{I}_{\mathcal{M}} \uplus splice_{u,a,b}) \circ \theta) \vDash_E \varphi \ .$$

Proof. Note that $splice_{u,a,b}$ is a strictly monotonous bijection from $\mathbb{Q}$ to $\mathbb{Q}$. Moreover, $splice_{u,a,b}$ is the identity on $k \in \mathbb{Q}$ with $k \leq |tr|$. We use structural induction over φ.

- $f@i$: This holds because $k \in idx(tr)$ implies $k \leq |tr|$ for all $k \in \mathbb{Q}$.
- $i < j$: This follows from the strict monotonicity of $splice_{u,a,b}$.
- $\text{last}(i)$: This holds because $splice_{u,a,b}$ is the identity on all $k \in \mathbb{Q}$ with $k \leq |tr|$.
- $i \doteq j$: This holds because $splice_{u,a,b}$ is a bijection.
- $t \approx s$: This holds because θ and $(\mathbb{I}_{\mathcal{M}} \uplus splice_{u,a,b}) \circ \theta$ agree on all terms.
- $\neg\varphi$: This follows trivially from the induction hypothesis.
- $\varphi \wedge \psi$ This follows trivially from the induction hypothesis.

$\exists x.\varphi$: For $x \in \mathcal{V}_{msg}$, this follows trivially from the induction hypothesis. For $x = i \in \mathcal{V}_{temp}$, we use the following reasoning.

$$
\begin{aligned}
& (tr, \theta) \vDash_E \exists i.\varphi \\
\text{iff} \quad & \textit{there is } w \in \mathbb{Q} \textit{ such that } (tr, \theta[i \mapsto w]) \vDash_E \varphi \\
\text{iff} \quad & \textit{there is } w \in \mathbb{Q} \textit{ such that } (tr, (\mathbb{I}_{\mathcal{M}} \uplus \mathit{splice}_{u,a,b}) \circ (\theta[i \mapsto w])) \vDash_E \varphi \qquad [IH] \\
\text{iff} \quad & \textit{there is } w' \in \mathbb{Q} \textit{ such that } (tr, ((\mathbb{I}_{\mathcal{M}} \uplus \mathit{splice}_{u,a,b}) \circ \theta)[i \mapsto w']) \vDash_E \varphi \qquad [*] \\
\text{iff} \quad & (tr, (\mathbb{I}_{\mathcal{M}} \uplus \mathit{splice}_{u,a,b}) \circ \theta) \vDash_E \exists i.\varphi
\end{aligned}
$$

[$*$] We ensure that $w' = \mathit{splice}_{u,a,b}(w)$, which is possible in both directions, as $\mathit{splice}_{u,a,b}$ is a bijection.

□

Lemma (Justification of Lemma 4 on page 106)**.** *For every last-free trace formula φ, every trace tr, every set of actions A, and every valuation θ with $\forall i \in \mathit{vars}(\varphi) \cap \mathcal{V}_{temp}.\ \theta(i) \neq |tr| + 1$,*

$$(tr, \theta) \vDash_E \varphi \quad \textit{iff} \quad (tr \cdot [A], \theta) \vDash_E \text{IH}(\varphi)\ .$$

Proof. By structural induction over φ.

The cases for $i \doteq j$, $i \lessdot j$, $t \approx s$, negation, conjunction, and existential quantification over message variables are trivial. The case for $\text{last}(i)$ is trivial because φ is last-free. The two interesting cases are the ones for actions and existential quantification over timepoints. They are proven as follows.

- $(tr, \theta) \vDash_E f@i$
  iff $\theta(i) \in \mathit{idx}(tr) \wedge f\theta \in_E tr_{\theta(i)}$
  iff $\theta(i) \in \mathit{idx}(tr \cdot [A]) \wedge f\theta \in_E (tr \cdot [A])_{\theta(i)}$ [because $\theta(i) \neq |tr| + 1$]
  iff $(tr \cdot [A], \theta) \vDash_E f@i$

- $\exists i.\ \varphi$ and $i{:}\mathit{temp}$: We show both directions separately.

  $\Rightarrow$: From $(tr, \theta) \vDash_E \exists i.\ \varphi$, we obtain $w \in \mathbb{Q}$ such that $(tr, \theta[i \mapsto w]) \vDash_E \varphi$. We define $\theta' := \theta[i \mapsto w]$. If $\forall j \in \mathit{vars}(\varphi) \cap \mathcal{V}_{temp}.\ \theta'(j) \neq |tr| + 1$, then the induction hypothesis applies. Otherwise, we have $\exists j \in \mathit{vars}(\varphi) \cap \mathcal{V}_{temp}.\ \theta'(j) > |tr|$ and can therefore define

  $$a := \min\{\theta'(j) \mid j \in \mathit{vars}(\varphi) \cap \mathcal{V}_{temp},\ \theta'(j) > |tr|\} - |tr|\ ,$$

  i.e., the distance from $|tr|$ to the next temporal variable used in φ and interpreted with respect to θ'. Due to Lemma 8, we have that

  $$(tr, (\mathbb{I}_{\mathcal{M}} \uplus \mathit{splice}_{|tr|,a,2}) \circ \theta[i \mapsto w]) \vDash_E \varphi\ .$$

  Moreover, $\forall i \in \mathit{vars}(\varphi) \cap \mathcal{V}_{temp}.\ \mathit{splice}_{|tr|,a,2}(\theta'(i)) \neq |tr| + 1$. This holds because $\mathit{splice}_{|tr|,a,2}$ shifts all temporal variables $j \in \mathit{vars}(\varphi)$ with $\theta'(j) > |tr|$ to a position that is greater or equal to $|tr| + 2$. Thus, the induction hypothesis applies.

  $\Leftarrow$: From $(tr \cdot [A], \theta) \vDash_E \exists i.\ \text{IH}(\varphi) \wedge \neg\text{last}(i)$, we obtain $w \in \mathbb{Q}$ such that $w \neq |tr| + 1$ and $(tr \cdot [A], \theta[i \mapsto w]) \vDash_E \text{IH}(\varphi)$. Thus $\forall i \in \mathit{vars}(\varphi) \cap \mathcal{V}_{temp}.\ \theta(i) \neq |tr| + 1$ and the induction hypothesis applies.

