DISS. ETH NO. 20856

FORMAL ANALYSIS OF KEY EXCHANGE PROTOCOLS AND PHYSICAL PROTOCOLS

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

presented by
BENEDIKT SCHMIDT
Dipl.-Inf., Universität Karlsruhe (TH), Germany
born May 1st, 1980
Citizen of Germany

accepted on the recommendation of
Prof. Dr. David Basin, examiner
Prof. Dr. Srdjan Capkun, co-examiner
Prof. Dr. Ralf Küsters, co-examiner

2012

Abstract

A security protocol is a distributed program that might be executed on a network controlled by an adversary. Even in such a setting, the protocol should satisfy the desired security property. Since it is hard to consider all possible executions when designing a protocol, formal methods are often used to ensure the correctness of a protocol with respect to a model of the protocol and the adversary. Many such formal models use a symbolic abstraction of cryptographic operators by terms in a term algebra. The properties of these operators can then be modeled by equations. In this setting, we make the following contributions:

1. We present a general approach for the automated symbolic analysis of security protocols that use Diffie-Hellman exponentiation and bilinear pairings to achieve advanced security properties. We model protocols as multiset rewriting systems and security properties as first-order formulas. We analyze them using a novel constraint-solving algorithm that supports both falsification and verification, even in the presence of an unbounded number of protocol sessions. The algorithm exploits the finite variant property and builds on ideas from strand spaces and proof normal forms. We demonstrate the scope and the effectiveness of our algorithm on non-trivial case studies. For example, the algorithm successfully verifies the NAXOS protocol with respect to a symbolic version of the eCK security model.
2. We examine the general question of when two agents can create a shared secret. Namely, given an equational theory describing the cryptographic operators available, is there a protocol that allows the agents to establish a shared secret? We examine this question in several settings. First, we provide necessary and sufficient conditions for secret establishment using subterm-convergent theories. This yields a decision procedure for this problem. As a consequence, we obtain impossibility results for symmetric encryption. Second, we use algebraic methods to prove impossibility results for monoidal theories including XOR and abelian groups. Third, we develop a general combination result that enables modular impossibility proofs. For example, the results for symmetric encryption and XOR can be combined to obtain impossibility for the joint theory.
3. We develop a framework for the interactive analysis of protocols that establish and rely on properties of the physical world. Our model extends standard, inductive, trace-based, symbolic approaches with location, time, and communication. In particular, communication is subject to physical constraints, for example, message transmission takes time determined by the communication medium used and the distance between nodes. All agents, including intruders, are subject to these constraints and this results in a distributed intruder with restricted, but more realistic, communication capabilities than those of the standard Dolev-Yao intruder. Building on our message theory that includes XOR, we also account for the possibility of overshadowing of message parts. We have formalized our model in Isabelle/HOL and have used it to verify protocols for authenticated ranging, secure time synchronization, and distance bounding. The analysis of distance bounding attacks accounts for overshadowing and distance hijacking attacks.

Zusammenfassung

Sicherheitsprotokolle sind verteilte Algorithmen die in einem Netzwerk ausgeführt werden können, das von einem Angreifer kontrolliert wird. Dabei sollen die gewünschten Sicherheitseigenschaften des Protokolls in allen solchen Szenarien gewährleistet sein. Da es schwierig ist während des Protokoll-Entwurfs alle möglichen Ausführungen zu berücksichtigen werden oft formale Methoden eingesetzt, um die Korrektheit eines Protokolls sicherzustellen. Dabei wird ein formales Modell des Protokols und aller möglichen Angreifer genutzt, in dem die kryptographischen Operationen mit Hilfe von Term-Algebren symbolisch modelliert werden. In diesem Zusammenhang präsentieren wir drei Beiträge.

1. Wir präsentieren eine allgemeine Methode für die automatische symbolische Analyse von Sicherheitsprotokollen die Diffie-Hellman Exponentiation und bilineare Pairings benutzen um Sicherheitseigenschaften zu erreichen. Wir modellieren Protokolle als Multiset Rewriting Systeme und Sicherheitseigenschaften als First-Order Formeln. Unser Algorithmus zur automatischen Analyse solcher Protokolle basiert auf Constraint Solving und nutzt Ideen aus der Beweistheorie und von Strand Spaces. Wir demonstrieren die Anwendbarkeit unseres Algorithmus anhand von nicht-trivialen Fallstudien.
2. Wir betrachten die allgemeine Fragestellung ob zwei Agenten ein gemeinsames Geheimnis erzeugen können. Dabei gehen wir davon aus, dass die verfügbaren Operationen durch eine Term-Algebra und Gleichungen beschrieben werden. Wir beweisen mehrere Unmöglichkeitsresultate. Unter anderem beweisen wir Resultate für symmetrische Verschlüsselung, für XOR und ein Kombinationsresultat, mit dem Unmöglichkeitsresultate modular bewiesen werden können.
3. Wir entwickeln ein System zur interaktiven Analyse von Protokollen, die physikalische Eigenschaften der Umgebung in der sie ausgeführt werden nutzen und sicherstellen. Unser zugrundeliegendes Modell erweitert Standard-Modelle mit den Konzepten von Ort, Zeit, und Netzwerk-Kommunikation. Dabei ist die Netzwerk-Kommunikation eingeschänkt, so dass physikalische Gesetze nicht verletzt werden. Wir haben unser Modell in dem interaktiven Beweis-Assistenten Isabelle/HOL formalisiert und die Formalisierung benutzt um die Sicherheit von Authenticated Ranging, Secure Time-Synchronization, und Distance Bounding Protokollen zu analysieren.

Acknowledgements

First, I would like to thank my supervisor David Basin for providing me with the opportunity to perform research in such a great environment. I am also grateful for all his support and advice in developing ideas and presenting them. I also want to thank my coauthors Patrick Schaller, Srdjan Capkun, Simon Meier, Cas Cremers, and Kasper Bonne Rasmussen for the wonderful cooperations. It was a pleasure to interact with you, at work and outside of work. Next, I want to thank Ralf Kuesters for his willingness be a co-examiner and valuable comments on the thesis.

I would also like to thank all members of the Information Security Institute at ETH for a productive and fun workplace. There were many people that provided valuable discussions, insights, and ideas. In particular, I would like to thank Thanh Binh Nguyen, Michèle Feltz, Mario Frank, Simone Frau, Matus Harvan, Felix Klaedtke, Robert Künnemann, Ognjen Maric, Srdan Marinovic, Bruno Conchinha Montalto, Sasa Radomirovic, Ralf Sasse, Matthias Schmalz, Christoph Sprenger, Nils Ole Tippenhauer, Mohammad Torabi Dashti, and Eugen Zalinescu for feedback on the content of this thesis.

For helping me to find the right balance between work and play, I would like to thank my friends in Zürich and Trier. Most importantly, I would like to thank my parents and my two brothers for their patience and for supporting me all the way from the beginning to the end of this journey.

Table of contents

Abstract . . . 3

Zusammenfassung . . . 5

Acknowledgements . . . 7

1 Introduction . . . 13

1.1 Problems . . . 14
- 1.1.1 Key Exchange Protocols and Compromising Adversaries . . . 15
- 1.1.2 Impossibility of Key Establishment . . . 17
- 1.1.3 Physical Protocols . . . 18

1.2 Contributions . . . 19
- 1.2.1 Automated Protocol Analysis . . . 19
- 1.2.2 Impossibility Results for Key Establishment . . . 20
- 1.2.3 Interactive Analysis of Physical Protocols . . . 21

1.3 Outline . . . 23

2 Background . . . 25

2.1 Security Protocols . . . 25
- 2.1.1 Classical Security Protocols and the Dolev-Yao Adversary . . . 25
- 2.1.2 Key Exchange Protocols and Compromising Adversaries . . . 27
- 2.1.3 Tripartite and Identity-Based Key Exchange Protocols . . . 31
- 2.1.4 Physical Protocols . . . 33

2.2 Notational Preliminaries . . . 35

2.3 Term Rewriting . . . 36
- 2.3.1 Standard Notions . . . 36
- 2.3.2 Rewriting Modulo and Finite Variants . . . 37
- 2.3.3 Ordered Completion . . . 39

2.4 Isabelle/HOL and the Inductive Approach . . . 39

3 Automated Protocol Analysis . . . 41

3.1 Security Protocol Model . . . 41
- 3.1.1 Cryptographic Messages . . . 41
- 3.1.2 Labeled Multiset Rewriting with Fresh Names . . . 43
- 3.1.3 Protocol and Adversary Specification . . . 44
- 3.1.4 Trace Formulas . . . 46
- 3.1.5 Formalization of the one-pass UM Protocol . . . 48

3.2 Verification Theory . . . 50

3.2.1 Dependency Graphs 50
3.2.2 Dependency Graphs Modulo AC 52
3.2.3 Normal Message Deduction Rules 53
3.2.4 Normal Dependency Graphs 57
3.2.5 Guarded Trace Formulas 60
3.3 Constraint Solving 62
3.3.1 Syntax and Semantics of Constraints 62
3.3.2 Constraint Solving Algorithm 63
3.3.3 Constraint Solving Rules 64
3.3.4 Optimizations for Constraint Solving 74
3.3.5 One-pass UM: Security proof 80
3.4 Extension with Bilinear Pairings and AC operators 84
3.4.1 Security Protocol Model 84
3.4.2 Formalization of the Joux Protocol 85
3.4.3 Verification Theory 85
3.4.4 Constraint Solving 89
3.5 Implementation and Case Studies 92
3.5.1 The TAMARIN prover 92
3.5.2 Security of NAXOS in the eCK Model 94
3.5.3 wPFS Security of RYY 98
3.5.4 Experimental Results 99
3.5.5 Limitations and Extensions 102

4 Impossibility Results for Key Establishment 105

4.1 Preliminaries 105
4.2 Traces and Deducibility 105
4.2.1 Messages, Frames, and Deducibility 106
4.2.2 Derivation Traces 107
4.2.3 Shared Secrets and Deducibility 107
4.2.4 Relating Protocols and Derivation Traces 110
4.3 Symmetric Encryption and Subterm-Convergent Theories 110
4.3.1 Symmetric Encryption 110
4.3.2 Subterm-Convergent Theories 112
4.4 Monoidal Theories 115
4.4.1 Impossibility for Group Theories 118
4.4.2 Protocol for ACU 119
4.4.3 Impossibility for $ACUI$ and $ACUIh$ 119
4.5 Combination Results for Impossibility 120
4.5.1 Factors, Interface Subterms, Normalization 121
4.5.2 Combination Result 122
4.6 Summary 123

5 Analysis of Physical Protocols 125

5.1 Formal Model 125
5.1.1 Modeled Concepts 125
5.1.2 Agents and Transmitters 126

5.1.3 Physical and Communication Distance 127
5.1.4 Messages 128
5.1.5 Events and Traces 129
5.1.6 Network, Intruder, and Protocols 130
5.1.7 Protocol-Independent Results 132
5.2 Case Studies 134
5.2.1 Authenticated Ranging 134
5.2.2 Ultrasound Distance Bounding 136
5.2.3 Secure Time Synchronization 139
5.3 Extended Model with *Xor* and Overshadowing 143
5.3.1 Message Theory for *Xor* 143
5.3.2 Network Rule with Overshadowing 146
5.4 Case Study: The Brands-Chaum Protocol 147
5.4.1 Security Analysis 150
5.4.2 Distance Hijacking Attack on Wrongly Fixed Version 151
5.4.3 Security of Version with Explicit Binding 153
5.4.4 Security of Version with Implicit Binding 154
5.5 Isabelle Formalization 155

6 Related Work 159

6.1 Security Protocol Analysis 159
6.1.1 Cryptographic Messages 159
6.1.2 Protocol Specification and Execution 161
6.1.3 Property Specification Languages 161
6.1.4 ProVerif 162
6.1.5 Maude-NPA 163
6.1.6 Athena, CPSA, Scyther 163
6.2 Impossibility Results 163
6.2.1 Impossibility Results and Bounds for Protocols 164
6.2.2 Impossibility Results in Cryptography 164
6.2.3 Formal Models of Security Relations 164
6.3 Protocol Analysis with Time, Network Topology, and Location 164

7 Conclusion 167

Appendix A Proofs for Chapter 3 171

A.1 Proofs for Verification Theory 171
A.1.1 Proofs for Multiplication 171
A.1.2 Proofs for Normal Dependency Graphs 179
A.1.3 Proofs for Formulas 184
A.2 Proofs for Constraint Solving 187

Bibliography 191

Chapter 1
Introduction

In the last decade, our reliance on security protocols has increased significantly. The main reasons are the increased usage of the Internet and the ubiquity of wireless networks and devices. On the Internet, security protocols are often used to ensure privacy of personal data or to secure transactions, e.g., in electronic commerce or electronic banking. While wireless devices are also used for similar purposes, new application areas for wireless devices have emerged which require new types of security protocols. For example, car manufacturers have started to replace physical keys with contactless car entry systems. Here, the car door unlocks automatically if the electronic key for the car is sufficiently close.