□

Theorem (Justification of Theorem 6 on page 107)**.** *For every closed, last-free trace formula φ and every prefix-closed set of traces Tr, it holds that*

$$Tr \vDash_E^{\forall} \varphi \quad \textit{iff} \quad Tr \vDash_E^{\forall} \mathrm{BC}(\varphi) \wedge (\mathrm{IH}(\varphi) \Rightarrow \varphi) \ .$$

Proof. If $Tr = \varnothing$, then this statement is trivial. We thus assume that $Tr \neq \varnothing$. Note that therefore $[] \in Tr$, as Tr is prefix-closed and $[]$ is a prefix of every trace. We prove both directions separately.

⇒: From $Tr \vDash_E^{\forall} \varphi$, we have that $(tr, \theta) \vDash_E \varphi$ holds for every trace $tr \in Tr$ and every valuation θ.

We must show that, for an arbitrary trace $tr \in Tr$ and an arbitrary valuation θ, (1) $(tr, \theta) \vDash_E \mathrm{BC}(\varphi)$ and (2) $(tr, \theta) \vDash_E \mathrm{IH}(\varphi)$ implies $(tr, \theta) \vDash_E \varphi$.

Note that $[] \in Tr$. Proposition (1) holds because of Lemma 3 applied to $([], \theta) \vDash_E \varphi$. Proposition (2) trivially holds.

⇐: From $Tr \vDash_E^{\forall} \mathrm{BC}(\varphi) \wedge (\mathrm{IH}(\varphi) \Rightarrow \varphi)$, we have that, for every trace $tr \in Tr$ and every valuation θ, $(tr, \theta) \vDash_E \mathrm{BC}(\varphi)$ holds and $(tr, \theta) \vDash_E \mathrm{IH}(\varphi)$ implies $(tr, \theta) \vDash_E \varphi$.

Let the valuation θ be arbitrary. We show that $(tr, \theta) \vDash_E \varphi$ holds for all traces $tr \in Tr$ using well-founded induction over the prefix-order of the traces in Tr. Let tr be an arbitrary trace in Tr. We perform a case distinction whether $tr = []$ or $tr = tr' \cdot [A]$ for some trace tr' and some set of actions A.

If $tr = []$, then $([], \theta) \vDash_E \varphi$ holds because of Lemma 3 and $([], \theta) \vDash_E \mathrm{BC}(\varphi)$.

If $tr = tr' \cdot [A]$ for some trace tr' and some set of actions A, then we show that $(tr' \cdot [A], \theta) \vDash_E \varphi$ holds as follows. Note that $vars(\varphi) = \varnothing$ because φ is closed. Moreover, $(tr', \theta) \vDash_E \varphi$, as tr' is a prefix of $tr' \cdot [A]$. Hence, $(tr' \cdot [A], \theta) \vDash_E \mathrm{IH}(\varphi)$ due to Lemma 4. Hence, $(tr, \theta) \vDash_E \mathrm{IH}(\varphi)$, as $tr' \cdot [A] = tr$. Due to our assumptions, this implies $(tr, \theta) \vDash_E \varphi$ because $tr \in Tr$, which concludes this case. □

B.3. Proofs Concerning $\rightsquigarrow_{R,E}^{\mathtt{last}}$

Theorem (Justification of Theorem 7 on page 109)**.** *The constraint-reduction relation $\rightsquigarrow_{R,E}^{\mathtt{last}}$ is R, E-correct and R, E-complete.*

Proof. We prove this statement individually for each of the constraint-reduction rules defining $\rightsquigarrow_{R,E}^{\mathtt{last}}$, given in Figure 8.7 on page 108. The correctness of the rule $\mathtt{R}_{\mathtt{basic}}$ follows from Theorem 4. All rules other than $\mathtt{R}_{\mathtt{basic}}$ are correct because they only add further constraints. It remains to prove the completeness of a constraint-reduction step $\Gamma \rightsquigarrow_{R,E}^{\mathtt{last}} \mathbf{\Gamma'}$ for an arbitrary, well-formed constraint system Γ and an arbitrary set of constraint systems $\mathbf{\Gamma'}$.

Let $((I, D), \theta)$ be an arbitrary R, E-model of Γ and define $dg := (I, D)$ and $M := (trace(dg), \theta)$. We must show that there exists a constraint system $\Gamma' \in \mathbf{\Gamma'}$ such that $(dg, \theta') \Vdash_E \Gamma'$ for some valuation θ'. We perform a case distinction on the constraint-reduction rules defining $\rightsquigarrow_{R,E}^{\mathtt{last}}$.

$\mathtt{R}_{\mathtt{basic}}$ The completeness of this rule follows from Theorem 4.

$\mathtt{S}_{\mathrm{last},\lessdot}$ From the rule's side-condition, we have $\theta(i) = |\mathit{trace}(dg)| = |I|$, $i \lessdot_\Gamma j$, and $\theta(j) \in \mathit{idx}(I)$. Due to Lemma 7 and the definition of idx, we have $\theta(i) < \theta(j)$ and $\theta(j) \leq |I|$. Thus, $|I| = \theta(i) < \theta(j) \leq |I|$, which is a contradiction and concludes this case.

$\mathtt{S}_{\mathrm{last},\mathrm{last}}$ From the rule's side-condition, we have $\theta(i) = |\mathit{trace}(dg)| = \theta(j)$, which concludes this case.

$\mathtt{S}_{\neg,\mathrm{last}}$ From the rule's side-condition, we have that $\theta(i) \neq |\mathit{trace}(dg)| = |I|$. Hence, either $\theta(i) < |I|$ or $|I| < \theta(i)$. We perform a case distinction. If $\theta(i) < |I|$, then

$$(dg, \theta[j \mapsto |I|]) \Vdash_E \{i \lessdot j, \mathrm{last}(j)\} \cup \Gamma \ ,$$

as j is fresh. This concludes the case for $\theta(i) < |I|$. The case for $|I| < \theta(i)$ is proven analogously.

This concludes the proof that the constraint-reduction relation $\rightsquigarrow^{\mathtt{last}}_{R,E}$ is R,E-correct and R,E-complete. □

Theorem (Justification of Theorem 8 on page 109)**.** *We can construct an R,E-model for every wellformed, $\rightsquigarrow^{\mathtt{last}}_{R,E}$-irreducible constraint system Γ.*

Proof. Note that there is at most one temporal variable i with a constraint $\mathrm{last}(i) \in \Gamma$, as rule $\mathtt{S}_{\mathrm{last},\mathrm{last}}$ is not applicable. We proceed almost as in the proof of Theorem 5 on page 180. The only difference is that we ensure that the temporal variable i is instantiated with the last index of the constructed sequence of rewriting rule instances. We can ensure this because rule $\mathtt{S}_{\mathrm{last},\lessdot}$ is not applicable. We may be required to cut off some of the FRESH rule instances that we additionally introduced in the proof on page 180. The resulting structure satisfies all trace formulas because of its construction and because the rule $\mathtt{S}_{\neg,\mathrm{last}}$ is not applicable to Γ. □

B.4. Proofs Concerning $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$

In the proof that the constraint-reduction rule `N1` is complete for normal form solutions, we require the following lemma.

Lemma 9. *For every valuation θ and every term $t \in \mathcal{T}$, if t is not $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normal, then $t\theta$ is not $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normal.*

Proof. If t is not $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normal, then there is a position p in t that $\mathcal{AX}$-matches the left-hand side of a rewriting rule in Rw. The left-hand side of the same rule also Ax-matches with $t\theta$ at position p, which proves that $t\theta$ is not $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normal. □

Theorem (Justification of Theorem 11 on page 123)**.** *For every constraint-reduction step $\Gamma \rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}} \mathbf{\Gamma'}$ of a constraint system to a set of constraint systems $\mathbf{\Gamma'}$, it holds that*

(i) if Γ is wellformed, then all constraint systems in $\mathbf{\Gamma'}$ are wellformed and

(ii) $nsols_{P,ND,\mathcal{R},\mathcal{AX}}(\Gamma) =_{\mathcal{AX}} \bigcup_{\Gamma' \in \mathbf{\Gamma}'} nsols_{P,ND,\mathcal{R},\mathcal{AX}}(\Gamma')$

provided that the assumptions **A1-7** *hold.*

Proof. We prove this statement individually for each of the constraint-reduction rules defining $\rightsquigarrow^{msg}_{P,ND,\mathcal{R},\mathcal{AX}}$, which are given in Figure 8.12 on page 122. We show the correctness of the rule $\mathtt{R}_{\mathtt{induct}}$ with respect to normal form solutions analogously to the proof of Theorem 7 on page 184. All rules other than $\mathtt{R}_{\mathtt{induct}}$ are correct because they only add further constraints.

It remains to prove the completeness with respect to normal form solutions of a constraint-reduction step $\Gamma \rightsquigarrow^{msg}_{P,ND,\mathcal{R},\mathcal{AX}} \mathbf{\Gamma}'$ for an arbitrary, well-formed constraint system Γ and an arbitrary set of constraint systems $\mathbf{\Gamma}'$. Let $((I, D), \theta)$ be an arbitrary normal form model of Γ and define $dg := (I, D)$ and $M := (trace(dg), \theta)$. We must show that there exists a constraint system $\Gamma' \in \mathbf{\Gamma}'$ such that $(dg, \theta') \models_E \Gamma'$ for some valuation θ'. We perform a case distinction on the constraint-reduction rules defining $\rightsquigarrow^{msg}_{P,ND,\mathcal{R},\mathcal{AX}}$.

$\mathtt{R}_{\mathtt{induct}}$ We show the completeness of this rule analogously to the proof of Theorem 7 on page 184. The restriction to multiset rewriting rules from $\lceil P \rceil^{\mathcal{R}}_{\mathcal{AX}} \cup \{\text{ISEND}\}$ is sound because of Assumption **A2** and the side condition of $\mathtt{R}_{\mathtt{induct}}$.

N1 From the rule's side-condition, we have $(i : ri) \in \Gamma$ such that ri is not $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normal. Hence, $\theta(i) \in idx(I)$ and $ri\theta =_{\mathcal{AX}} I_{\theta(i)}$. Due to Lemma 9, $I_{\theta(i)}$ is not $\downarrow^{\mathcal{R}}_{\mathcal{AX}}$-normal, which contradicts Condition **N1** of Assumption **A6** and thus concludes this case.

$\mathtt{S}_{\mathsf{K}^{\uparrow}_{\blacktriangleright}}$ From the rule's side-condition, we have $\theta(i) \in idx(I)$ and $\mathsf{K}^{\uparrow}(t\theta) =_{\mathcal{AX}} prems(I_{\theta(i)})_v$. From **DG1-2** and the structure of the rules in P and ND, we obtain $k \in idx(I)$ such that $concs(I_k)_1 =_{\mathcal{AX}} \mathsf{K}^{\uparrow}(t\theta)$ and $k < \theta(i)$. Moreover, there is $ru \in ND$ and a grounding substitution σ such that $I_k = ru\sigma$. Hence, $\mathsf{K}^{\uparrow}(t\theta) \in_{\mathcal{AX}} set(acts(I_k))$, as we assume that $acts(ru) = [concs(ru)_1]$ for all rules $ru \in ND$. Thus, $(dg, \theta[j \mapsto k])$ is a normal form model of $\{\mathsf{K}^{\uparrow}(t)@j, j \lessdot i\} \cup \Gamma$, which concludes this case.

$\mathtt{S}_{\mathsf{K}^{\uparrow}_{@}}$ Analogous to the completeness proof of the rule $\mathtt{S}_{@}$ exploiting that only rules in ND contain $\mathsf{K}^{\uparrow}$-actions.

$\mathtt{S}_{\mathsf{K}^{\downarrow}_{\blacktriangleright}}$ From the rule's side-condition, we have $\theta(j) \in idx(I)$ and $\mathsf{K}^{\downarrow}(t\theta) =_{\mathcal{AX}} prems(I_{\theta(j)})_v$. We prove that assumptions **A1-5** imply the deconstruction-chain property, as formalized in Lemma 5 on page 117. We thus obtain $k \in idx(I)$ such that $I_k = (\text{IRECV})\sigma$ for some grounding grounding substitution σ and $(k, 1) \dashrightarrow_{dg} (\theta(j), v)$. The structure

$$(dg, \theta[i \mapsto k][y \mapsto \sigma(x)]) \ ,$$

where x is the variable x from the IRECV rule, is a normal form model of $\{i : \mathsf{Out}(y) \multimap[]\to \mathsf{K}^{\downarrow}(y), (i, 1) \dashrightarrow (\theta(j), v)\} \cup \Gamma$, which concludes this case.

$\mathtt{S}_{\dashrightarrow}$ From the rule's left-hand-side, we obtain $(j, v) = c$ such that $(\theta(j), v) \dashrightarrow_{dg} p\theta$. Due to the definition of $\dashrightarrow_{dg}$, $(\theta(j), v)$ is a $\mathsf{K}^{\downarrow}$-conclusion and either (a) $(\theta(j), v) \nrightarrow p\theta \in D$ or (b) there is a premise (k, u) such that $(\theta(j), v) \nrightarrow (k, u) \in D$ and $(k, 1) \dashrightarrow_{dg} p\theta$.