While these electronic versions of services or processes are often much more convenient than their traditional counterparts, obtaining comparable security is often a challenge. The main problem is that the employed security protocols must achieve their goals even in the presence of adversaries that interfere with their execution. For example, the security of protocols that use the Internet to communicate should not rely on the trustworthiness of the service provider who controls the network infrastructure. Hence, most security protocols employ cryptographic primitives such as hash functions, digital signatures, and encryption to remove the need to trust the communication medium. The remaining challenge for the security protocol designer consists of choosing the right primitives and using the primitives correctly in the protocol. This is a hard problem since the designer has to balance efficiency, simplicity, and different security goals. Therefore, a large number of security protocols is proposed each year and many of them contain flaws. This includes newly published protocols [57], standardized protocols [42, 23], and protocols with large-scale deployment [82, 12, 167]. In practice, protocol design is therefore often an iterative process where new attacks are discovered and the protocol is then modified to resist these attacks.

As shown by Lowe [115] who discovered an attack on the Needham-Schroeder protocol [134] that was unknown for 10 years, it is often unclear if *all* possible attacks have been considered. It is therefore desirable to perform the following steps:

1. Make all assumptions about the capabilities of the adversary explicit and give a complete specification of the protocol and the desired security property.
2. Prove that the specified protocol achieves the specified security property as long as the assumptions on the adversary hold.

This allows the protocol designer to obtain a guarantee that the protocol is correct *with respect to the given specification and assumptions*. Following this idea, two branches of research, adopted by the cryptographic community and the formal methods community, have emerged.

The computational (or cryptographic) model represents messages by bitstrings and formalizes the protocol as a probabilistic polynomial-time Turing machine. The security of the protocol is then defined in terms of a game, where an arbitrary probabilistic polynomial-time Turing machine, which models the adversary, interacts with the protocol. The protocol is secure if the probability of winning this game is negligible for all such adversaries. Usually, the proof is performed by reduction to a problem that is assumed to be hard such as factoring large composite integers. If there is an adversary that attacks the protocol with non-negligible probability, then the adversary can be used to construct an efficient solver for the hard problem, which was assumed to be infeasible. These proofs are usually long and intricate and many published proofs are incomplete or do not make all assumptions explicit [113, 103, 127, 98]. Recently, machine-support for performing proofs in the computational model has been developed [19, 29], but the fully automated construction of such proofs for security protocols is still out of reach.

The symbolic (or formal) model introduced by Dolev and Yao [69] models messages by terms in a term algebra and the possible operations on messages by message deduction rules. For example, $enc(m, k)$ represents the encryption of the message represented by the term m with the key represented by the term k. This term can then be used as follows to deduce new messages. It can be paired with other known messages, encrypted with known keys, or, if the key k is known, m can be extracted. The assumption that an encryption cannot be used in any other way is called the perfect cryptography assumption. In symbolic models, different formalisms are used to specify protocols and to define their possible executions. Usually, these formalisms use terms to denote the messages that a protocol receives and sends on the network. Given a protocol, its executions are defined as the possible interactions with an adversary that controls the network and deduces new messages by applying message deduction rules. A protocol satisfies a security property, such as secrecy of a certain message, if all its executions satisfy the property.

Compared to the computational model, the symbolic model abstracts away from many details. On the one hand, this means that the symbolic model may miss attacks captured by the computational model. On the other hand, the symbolic model is considerably simpler and many practically relevant attacks are still captured by the symbolic model. Because of the relative simplicity of the model, there are several tools [28, 62, 77] that perform fully automated security proofs for a wide range of security protocols. Some of them also provide counterexamples when the proof fails. The symbolic model is also a good fit for obtaining meta-theoretical results about security protocols. Finally, it is also considerably simpler to mechanize symbolic security proofs in a theorem prover to obtain machine-checked proofs since no reasoning about bitstrings, computational complexity, and probabilities is required.

1.1 Problems

Even though symbolic models models have been successfully applied to many different application areas, there are still several types of security protocols that are outside the scope of existing approaches. In this thesis, we focus on two such types of protocols. First, there is no method that supports the fully automated, unbounded analysis of many recent authenticated key exchange protocols with respect to their intended adversary models.

Second, there is no general formal model (computational or symbolic) that supports reasoning about protocols that utilize and establish physical properties of nodes and their environment. As an example of such a protocol, consider the protocol executed between an electronic key and a car to establish the distance between the two. Moreover, we investigate a third question in the symbolic setting: Which operations are required to allow two agents to establish a key using an authentic (but not secret) channel?

1.1.1 Key Exchange Protocols and Compromising Adversaries

Authenticated key exchange (AKE) protocols are widely used components in modern network infrastructures. They are often based on public-key infrastructures and their goal is to authenticate the communication partners and to establish a shared symmetric session key that can be used to secure further communication. Recent protocols are designed to achieve these goals even in the presence of strong adversaries who can, under certain conditions, reveal session keys, long-term private keys, and the randomness used by the participants. For example, the NAXOS protocol [111] is designed to be secure in the extended Canetti-Krawzyk (eCK) model [111] where only those combinations of reveals are forbidden that directly lead to an attack on this protocol.

The main building block for most AKE protocols is Diffie-Hellman (DH) exponentiation in a cyclic group $\mathbb{G}$ of prime order p generated by a group element g, i.e., $\mathbb{G} = \{\mathsf{g}^n | n \in \mathbb{N}\}$ and $\mathsf{g}^p = 1$. AKE protocols exploit equalities such as $(\mathsf{g}^n)^m = (\mathsf{g}^m)^n$ that hold in all such groups and that allow the participants to compute the same key in different ways. For instance, in the Diffie-Hellman protocol [68], the participant A knows his private exponent a and the public group element g^b of the other participant B. Similarly, B knows b and g^a. Then both can compute g^{ab} as $(\mathsf{g}^b)^a$ and $(\mathsf{g}^a)^b$. Attacks on AKE protocols also exploit equalities that hold in such DH groups. Interestingly, many attacks are generic in the following sense. They only use operations and equalities that hold in *all* DH groups and therefore work independently of the concrete group used by the protocol.

AKE protocols and their adversary models are an active area of research [42, 59, 43, 58, 166]. Many protocols are proposed together with a pen and paper proof in the computational model with respect to an adversary model adapted for the protocol under consideration. The proofs are either just proof sketches or long and complicated and it is hard to check if such a proof is correct or can be adapted to a modified adversary model. Hence, it would be useful to have a tool that takes a specification of a protocol and an adversary model and performs a fully automated search for a proof or a counterexample in the symbolic model. Of course, the existence of a symbolic proof does not rule out all attacks captured by the computational model. But with the right symbolic model, a symbolic proof is still meaningful since it rules out a well-defined and relevant class of attacks. Furthermore, a symbolic attack is also a computational attack and might help to discover flawed computational proofs or help to fix the protocol or adversary model before attempting a computational proof.