In Case (a), the structure (dg, θ) is a normal form model of the constraint system $(\{c \twoheadrightarrow p\} \cup \Gamma) \in \mathbf{\Gamma}'$. In Case (b), there is $ru \in ND$ and a grounding substitution σ such that $ru\sigma = I_k$ because only rules in ND feature $\mathsf{K}^{\downarrow}$-premises. We define

$$\Gamma' := \{i : ru\rho,\ c \twoheadrightarrow (i,u),\ (i,1) \dashrightarrow p,\ \Gamma\}$$

where ρ is a renaming of ru away from Γ. Note that $\Gamma' \in \mathbf{\Gamma}'$. The structure

$$(dg, \theta[i \mapsto k][\rho(x) \mapsto \sigma(x)]_{x \in dom(ru)})$$

is a normal form model of Γ', which concludes this case.

`N2` From the rule's side-condition, we have $\theta(i) \in idx(I)$ and

$$I_{\theta(i)} =_{\mathcal{AX}} (\mathsf{K}^{\downarrow}(\langle t_1, t_2\rangle) \multimap [\ \mathsf{K}^{\uparrow}(\langle t_1, t_2\rangle)\ ] \mapsto \mathsf{K}^{\uparrow}(\langle t_1, t_2\rangle))\theta\ .$$

Hence, $I_{\theta(i)}$ is an instance of the Coerce rule that deduces a pair. This contradicts Condition **N2** of Assumption **A6** and thus concludes this case.

`N3`$_{\uparrow}$ From the rule's side-condition, we have $\theta(i), \theta(j) \in idx(I)$ and $\mathsf{K}^{\uparrow}(t) \in_{\mathcal{AX}} set(acts(I_{\theta(i)}))$ and $\mathsf{K}^{\uparrow}(t) \in_{\mathcal{AX}} set(acts(I_{\theta(j)}))$. From the assumption that the rules in P do contain neither $\mathsf{K}^{\uparrow}$-facts nor $\mathsf{K}^{\uparrow}$-facts, we have $I_{\theta(i)}, I_{\theta(j)} \in ginsts(ND)$. Due to the structure of the rules in ND, we moreover have $concs(I_{\theta(i)})_1 =_{\mathcal{AX}} \mathsf{K}^{\uparrow}(t\theta) =_{\mathcal{AX}} concs(I_{\theta(j)})_1$. From Condition **N3** of Assumption **A6**, we thus have $\theta(i) = \theta(j)$, which concludes this case.

`N3`$_{\downarrow}$ From the rule's side-condition, we have $\theta(i), \theta(j) \in idx(I)$, and $concs(I_{\theta(i)})_1 =_{\mathcal{AX}} \mathsf{K}^{\downarrow}(t\theta) =_{\mathcal{AX}} concs(I_{\theta(j)})_1$. From Condition **N3** of Assumption **A6**, we thus have $\theta(i) = \theta(j)$, which concludes this case.

`N4` From the rule's side-condition, we have $\theta(i), \theta(j) \in idx(I)$ and $\mathsf{K}^{\downarrow}(t\theta) =_{\mathcal{AX}} concs(I_{\theta(i)})_1$. We moreover have $\mathsf{K}^{\uparrow}(t) \in_{\mathcal{AX}} set(acts(I_{\theta(j)}))$, which implies $\mathsf{K}^{\downarrow}(t\theta) =_{\mathcal{AX}} concs(I_{\theta(i)})_1$ due to the structure of the rules in P and ND. From Condition **N4** of Assumption **A6**, we thus have $\theta(i) < \theta(j)$, which concludes this case.

This concludes the proof that the constraint-reduction relation $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$ is correct and complete with respect to normal form solutions. □

Theorem (Justification of Theorem 12 on page 126)**.** *We can construct a normal form solution for every wellformed and* $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$*-irreducible constraint system* Γ*, provided that the assumption* **A1-7** *hold.*

Proof. We show this in two steps. We first convert the constraint system Γ to a constraint system Γ' that contains no deconstruction chains, no $\mathsf{K}^{\uparrow}$-actions without an associated rule instance, and no $\mathsf{K}^{\uparrow}$-premises without an incoming edge. We ensure that the normal form solutions of Γ' are also normal form solutions of Γ'. Then, we construct a solution to Γ' according to the proof of Theorem 8 on page 185 and show that the resulting solution constitutes a normal form solution of Γ.

The conversion of Γ to Γ' involves four transformations. First, we replace every message variable x with the public-name variable $x{:}pub$. We call the resulting constraint

system Γ_1. Its solutions are obviously contained in the ones of Γ. Second, for every action $\mathsf{K}^{\uparrow}(x{:}pub)@i \in \Gamma_1$, we introduce a constraint $i : \neg[\, \mathsf{K}^{\uparrow}(x{:}pub) \,]\!\!\rightarrow [\mathsf{K}^{\uparrow}(x{:}pub)]$. We call the resulting constraint system Γ_2. Its solutions are obviously contained in the ones of Γ_1. Moreover, Γ_2 does not contain any actions without an associated rule constraint, as otherwise one of the rules $\mathtt{S}_{@}$ and $\mathtt{S}_{\mathsf{K}^{\uparrow}@}$ would be applicable to Γ. Third, note that for every premise $(\mathsf{K}^{\uparrow}(m) \blacktriangleright_v i) \in \Gamma_2$ there exists an action $(\mathsf{K}^{\uparrow}(m)@j) \in_E \Gamma_2$ such that $j \lessdot_{\Gamma_2} i$. For each such pair of a $\mathsf{K}^{\uparrow}$-premise and its corresponding action, we introduce an edge $(j, 1) \rightarrowtail (i, v)$ and call the resulting constraint system Γ_3. The solutions of Γ_3 are obviously contained in the ones of Γ_2. Moreover, Γ' is wellformed according to conditions **WF1-4** and $\rightsquigarrow^{\mathtt{last}}_{\lceil P \rceil^{\mathcal{R}}_{\mathcal{AX}} \cup ND, \mathcal{AX}}$-irreducible. In the fourth and last step, we remove all deconstruction chains from Γ_3 and call the resulting constraint system Γ'. The proof that this does not introduce any new solutions with respect to Γ_3 exploits that the rules $\mathtt{S}_{\rightarrow}$, $\mathtt{N4}$, and $\mathtt{DG}_{\lessdot}$ are not applicable. Its key element is the observation that a $\rightsquigarrow^{\mathtt{msg}}_{P,ND,\mathcal{R},\mathcal{AX}}$-irreducible cannot contain a deconstruction chain ending in a message variable x. Because of condition **MSR3**, every message variable occurring in a conclusion of a multiset rewriting rule ru is covered by a variable in one of ru's premises. Thus, we can show by induction over the order $\lessdot_{\Gamma'}$ that the existence of a deconstruction chain ending in a message variable x implies the existence of an earlier $\mathsf{K}^{\uparrow}(x)$-premise. This contradicts the fact that the rules $\mathtt{N4}$ and $\mathtt{DG}_{\lessdot}$ are not applicable to Γ'.

Thus, we obtain a constraint system Γ' whose normal form solutions are also normal form solutions of Γ. As Γ' is wellformed and $\rightsquigarrow^{\mathtt{last}}_{\lceil P \rceil^{\mathcal{R}}_{\mathcal{AX}} \cup ND, \mathcal{AX}}$-irreducible, we can construct a solution according to the proof of Theorem 8 on page 185. It is easy to show that the constructed solution is a normal form solution, as the rules $\mathtt{N1}$, $\mathtt{N2}$, $\mathtt{N3}_{\uparrow}$, $\mathtt{N3}_{\downarrow}$, and $\mathtt{N4}$ are not applicable to Γ'. □

Bibliography

[AB05] M. Abadi and B. Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. *Journal of the ACM*, 52(1):102–146, January 2005.

[Aba99] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, September 1999.

[AC08] A. Armando and L. Compagna. SAT-based model-checking for security protocols analysis. *International Journal of Information Security*, 7(1):3–32, 2008.

[ACC+08] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In V. Shmatikov, editor, *FMSE*, pages 1–10. ACM, 2008.

[ACRR10] M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*, pages 107–121, 2010.

[AD07] M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In *Proceedings of the 27th international conference on Foundations of software technology and theoretical computer science*, FSTTCS'07, pages 376–387, Berlin, Heidelberg, 2007. Springer-Verlag.

[AF01] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In C. Hankin and D. Schmidt, editors, *POPL*, pages 104–115. ACM, 2001.

[AN96] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6 –15, 1996.

[AR10] K. Ables and M. D. Ryan. Escrowed data and the digital envelope. In A. Acquisti, S. W. Smith, and A.-R. Sadeghi, editors, *TRUST*, volume 6101 of *Lecture Notes in Computer Science*, pages 246–256. Springer, 2010.

[Arc02] M. Archer. Proving correctness of the basic TESLA multicast stream authentication protocol with TAME. In *In Workshop on Issues in the Theory of Security*, pages 14–15, 2002.

[ARR11] M. Arapinis, E. Ritter, and M. D. Ryan. StatVerif: Verification of stateful processes. In *Proceedings of the 24rd IEEE Computer Security Foundations Symposium*, pages 33–47, 2011.

[BAF08] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[Bal06] C. Ballarin. Interpretation of locales in Isabelle: Theories and proof contexts. In J. M. Borwein and W. M. Farmer, editors, *MKM*, volume 4108 of *Lecture Notes in Computer Science*, pages 31–43. Springer, 2006.

[BAN90] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

[BC10] D. Basin and C. Cremers. Modeling and analyzing security in the presence of compromising adversaries. In *Computer Security - ESORICS 2010*, volume 6345 of *LNCS*, pages 340–356. Springer, 2010.

[BCM12] D. Basin, C. Cremers, and S. Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. In P. Degano and J. D. Guttman, editors, *Principles of Security and Trust - First International Conference, POST 2012, Proceedings*, volume 7215 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2012.

[BCM13] D. A. Basin, C. Cremers, and S. Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. *Journal of Computer Security*, 2013. to appear.

[BCSS11] D. Basin, S. Capkun, P. Schaller, and B. Schmidt. Formal reasoning about physical properties of security protocols. *ACM Trans. Inf. Syst. Secur.*, 14(2):16:1–16:28, September 2011.

[Bel07] G. Bella. *Formal Correctness of Security Protocols (Information Security and Cryptography)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[BFG10] K. Bhargavan, C. Fournet, and A. D. Gordon. Modular verification of security protocol code by typing. In M. V. Hermenegildo and J. Palsberg, editors, *POPL*, pages 445–456. ACM, 2010.

[BGHZ11] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011.

[BGZ09] G. Barthe, B. Grégoire, and S. Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009*, pages 90–101. ACM, 2009.

[BJST08] B. Blanchet, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Computationally sound mechanized proofs for basic and public-key Kerberos. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'08)*, pages 87–99, Tokyo, Japan, March 2008. ACM.

[BL02] P. J. Broadfoot and G. Lowe. Analysing a stream authentication protocol using model checking. In D. Gollmann, G. Karjoth, and M. Waidner, editors, *ESORICS*, volume 2502 of *Lecture Notes in Computer Science*, pages 146–161. Springer, 2002.

[Bla01] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE, 2001.

[Bla05] B. Blanchet. Security Protocols: From Linear to Classical Logic by Abstract Interpretation. *Information Processing Letters*, 95(5):473–479, September 2005.

[Bla06] F. Blanqui. An Isabelle formalization of protocol-independent secrecy with an application to e-commerce. *CoRR*, abs/cs/0610069, 2006. available as http://arxiv.org/abs/cs/0610069.

[Bla08] B. Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, 2008.

[Bla09] B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[BLP06] L. Bozga, Y. Lakhnech, and M. Périn. Pattern-based abstraction for verifying secrecy in protocols. *STTT*, 8(1):57–76, 2006.

[BM10] A. Brucker and S. Mödersheim. Integrating automated and interactive protocol verification. In P. Degano and J. Guttman, editors, *Formal Aspects in Security and Trust*, volume 5983 of *Lecture Notes in Computer Science*, pages 248–262. Springer Berlin / Heidelberg, 2010.

[BMV05] D. A. Basin, S. Mödersheim, and L. Viganò. Ofmc: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[BP98] G. Bella and L. C. Paulson. Kerberos version 4: Inductive analysis of the secrecy goals. In J.-J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann, editors, *ESORICS*, volume 1485 of *Lecture Notes in Computer Science*, pages 361–375. Springer, 1998.

[BP05] B. Blanchet and A. Podelski. Verification of cryptographic protocols: tagging enforces termination. *Theor. Comput. Sci.*, 333:67–90, 2005.