To perform automated symbolic analysis of such AKE protocols, we identify the following requirements on the employed method:

1. The message theory must capture the required operations in DH groups and their algebraic properties. This means at least exponentiation and the previously mentioned equalities required to establish the shared key in the DH protocol must be captured.

Additionally, it would be nice to handle inverses of exponents, which are used by some protocols such as MTI/C0 [119]. Support for the addition of exponents and multiplication of DH group elements would further extend the scope to more protocols [113, 103]. Note that if more operations and equations are modeled, then more attacks are captured. This might be relevant even if the protocol under consideration does not employ these operations directly.

2. To support the analysis of modern tripartite and identity-based AKE protocols, the message theory should also support bilinear pairings.
3. Since the adversary models are defined by stateful queries such as

   "*SessionKeyReveal*(s): If the session is completed, but unexpired, then $\mathcal{M}$ obtains the corresponding session key" [42],

   it would be desirable if the method directly supports the modeling of state and reasoning about state such as the status of a session.
4. Since the winning conditions for the adversary that capture the adversary model often contain logical formulas with temporal statements such as

   "[there must be a] Long-Term Key Reveal(B) before completion of the session" [110],

   it would be desirable for the specification language to support such (temporal) formulas.
5. The method should support attack finding and proofs with respect to an unbounded number of protocol sessions.

Requirements 1.–4. apply to the specification languages for protocol and property and to the execution model. The usual way to support 1. and 2. is to represent cryptographic messages by terms modulo an equational theory $\mathcal{E}$ that captures the required algebraic properties. Requirement 5. applies to the method that is used to analyze protocols with respect to an execution model that satisfies the other requirements. Since secrecy for an unbounded number of sessions is already undecidable for simpler execution models [72], the best we can hope for is to find proofs or attacks for many relevant examples. Given requirement 5., we evaluate existing approaches for the automated unbounded analysis of security protocols with respect to these requirements. The approaches can be roughly assigned to three different groups which we consider separately.

The Horn-theory based approach [168] implemented in ProVerif [28] was originally restricted to secrecy and similar properties that can be encoded as derivability in Horn theories. Blanchet [30] extends the approach to support correspondence properties, which can be used to formalize authentication properties such as injective agreement. Blanchet et al. [31] also show how to support a restricted set of equational theories that does not include associative and commutative operators. For DH, this approach supports a model that captures exactly the equality $(\mathsf{g}^x)^y = (\mathsf{g}^y)^x$, but does not account for inverses of exponents. Together, these extensions have been used in [1] to analyze AKE protocols with respect to adversary models that account for session key compromise and long-term key compromise. This work combines correspondence properties with ProVerif's support for phases and distinguishing between compromised and uncompromised sessions. It is unclear how well this approach generalizes to more advanced adversary models such as eCK (with perfect forward secrecy) because of the limited support for temporal statements in correspondence properties and the limited support for non-monotonic state in ProVerif. Küsters and Truderung [108, 109] later lift the previously described restriction on equational theories and show how ProVerif can be used to verify protocols with respect to XOR and a model of Diffie-Hellman exponentiation with inverses. Pankova and Laud [138] have recently extended this approach with support for bilinear pairings. Unfortunately, these extensions are not compatible with the support for correspondence properties and are therefore restricted to authentication properties that can be encoded as derivability in Horn theories.

Maude-NPA [77] is based on backwards narrowing modulo an equational theory. Protocols can be specified by linear role scripts and properties can be specified by characterizing the attack states by symbolic terms. The backwards narrowing approach is very general with respect to the supported equational theories and only requires an implementation of equational unification to work. To achieve termination in bounded and unbounded scenarios, the tool employs various state-space reduction techniques [78] that have to be adapted for different equational theories. In [76], the Diffie-Hellman protocol is analyzed with respect to the standard Dolev-Yao adversary and an equational theory that does not account for inverses. We are not aware of any case studies that consider an equational theory with inverses or advanced adversary models.

The Athena [162], Scyther [62], and CPSA [150] tools are all based on similar ideas. They perform a complete search for attacks by representing attack states symbolically by the set of events that must have occurred and the causal dependencies between these events. The causal dependencies capture, for example, that a message must be known by the adversary before it is received by the protocol. The search can terminate for two different reasons. First, if all causal dependencies are satisfied, a ground attack trace (or bundle) can be extracted from the symbolic state. Second, if some dependency cannot be satisfied, then the attack state is not reachable and the security of the protocol has been proved. All three tools only support linear role scripts and do not support equational theories. The tools all support security properties of the form $\forall \vec{x}.\varphi \Longrightarrow \exists \vec{y}.\psi$ where φ is a conjunction of events and ψ is a disjunction of events. Scyther has been extended [22] with support for numerous compromising adversary models, but these are hardcoded and cannot be specified by the user. We refer to Chapter 6 for a detailed discussion of existing approaches.

1.1.2 Impossibility of Key Establishment

Consider a pair (or more generally a group) of honest agents who have no shared secret, but who can communicate over a public channel in the presence of a passive adversary. Furthermore, assume that each agent can generate unguessable nonces, has access to public information, and may use different cryptographic operators. Is it possible for these agents to establish a shared secret?

There are of course many ways to answers this question positively. For example, if the cryptographic operators include a public-key cryptosystem, an agent may simply send his public key over the public channel. Any other agent could then encrypt a secret with the public key that can be decrypted only by the agent holding the corresponding private key. Similarly, if a multiplicative group is given for which the so called Diffie-Hellman problem is hard, agents can use the Diffie-Hellman protocol to establish a shared secret. There are also negative answers if the set of cryptographic operators is sufficiently restricted. In particular, there is a folk theorem that no protocol exists if only symmetric encryption can be used. However, to the best of our knowledge, no formal proof of this folk theorem has previously been given.

Establishing impossibility results and developing related proof methods are of fundamental theoretical importance as they explain what cannot be achieved using cryptographic operators, specified equationally. Practically, impossibility results delineate the solution space in protocol design and enable a more systematic approach to protocol development by guiding the choice of cryptographic operators. This is especially relevant in resource constrained scenarios, like with smartcards or sensor networks, where operations like public-key cryptography are sometimes considered too expensive and should be avoided, where possible.