[BW99] S. Berghofer and M. Wenzel. Inductive datatypes in HOL – lessons learned in formal-logic engineering. In Y. Bertot, G. Dowek, L. Théry, A. Hirschowitz, and C. Paulin, editors, *Theorem Proving in Higher Order Logics*, volume 1690 of *Lecture Notes in Computer Science*, pages 839–839. Springer Berlin / Heidelberg, 1999.

[BWM99a] S. Blake-Wilson and A. Menezes. Authenticated Diffie-Hellman key agreement protocols. In *Selected Areas in Cryptography*, volume 1556 of *LNCS*, pages 630–630. Springer, 1999.

[BWM99b] S. Blake-Wilson and A. Menezes. Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol. In *Proceedings of the 2nd International Workshop on Practice and Theory in Public Key Cryptography*, pages 154–170. Springer, 1999.

[CCK12] R. Chadha, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocols. In H. Seidl, editor, *ESOP*, volume 7211 of *Lecture Notes in Computer Science*, pages 108–127. Springer, 2012.

[CCLD11] V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: negative tests and non-determinism. In *Proceedings of 18th ACM Conference on Computer and Communications Security, CCS 2011*, pages 321–330, 2011.

[CDL+99] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of CSFW 1999*, pages 55–69. IEEE, 1999.

[CDL+02] I. Cervesato, N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. A comparison between strand spaces and multiset rewriting for security protocol analysis. In *ISSS*, volume 2609 of *LNCS*, pages 356–383. Springer, 2002.

[CK01] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001.

[CLD05] H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. *Term Rewriting and Applications*, pages 294–307, 2005.

[CLN09] C. J. Cremers, P. Lafourcade, and P. Nadeau. Comparing state spaces in automatic protocol analysis. In *Formal to Practical Security*, volume 5458/2009 of *Lecture Notes in Computer Science*, pages 70–94. Springer Berlin / Heidelberg, 2009.

[CM05] C. Cremers and S. Mauw. Operational semantics of security protocols. In S. Leue and T. Systä, editors, *Scenarios: Models, Transformations and Tools, International Workshop, Dagstuhl Castle, Germany, September 7-12, 2003, Revised Selected Papers*, volume 3466 of *Lecture Notes in Computer Science*. Springer, 2005.

[CM10] L. Chen and C. J. Mitchell. Parsing ambiguities in authentication and key establishment protocols. *International Journal of Electronic Security and Digital Forensics*, 3:82–94, 2010.

[CMdV06] C. Cremers, S. Mauw, and E. de Vink. Injective synchronisation: an extension of the authentication hierarchy. *Theoretical Computer Science*, pages 139–161, 2006.

[CMU11] S. Chatterjee, A. Menezes, and B. Ustaoglu. A generic variant of NIST's KAS2 key agreement protocol. In *Proceedings of the 16th Australasian conference on Information security and privacy (ACISP'11)*, pages 353–370. Springer, 2011.

[Cou97] P. Cousot. Types as abstract interpretations, invited paper. In *Conference Record of the Twentyfourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 316–331, Paris, France, January 1997. ACM Press, New York, NY.

[Cre06] C. Cremers. *Scyther - Semantics and Verification of Security Protocols.* Ph.D. dissertation, Eindhoven University of Technology, 2006.

[Cre08a] C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, volume 5123/2008 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.

[Cre08b] C. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 119–128, New York, NY, USA, 2008. ACM.

[Cre10] C. Cremers. Session-StateReveal is stronger than eCK's EphemeralKeyReveal: Using automatic analysis to attack the NAXOS protocol. *International Journal of Applied Cryptography (IJACT)*, 2:83–99, 2010.

[Cre11] C. Cremers. Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 80–91. ACM, 2011.

[DDMP04] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. Abstraction and refinement in protocol derivation. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 30–45. IEEE Comp. Soc., June 2004.

[DG12] D. J. Dougherty and J. D. Guttman. Symbolic protocol analysis for Diffie-Hellman. *CoRR*, abs/1202.2168, 2012.

[DGT07a] S. Doghmi, J. Guttman, and F. Thayer. Searching for shapes in cryptographic protocols. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *LNCS*, pages 523–537. Springer, 2007.

[DGT07b] S. Doghmi, J. Guttman, and J. Thayer. Skeletons, homomorphisms, and shapes: Characterizing protocol executions. *ENTCS*, 173(0):85 – 102, 2007.

[DJP10] N. Dong, H. L. Jonker, and J. Pang. Analysis of a receipt-free auction protocol in the applied pi calculus. In *Formal Aspects in Security and Trust 2010*, volume 6561, pages 223–238, 2010.

[DKR10] S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols: A taster. In D. Chaum, M. Jakobsson, R. L. Rivest, P. Y. A. Ryan, J. Benaloh, M. Kutyłowski, and B. Adida, editors, *Towards Trustworthy Elections – New Directions in Electronic Voting*, volume 6000 of *Lecture Notes in Computer Science*, pages 289–309. Springer, May 2010.

[DKRS10a] S. Delaune, S. Kremer, M. D. Ryan, and G. Steel. A formal analysis of authentication in the TPM. In *Formal Aspects in Security and Trust 2010*, pages 111–125, 2010.

[DKRS10b] S. Delaune, S. Kremer, M. D. Ryan, and G. Steel. Website – some verifications of protocols based on TPM state registers, 2010. Retrieved in September 2012 from http://www.lsv.ens-cachan.fr/~delaune/TPM-PCR/.

[DKRS11] S. Delaune, S. Kremer, M. D. Ryan, and G. Steel. Formal analysis of protocols based on TPM state registers. In *Proceedings of the 24rd IEEE Computer Security Foundations Symposium*, pages 66–80, 2011.

[DKS08] S. Delaune, S. Kremer, and G. Steel. Formal analysis of PKCS#11. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium*, pages 331–344, 2008.

[DLM04] N. A. Durgin, P. Lincoln, and J. C. Mitchell. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

[DLMS99] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In N. Heintze and E. Clarke, editors, *Proceedings of the Workshop on Formal Methods and Security Protocols — FMSP, Trento, Italy*, 1999.

[DNL99] B. Donovan, P. Norris, and G. Lowe. Analyzing a library of security protocols using Casper and FDR. In *Proc. of the Workshop on Formal Methods and Security Protocols*, 1999.

[DY81] D. Dolev and A. C.-C. Yao. On the security of public key protocols (extended abstract). In *FOCS*, pages 350–357. IEEE, 1981.

[EMM07] S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, pages 1–50, 2007.