1.1.3 Physical Protocols

The shrinking size of microprocessors combined with the ubiquity of wireless networks and devices has led to new application areas for networked systems with novel security requirements for the employed protocols. Whereas traditional security protocols are mainly concerned with message secrecy or variants of authentication, new application areas often call for new protocols that securely establish properties of the network environment. Examples include:

Physical Proximity. One node must prove to another node that a given value is a reliable upper bound on the physical distance between them. Such protocols may use authentication patterns along with assumptions about the underlying communication medium [34, 38, 92, 123, 151]. An important use-case for such protocols are contactless car entry systems.

Secure Localization. A node must determine its true location in an adversarial setting or make verifiable statements about its location by executing protocols with other nodes [40, 105, 114, 156]. Secure localization and physical proximity verification protocols, and attacks on them, have been implemented on RFID, smart cards, and Ultra-Wide Band platforms [71, 153, 151].

Secure Time Synchronization. A node must securely synchronize its clock to the clock of another trusted node in an adversarial setting [84, 164].

Secure Neighbor Discovery or Verification. A node must determine or verify its direct communication partners within a communication network [139]. Reliable information about the topology of a network is essential for all routing protocols.

What these examples have in common is that they all concern physical properties of the communication medium or the environment in which the nodes live. Furthermore, standard symbolic protocol models based on the Dolev-Yao intruder lack the required features to formalize these protocols:

1. They do not model global time and local clocks that can be accessed by nodes and deviate from the global time.
2. They do not reflect the location of nodes and the distance between nodes induced by their locations.
3. They do not reflect the network topology which characterizes possible communication between nodes and lower bounds on message transmission times.
4. They do not support a distributed intruder. If location and communication abilities of individual nodes are reflected in a model, then this implies certain bounds on the exchange of information between nodes. Since these bounds should also apply to the intruder, e.g., the intruder cannot transfer knowledge instantaneously from one location to another, we cannot collapse all intruders into one intruder.

There are some models that account for one of these aspects, but none that capture all of them. Again, we refer to Chapter 6 for a detailed discussion of existing approaches.

1.2 Contributions

To address the three problems described in the previous section this thesis presents the following contributions.

1.2.1 Automated Protocol Analysis

Starting with an expressive and general security protocol model where protocol and adversary are specified as multiset rewriting rules, security properties are specified as first-order trace-formulas, and executions are defined by multiset rewriting modulo an equational theory, we present four contributions. First, we demonstrate that such a model is well-suited for the formalization of AKE protocols and their adversary models. Second, we give a verification theory based on an alternative representation of executions that reduces the search space and allows for a compact symbolic representation of executions by constraints. Third, we give a sound and complete constraint solving algorithm for the falsification and unbounded verification of protocols with respect to our model. Fourth, we implemented our algorithm in a tool, the TAMARIN prover [126], and validated its effectiveness on a number of case studies.

Our security protocol model uses an equational theory $\mathcal{E}$ to specify the considered cryptographic operators, their properties, and the message deduction capabilities of the adversary. We support the disjoint combination of a subterm-convergent theory and a theory that models DH exponentiation with inverses, and bilinear pairings. To specify the protocol and adversary capabilities, we use a labeled multiset rewriting system $\mathcal{S}$ with support for fresh name generation and persistent facts. The traces are then defined by labeled multiset rewriting modulo $\mathcal{E}$ with $\mathcal{S}$. In our setting, a trace is a sequence of sets of facts. Traces that satisfy a given security property are then characterized by first-order formulas built over the predicate symbols $f@i$ (fact f occurs in the trace at position i), $i \lessdot j$ (timepoint i is smaller than timepoint j), and $t \approx s$ ($t =_{\mathcal{E}} s$ for terms t and s). We support arbitrary nesting of quantifiers and quantification over messages and timepoints, but all quantified variables must be guarded by atoms [7].

We show that our model allows for a natural formalization of a wide range of protocols and adversary models. For example, we present models for NAXOS [111] security in the eCK model [111], perfect forward secrecy of the Joux tripartite key exchange protocol [97], and security of the RYY protocol [155] under session key reveals and long-term key reveals.

The main result of our verification theory is that instead of defining executions via multiset rewriting modulo $\mathcal{E}$ with $\mathcal{S}$, we can use an alternative definition based on $\mathcal{E}$ and $\mathcal{S}$ that reduces the search space and simplifies the symbolic representation of our search-state by constraints. To arrive at this alternative definition, we perform a series of reduction steps which preserves the set of traces. First, we switch from multiset rewriting derivations to dependency graphs, a representation that uses sequences of multiset rewriting rule instances and causal dependencies between generated and consumed facts, similar to strand spaces [83]. Second, we decompose $\mathcal{E}$ into a rewrite system $\mathcal{R}$ and an equational theory $\mathcal{AX}$ (that contains no cancellation rules) such that $\mathcal{R}$,$\mathcal{AX}$ has the finite variant property. This allows us to replace $\mathcal{S}$ by its set of $\mathcal{R}$,$\mathcal{AX}$-variants and use dependency graphs modulo $\mathcal{AX}$. Third, we partition the message deduction rules into deconstruction and construction rules, which is possible since we now use $\mathcal{R}$,$\mathcal{AX}$-variants, to apply the notion of normal proofs [146, 48] to message deduction in dependency graphs. After extending this notion from proof trees to proof graphs and from the standard Dolev-Yao operators to DH exponentiation and bilinear pairings, we show that we can restrict ourselves to normal dependency graphs in our search since normal message deduction is complete.

Our constraint solving algorithm takes a security property φ and a protocol P and performs a complete search for counterexamples to φ that are also traces of P. Our algorithm exploits the results from our verification theory and searches for normal dependency graphs modulo $\mathcal{AX}$. We give a full proof of soundness and completeness of the constraint solving rules and also show that we can extract an attack if we reach a solved constraint system.

We show that despite the undecidability of the problem, Tamarin performs well: For non-trivial AKE protocols and adversary models from the literature, it terminates in the vast majority of cases. For most 2-round AKE protocols and their intended adversary models, the times for falsification and unbounded verification are in the range of a few seconds. For more complicated models including multi-protocol scenarios, tripartite AKE protocols, and identity-based AKE protocols, Tamarin terminates in under two minutes.

1.2.2 Impossibility Results for Key Establishment

We present a formal framework to prove impossibility results for secret establishment for arbitrary cryptographic operators in the symbolic setting. We model messages and operations by equational theories and communication by traces of events as is standard in symbolic protocol analysis. The initial question of whether it is possible for two agents to establish a shared secret therefore reduces to the question: Is there a valid trace where two agents end up sharing a message that cannot be deduced from the exchanged messages?

We start by applying our framework to the equational theory that models symmetric encryption and prove the folk theorem that secret establishment is impossible in this setting. It turns out that symmetric encryption is actually an instance of the more general case where the properties of the involved operators can be described by a subterm-convergent theory. For this general class of equational theories, we present a necessary and sufficient condition for the possibility of secret establishment based on labelings of the equations. This directly yields a decision procedure that either returns a labeling that corresponds to a trace where two agents establish a shared secret or returns “impossible” if there is none. For an equational theory that models a public-key cryptosystem, the labeling returned corresponds to the message exchange previously mentioned where the secret is encrypted using the public key previously exchanged. Afterwards, we consider monoidal theories. First, we show that secret establishment is impossible for the subset of group theories which includes theories that model XOR and abelian groups. For the remaining monoidal theories that are usually considered in the setting of security protocol analysis, we either exhibit a protocol from the literature that can be used to establish a shared secret or give a separate proof that secret establishment is impossible. This includes theories that model multisets and sets. Monoidal theories are not subterm-convergent since they define associative and commutative operators. Therefore, we use algebraic methods that exploit the isomorphisms between the term algebras modulo the equational theory and the standard algebraic structure of a (semi)ring.

The above results are for theories in isolation. We also investigate the problem of combining theories and prove a combination result for disjoint theories: Secret establishment in the combination of the theories is possible if and only if it is already possible for one of the individual theories alone. This allows for modular proofs where separate results are combined. For example, we prove this way that secret establishment is impossible for symmetric encryption together with XOR.

1.2.3 Interactive Analysis of Physical Protocols

We present a formal model for reasoning about security guarantees of physical protocols like those listed earlier. Our model builds on standard symbolic approaches and accounts for physical properties like time, the location of network nodes, and properties of the communication medium. Honest agents and the intruder are modeled as network nodes. The intruder, in particular, is not modeled as a single entity but rather as a distributed one and therefore corresponds to a set of nodes. The ability of the nodes to communicate and the speed of communication are determined by nodes' locations and by the propagation delays of the communication technologies they use. As a consequence, nodes (both honest and those controlled by the intruder) require time to share their knowledge and information that they exchange cannot travel between nodes at speeds faster than the speed of light. The intruder and honest agents are therefore subject to physical restrictions. This results in a distributed intruder with communication abilities that are restricted, but more realistic than those of the classical Dolev-Yao intruder.

Our model bridges the gap between informal approaches used to analyze physical protocols for wireless networks and the formal approaches taken for security protocol analysis. Informal approaches typically demonstrate the absence of a given set of attacks, rather than proving that the protocol works correctly in the presence of an active adversary taking arbitrary actions. In contrast, existing formal approaches fail to capture the details necessary to model physical protocols and their intended properties. To bridge this gap, our model formalizes an operational, trace-based semantics of security protocols that accounts for time, location, network topology, and distributed intruders. To model cryptographic operators and message derivability, we reuse Paulson's [140] message theory based on the perfect cryptography assumption and extend it with XOR.

In what follows, we explain our contributions in more detail. First, we give a novel operational semantics that captures the essential physical properties of space and time and thereby supports natural formalizations of many physical protocols and their corresponding security properties. For example, properties may be stated in terms of the relative distance between nodes, the locations of nodes, and the times associated with the occurrence of events. Moreover, protocols can compute with, and base decisions upon, these quantities. To obtain a realistic model of the communication technology used, for example, by distance bounding protocols, our operational semantics accounts for the modification of messages that are in transmission. More precisely, we account for the following concrete scenarios. We allow the intruder to overshadow individual components of a concatenation by known messages. Additionally, we account for the non-negligible probability of flipping a low number of bits in an unknown message m by randomly sending data. This allows the adversary to modify the original message m into a close message m', with respect to the Hamming-distance.

Second, despite its expressiveness, our operational semantics is still simple and abstract enough to allow its complete formalization. We have formalized our model in Isabelle/HOL and used it to formally derive both protocol-independent and protocol-specific properties, directly from the semantics. Protocol-independent properties formalize properties of communication and cryptography, independent of any given protocol. For example, it follows from our operational semantics that there are no collisions for randomly chosen nonces and that communication cannot travel faster than the speed of light. This allows us to prove in a protocol-independent way a lower bound on the time until an adversary learns a nonce depending on his distance to the node generating the nonce. We use these properties, in turn, to prove protocols correct or to uncover weaknesses or missing assumptions through unprovable subgoals.

Third, we show that our approach is viable for the mechanized analysis of a range of wireless protocols. We demonstrate this by providing four case studies that highlight different features of the model:

- Our formalization of an authenticated ranging protocol shows how time-of-flight measurements of signals relate to physical distances between nodes. Additionally, the model has to account for local computation times which are included in protocol messages.
- Our formalization of an ultrasound distance bounding protocol demonstrates how the model accounts for transceivers that employ different communication technologies and their interaction. Furthermore, the example shows how our notion of location can be used to formalize private space assumptions.
- Our formalization of a secure time synchronization protocol illustrates how we can model relations between local clock offsets of different nodes. It also shows how bounds on the message transmission time can be specified.
- Our formalization of the Brands-Chaum distance bounding protocol [34] demonstrates the usage of XOR and why the model accounts for overshadowing. We present an attack on the protocol and various flawed and successful fixes.

Finally, we discuss how our security property for distance bounding protocols captures a new class of attacks that is relevant in practical scenarios, but has previously been overlooked by informal approaches. Consequently, we were the first to discover such an attack on a distance bounding protocol by Meadows et al. [123].

1.3 Outline

Chapter 2. Presents background on security protocols, some mathematical preliminaries, and background on term rewriting.

Chapter 3. Describes the theory and implementation of the Tamarin prover. The chapter is based on joint work and extends it with support for bilinear pairings and associative and commutative operators. The joint work has been published in:

Automated analysis of Diffie-Hellman protocols and advanced security properties.
With Simon Meier, Cas Cremers and David A. Basin.
In *Proceedings of the 25rd IEEE Computer Security Foundations Symposium, CSF 2012*, pages 78–94. IEEE Computer Society, 2012.

Chapter 4. Describes the impossibility results for key establishment. The chapter is based on joint work and generalizes the impossibility results for XOR and abelian groups to (monoidal) group theories. The joint work has been published in:

Impossibility results for secret establishment.
With Patrick Schaller and David Basin.
In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010*, pages 261–273. IEEE Computer Society, 2010.

Chapter 5. Describes the framework for physical protocols and the case studies. This is joint work that has been published in:

Modeling and verifying physical properties of security protocols for wireless networks.
With Patrick Schaller, David Basin, and Srdjan Capkun.
In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009*, pages 109–123. IEEE Computer Society, 2009.