[EMM11] S. Escobar, C. Meadows, and J. Meseguer. State space reduction in the Maude-NRL protocol analyzer. *CoRR*, abs/1105.5282, 2011. http://arxiv.org/abs/1105.5282.

[ES05] N. Evans and S. A. Schneider. Verifying security protocols with PVS: widening the rank function approach. *J. Log. Algebr. Program.*, 64(2):253–284, 2005.

[ESM12] S. Escobar, R. Sasse, and J. Meseguer. Folding variant narrowing and optimal variant termination. *Journal of Logic and Algebraic Programming*, 81(78):898 – 928, 2012.

[Eur09] European Payments Council. Guidelines on algorithms usage and key management. Technical report, 2009. EPC342-08 Version 1.1.

[Far10] S. Farrell. Why didn't we spot that? *IEEE Internet Computing*, 14(1):84–87, January 2010.

[FKS11] C. Fournet, M. Kohlweiss, and P.-Y. Strub. Modular code-based cryptographic verification. In *Proceedings of 18th ACM Conference on Computer and Communications Security, CCS 2011*, pages 341–350, 2011.

[GJ04] A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12:435–483, 2004.

[GJM99] J. A. Garay, M. Jakobsson, and P. D. MacKenzie. Abuse-free optimistic contract signing. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 449–466. Springer, 1999.

[GL10] J. Goubault-Larrecq. Finite models for formal security proofs. *Journal of Computer Security*, 18(6):1247–1299, 2010.

[GLRV05] J. Goubault-Larrecq, M. Roger, and K. Verma. Abstraction and resolution modulo AC: How to verify Diffie-Hellman-like protocols automatically. *Journal of Logic and Algebraic Programming*, 64(2):219–251, 2005.

[Gor89] M. J. C. Gordon. Mechanizing programming logics in higher order logic. In G. Birtwistle and P. A. Subrahmanyam, editors, *Current trends in hardware verification and automated theorem proving*, pages 387–439. Springer-Verlag New York, Inc., 1989.

[GT00] J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proc. 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 24–34, 2000.

[GT02] J. D. Guttman and F. J. Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, June 2002. Conference version appeared in *IEEE Symposium on Security and Privacy*, May 2000.

[Gut01] J. D. Guttman. Key compromise, strand spaces, and the authentication tests. *Electr. Notes Theor. Comput. Sci.*, 45:141–161, 2001.

[Gut04] J. D. Guttman. Authentication tests and disjoint encryption: A design method for security protocols. *Journal of Computer Security*, 12:409–433, 2004.

[Gut12] J. D. Guttman. State and progress in strand spaces: Proving fair exchange. *Journal of Automated Reasoning*, 48(2):159–195, 2012.

[Her06] J. Herzog. Applying protocol analysis to security device interfaces. *IEEE Security and Privacy*, 4:84–87, 2006.

[HLS03] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[Int98] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-3:1998, Information technology – Security techniques – Entity Authentication – Part 3: Mechanisms using digital signature techniques, 1998. Second edition.

[Int99] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-4:1999, Information technology – Security techniques – Entity Authentication – Part 3: Mechanisms using a cryptographic check function, 1999. Second edition.

[Int08] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-2:2008, Information technology – Security techniques – Entity Authentication – Part 2: Mechanisms using symmetric encipherment algorithms, 2008. Third edition.

[Int09a] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-3:1998/Cor.1:2009, Information technology – Security techniques – Entity Authentication – Part 3: Mechanisms using digital signature techniques. Technical Corrigendum 1, 2009.

[Int09b] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-4:1999/Cor.1:2009, Information technology – Security techniques – Entity Authentication – Part 3: Mechanisms using a cryptographic check function. Technical Corrigendum 1, 2009.

[Int10a] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-1:2010, Information technology – Security techniques – Entity Authentication – Part 1: General, 2010. Third edition.

[Int10b] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-2:2008/Cor.1:2010, Information technology – Security techniques – Entity Authentication – Part 2: Mechanisms using symmetric encipherment algorithms. Technical Corrigendum 1, 2010.

[Int10c] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-3:1998/Amd.1:2010, Information technology – Security techniques – Entity Authentication – Part 3: Mechanisms using digital signature techniques. Amendment 1, 2010.

[ITU03] ITU-T. Recommendation H.235 - Security and encryption for H-series (H.323 and other H.245-based) multimedia terminals, 2003.

[JH09] B. Jacobs and I. Hasuo. Semantics and logic for security protocols. *Journal of Computer Security*, 17(6):909–944, 2009.

[JKL04] I. R. Jeong, J. Katz, and D. H. Lee. One-round protocols for two-party authenticated key exchange. In *Applied Cryptography and Network Security*, pages 220–232. Springer, 2004.

[JKL08] I. R. Jeong, J. Katz, and D. H. Lee. One-round protocols for two-party authenticated key exchange (full), 2008. http://www.cs.umd.edu/~jkatz/papers/1round_AKE.pdf.

[JV96] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *ASIACRYPT 1996*, volume 1163 of *LNCS*, pages 36–49. Springer, 1996.

[KNW03] D. Kapur, P. Narendran, and L. Wang. An E-unification algorithm for analyzing protocols that use modular exponentiation. In *Rewriting Techniques and Applications*, pages 165–179. Springer, 2003.

[Kra05] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *Advances in Cryptology–CRYPTO 2005*, volume 3621 of *LNCS*, pages 546–566. Springer, 2005.

[KS12] R. Künnemann and G. Steel. YubiSecure? formal security analysis results for the Yubikey and YubiHSM. In *Proc. of the 8th Workshop on Security and Trust Management (STM 2012)*, Pisa, Italy, September 2012.

[KT09] R. Küsters and T. Truderung. Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium*, pages 157–171. IEEE Computer Society, 2009.

[Lau05] P. Laud. Secrecy types for a simulatable cryptographic library. In V. Atluri, C. Meadows, and A. Juels, editors, *ACM Conference on Computer and Communications Security*, pages 26–35. ACM, 2005.

[LLM07] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In *Provable Security*, volume 4784 of *LNCS*, pages 1–16. Springer, 2007.

[LM04] C. Lynch and C. Meadows. Sound approximations to Diffie-Hellman using rewrite rules. In *Information and Communications Security*, volume 3269 of *LNCS*, pages 65–70. Springer, 2004.

[LM06] K. Lauter and A. Mityagin. Security analysis of KEA authenticated key exchange protocol. In *PKC 2006*, volume 3958 of *LNCS*, pages 378–394. Springer, 2006.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[Low97] G. Lowe. A hierarchy of authentication specifications. *Computer Security Foundations Workshop, IEEE*, 0:31, 1997.