Let's get physical: models and methods for real-world security protocols.
With David Basin, Srdjan Capkun, and Patrick Schaller.
In *Theorem Proving in Higher Order Logics (TPHOLs)*, pages 1–22. 2009 (invited paper).

Formal reasoning about physical properties of security protocols.
With David Basin, Srdjan Capkun, and Patrick Schaller.
In *ACM Transactions on Information and System Security (TISSEC)*, 14(2):16, 2011.

Distance Hijacking Attacks on distance bounding protocols.
With Cas Cremers, Kasper B. Rasmussen, and Srdjan Capkun.
In *33rd IEEE Symposium on Security and Privacy, S&P 2012*, pages 113–127. IEEE Computer Society, May 2012.

Chapter 6. Summarizes the related work and compares it to the contributions of this thesis.

Chapter 7. Presents the conclusion and discusses future work.

Chapter 2
Background

In this chapter, we present background on security protocols and technical background. For security protocols, we focus on authenticated key exchange protocols and physical protocols. The technical background consists of the mathematical preliminaries, a short introduction to term rewriting, and a short introduction to Isabelle/HOL [136] and Paulson's inductive approach to protocol verification [140].

2.1 Security Protocols

In this section, we first introduce the classical Dolev-Yao model for security protocols. Then we present key exchange protocols based on Diffie-Hellman exponentiation and bilinear pairings and their desired security properties. Finally, we introduce physical protocols such as authenticated ranging and distance bounding protocols.

2.1.1 Classical Security Protocols and the Dolev-Yao Adversary

A security protocol is a distributed program that is executed by different entities called agents that exchange messages. The goal of a security protocol is to achieve a certain objective, even in the presence of external or internal attackers that do not follow the rules of the protocol. For example, the goal of a security protocol could be to enable the participants to agree on a secret without revealing it to non-participants. A security protocol is usually split into multiple roles such as initiator and responder. Each role specifies the actions of the agent executing the role. The set of possible actions usually includes sampling random values, applying cryptographic operators, sending messages, and receiving messages.

As an example, consider the simple challenge-response protocol given in Figure 2.1. The protocol consists of two roles $\mathcal{A}$ and $\mathcal{B}$ whose actions are given on the left and right side of the figure. The arrows in the middle depict the exchanged messages. Note that we use the roles $\mathcal{A}$ and $\mathcal{B}$ as placeholders for the agents executing the roles in the specification of the messages. An agent A executing an instance of the role $\mathcal{A}$ that intends to execute the protocol with an agent B executing role $\mathcal{B}$ proceeds as follows. First, he chooses a fresh nonce na and encrypts the nonce with B's public key sending the result to B. Then, he waits for a response by B. If the response is equal to the hash of na, he successfully finishes the protocol. Otherwise, the protocol execution fails. An agent B executing an instance of the role $\mathcal{B}$ receives a message from A and tries to decrypt the message with his secret key $sk_{\mathcal{B}}$. If the decryption succeeds, he applies a hash function to the resulting cleartext and sends this message back to A. We call an instance of a protocol role executed by some agent a *session*. We call the intended communication partner of a session the *peer*.

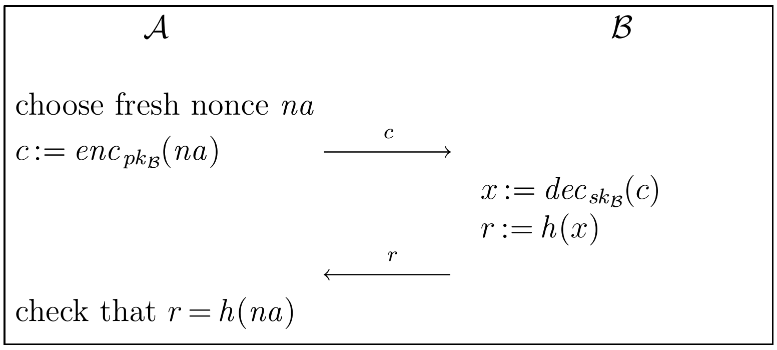


Figure 2.1. A simple challenge-response protocol.

Even though the figure suggests that the message exchange between $\mathcal{A}$ and $\mathcal{B}$ is authentic, protocols are often analyzed in an open network where this is not the case. For example, many symbolic approaches analyze security protocols with respect to the so called Dolev-Yao model [69] which is based on the following three assumptions:

1. Each agent can simultaneously execute an arbitrary number of sessions of the protocol under consideration, possibly executing different roles with different parameters at the same time.
2. The perfect cryptography assumption is employed. This means that cryptographic operators are modeled by a term algebra and only a fixed set of operations can be used to deduce new messages from known messages. For example, given a ciphertext, the only way to recover the plaintext is to perform decryption with the right key. Furthermore, the set of agents is partitioned into honest agents and dishonest agents. The long-term keys of honest agents are assumed to be secret and the long-term keys of dishonest agents are known to the adversary.
3. With respect to the network, the strongest possible adversary is assumed. The Dolev-Yao model identifies the network and the adversary. This means the adversary receives all messages sent by other agents and agents can only receive messages from the adversary. Here, the adversary is allowed to send any message that is deducible from received messages and from his initial knowledge.

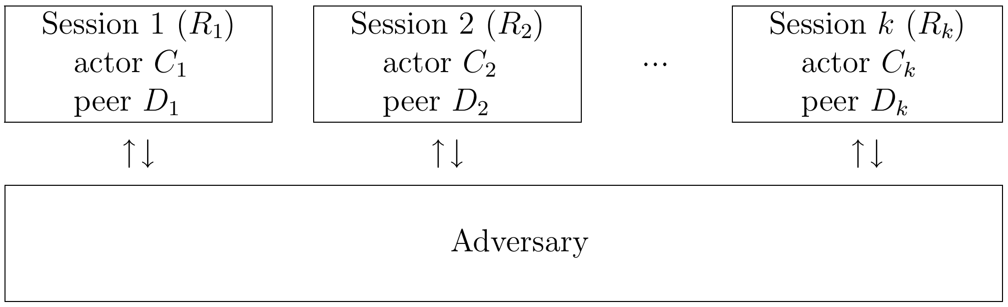


Figure 2.2. Protocol execution in the Dolev-Yao model.

Figure 2.2 depicts a protocol execution in the Dolev-Yao model. Different protocol sessions receive messages from the adversary and send messages to the adversary. There is no guarantee that the sessions match up, i.e., that the communication patterns correspond to the desired ones. The adversary can delete, modify, replay, and forward messages from different sessions. The goal of a security protocol is often to prevent such undesired behaviors. An *authentication property* ensures that if a protocol session finishes successfully under certain conditions, then there is another session that agrees on certain parameters such as the participants and the exchanged messages. Authentication properties are often used together with *secrecy* properties that guarantee that the adversary cannot learn certain protocol data.