[LYH05] Y. Li, W. Yang, and C.-W. Huang. On preventing type flaw attacks on security protocols with a simplified tagging scheme. *J. Inf. Sci. Eng.*, 21(1):59–84, 2005.

[MCB10] S. Meier, C. Cremers, and D. A. Basin. Strong invariants for the efficient construction of machine-checked protocol security proofs. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*, pages 231–245. IEEE Computer Society, 2010.

[MCB13] S. Meier, C. Cremers, and D. A. Basin. Efficient construction of machine-checked symbolic protocol security proofs. *Journal of Computer Security*, 2013. to appear.

[Mei12] S. Meier. Source code of the `scyther-proof` tool including the Isabelle/HOL security protocol verification theory and the models of the repaired ISO/IEC 9798 protocols, May 2012. http://hackage.haskell.org/package/scyther-proof-0.5.0.0.

[MMOB10] S. Matsuo, K. Miyazaki, A. Otsuka, and D. A. Basin. How to evaluate the security of real-life cryptographic protocols? - the cases of ISO/IEC 29128 and CRYPTREC. In *Financial Cryptography and Data Security, FC 2010 Workshops, RLCPS, WECSR, and WLC 2010, Revised Selected Papers*, volume 6054 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2010.

[Möd10a] S. Mödersheim. Abstraction by set-membership: verifying security protocols and web services with databases. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 351–360. ACM, 2010.

[Möd10b] S. Mödersheim. Version 2010b of the AIF framework, 2010. Retrieved in August 2012 from http://www2.imm.dtu.dk/~samo/AIF-framework-2010b.zip.

[Möd11] S. Mödersheim. Diffie-Hellman without difficulty. In *Proceedings of the 8th International Workshop on Formal Aspects of Security & Trust (FAST)*, 2011.

[MS01] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In M. K. Reiter and P. Samarati, editors, *ACM Conference on Computer and Communications Security*, pages 166–175. ACM, 2001.

[MS12] S. Meier and B. Schmidt. The Tamarin prover: source code and case studies, September 2012. http://hackage.haskell.org/package/tamarin-prover-0.8.2.0.

[MT12] S. F. Mjølsnes and J.-K. Tsay. Computational security analysis of the UMTS and LTE authentication and key agreement protocols. *CoRR*, abs/1203.3866, 2012.

[MV09] S. Mödersheim and L. Viganò. The Open-source Fixed-point Model Checker for symbolic analysis of security protocols. In *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, pages 166–194, 2009.

[Nat09] National Institute of Standards and Technology. *SP 800-56B, Special Publication 800-56B, Recommendation for Pair-Wise Key Establishment Schemes using Integer Factorization Cryptography*, August 2009.

[NBN11] L. Ngo, C. Boyd, and J. Nieto. Automated proofs for Diffie-Hellman-based key exchanges. In *Proceedings of the 24rd IEEE Computer Security Foundations Symposium*, pages 51 –65, 2011.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[NS78] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, December 1978.

[Pau97] L. C. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 84–95, 1997.

[Pau98] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[Pau99] L. C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Trans. Inf. Syst. Secur.*, 2(3):332–351, 1999.

[Pau01] L. C. Paulson. Relations between secrets: Two formal analyses of the Yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.

[PCTS00] A. Perrig, R. Canetti, J. D. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, 2000.

[Pra65] D. Prawitz. *Natural Deduction*. Almqvist & Wiksell, Stockholm, 1965.

[RG09] J. D. Ramsdell and J. D. Guttman. CPSA: A cryptographic protocol shapes analyzer. In *Hackage*. The MITRE Corporation, 2009. http://hackage.haskell.org/package/cpsa; see esp. `doc` subdirectory.

[SB10] C. Sprenger and D. A. Basin. Developing security protocols by refinement. In *Proceedings of 17th ACM Conference on Computer and Communications Security, CCS 2010*, pages 361–374, 2010.

[SB12] C. Sprenger and D. A. Basin. Refining key establishment. In *Proceedings of the 25th IEEE Computer Security Foundations Symposium*, pages 230–246, 2012.

[SBP01] 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.

[Sch97] S. Schneider. Verifying authentication protocols with csp. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 3–17, 1997.

[Sch11] M. Schaub. Efficient interactive construction of machine-checked protocol security proofs in the context of dynamically compromising adversaries. Master's thesis, ETH Zurich, 2011.

[Sch12] B. Schmidt. *Formal Analysis of Key Exchange Protocols and Physical Protocols*. PhD thesis, ETH Zurich, 2012.

[SMCB12a] B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. *Computer Security Foundations Symposium, IEEE*, 0:78–94, 2012.

[SMCB12b] B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties (extended version), 2012. available http://www.infsec.ethz.ch/research/software/tamarin.

[SPO05] SPORE — Security Protocols Open Repository, 2005. http://www.lsv.ens-cachan.fr/spore.

[Sta11] C. Staub. A user interface for interactive security protocol design. Bachelor's thesis, ETH Zurich, 2011.

[TEB05] F. L. Tiplea, C. Enea, and C. V. Birjoveanu. Decidability and complexity results for security protocols. In E. M. Clarke, M. Minea, and F. L. Tiplea, editors, *VISSAS*, volume 1 of *NATO Security through Science Series D: Information and Communication Security*, pages 185–211. IOS Press, 2005.

[THG99] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(1), 1999.

[Vir02] P. Viry. Equational rules for rewriting logic. *Theor. Comput. Sci.*, 285(2):487–517, August 2002.

[WCW12] R. Wang, S. Chen, and X. Wang. Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services. In *IEEE Symposium on Security and Privacy*, pages 365–379. IEEE Computer Society, 2012.

[WPN08] M. Wenzel, L. C. Paulson, and T. Nipkow. The Isabelle framework. In O. A. Mohamed, C. Muñoz, and S. Tahar, editors, *TPHOLs*, volume 5170 of *Lecture Notes in Computer Science*, pages 33–38. Springer, 2008.

[Yub12] Yubico Inc. Homepage, 2012. Retrieved in September 2012 from https://www.yubico.com/welcome.

[ZN01] R. Zuccherato and M. Nystrom. RFC 3163: ISO/IEC 9798-3 Authentication SASL Mechanism, 2001. http://www.rfc-editor.org/info/rfc3163.

Index

This index contains notions and definitions relevant to Part II. Its objective is to faciliate understanding and extending the work presented in Part II.