Consider the simple challenge-response protocol from Figure 2.1. Here, an honest agent A executing role $\mathcal{A}$ with an honest peer B obtains two guarantees after successfully finishing the protocol. First, B executes a session of the role $\mathcal{B}$ that sends the response after A sent the challenge and before A receives the response. Second, the nonce na is known to B and not known to any other agents. From the perspective of B, there are no useful guarantees since the adversary can always fake the message $enc_{pk_{\mathcal{B}}}(na)$.

2.1.2 Key Exchange Protocols and Compromising Adversaries

In this section, we discuss the mathematical primitives employed by most authenticated key exchange (AKE) protocols, the signed Diffie-Hellman protocol, the Unified Model (UM) protocol, and the extended Canetti-Krawczyk adversary model for AKE protocols.

2.1.2.1 Diffie-Hellman Exponentiation

In the following, we consider a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order p. For $n \in \mathbb{N}$, g^n denotes the n-fold product $\prod_{i=1}^{n} g$ of g. Since $g^p = 1$, we can consider the exponents as elements of $\mathbb{Z}_p$, the integers modulo p. We are interested in such groups, where, given g^n and g^m for randomly chosen n and m, it is hard to compute g^{nm}. This problem is called the *computational Diffie-Hellman* (CDH) *problem*. The hardness of the CDH problem implies the hardness of the *discrete logarithm* (DL) *problem*, which states that it is hard to compute n from g^n for a random n. In the following, we call a group as described above a *Diffie-Hellman (DH) group*. It is easy to see that the equality $(g^n)^m = g^{nm} = (g^m)^n$ holds in all DH groups. This equality states that exponentiation with n and exponentiation with m commute and it is the basis of the Diffie-Hellman Key Exchange protocol.

2.1.2.2 Diffie-Hellman Key Exchange Protocol

The basic DH protocol proposed by Diffie and Hellman [68] in 1976 is the first realistic protocol that allows two agents to establish a shared secret over an authentic (not confidential) channel without sharing any information beforehand. In Figure 2.3, we show a variant of the DH protocol that establishes a shared key between two agents. The *SIGDH* protocol uses signatures to support insecure channels. We assume that before starting the protocol, $\mathcal{A}$ knows his own signing key $sk_{\mathcal{A}}$ and $\mathcal{B}$'s signature verification key $pk_{\mathcal{B}}$ and we have analogous assumptions on $\mathcal{B}$'s knowledge. To initiate the protocol, $\mathcal{A}$ chooses the responder $\mathcal{B}$ and a random exponent $x \in \mathbb{Z}_p$. This exponent is often called the *ephemeral private key* of the given session. The group element g^x which is sent to $\mathcal{B}$ is the corresponding *ephemeral public key*. To proceed, $\mathcal{A}$ signs the concatenation of $\mathcal{B}$'s identity and g^x with his signing key and sends the resulting message to $\mathcal{B}$. When $\mathcal{B}$ receives the message from $\mathcal{A}$, he verifies the signature and stores the received ephemeral public key X. Then, he chooses a random exponent $y \in \mathbb{Z}_p$ and replies with the concatenation of g^y and a signature on the concatenation of $\mathcal{A}$'s identity and g^y. $\mathcal{A}$ verifies the signature and stores the received group element as Y. Now, both can compute the shared key $h(g^{xy})$ as $h(Y^x)$ and $h(X^y)$. The value g^{xy} is often called *ephemeral DH key*.

$\mathcal{A}$		$\mathcal{B}$
signature keys: $(sk_{\mathcal{A}}, pk_{\mathcal{A}})$		signature keys: $(sk_{\mathcal{B}}, pk_{\mathcal{B}})$
choose random exponent x		
$X := g^x$, $\sigma_{\mathcal{A}} := sig_{sk_{\mathcal{A}}}(\langle \mathcal{B}, X \rangle)$	$\xrightarrow{\langle X, \sigma_A \rangle}$	check signature σ_A
		choose random exponent y
		$Y := g^y, \sigma_{\mathcal{B}} := sig_{sk_{\mathcal{B}}}(\langle \mathcal{A}, Y \rangle)$
check signature $\sigma_{\mathcal{B}}$	$\xleftarrow{\langle Y, \sigma_{\mathcal{B}} \rangle}$	
compute key $h(Y^x)$		compute key $h(X^y)$

Figure 2.3. The *SIGDH* protocol.

In the classical Dolev-Yao model, the protocol satisfies the following security property: If both the actor and the peer are honest and successfully execute the protocol, then they agree on the participants and the key. Additionally, the key is secret. We can also include the roles in the signatures to enforce agreement on the roles.

In many settings, it makes sense to consider a stronger adversary. For example, an adversary may learn the long-term secrets of one or both participants after the session is finished. It may also be possible for an adversary to learn the ephemeral secret keys of some sessions, e.g., by performing side-channel attacks during the computation of the ephemeral public key or because the random generator of a participant is broken. It is also important to consider the loss of unrelated session keys. These considerations have led to several additional security requirements. The main objective in the design of AKE protocols is to obtain protocols that are resilient to such threats while achieving good performance and remaining simple to understand and analyze.

At first, the following additional attacks and requirements have been considered in addition to the security property described above.

Unknown Key Share (UKS) Attack. An agent A establishes a key with an honest agent B, but B thinks that he shares this key with a (possibly dishonest) $C \neq A$.

Key Compromise Impersonation (KCI) Resilience. Even if the long-term key of A has leaked, the adversary cannot impersonate an honest agent B to A.

Perfect Forward Secrecy (PFS). If the long-term keys of both participants leak after the session key is established, the session key still remains secret.

The *SIGDH* protocol is resilient against UKS and KCI attacks and also provides PFS. But if *one* ephemeral private key x of A leaks, the adversary can reuse the signature and establish a key with an arbitrary B impersonating A repeatedly. In Section 2.1.2.4, we will present a game-based security definition that unifies all these scenarios and also implies resistance against other types of attacks.

2.1.2.3 The UM protocol

The Unified Model (*UM*) protocol [93] has been designed as an improved version of the original Diffie-Hellman protocol. The protocol does not use signatures and provides *implicit key authentication.* This means that if $\mathcal{A}$ assumes that he shares a key k with $\mathcal{B}$ and the adversary did not learn any forbidden secrets, then $\mathcal{B}$ is the only agent who can compute the key k.

Before starting the *UM* protocol, the agents $\mathcal{A}$ and $\mathcal{B}$ know their own long-term private key a (respectively b) and the other agent's long-term public key $\hat{\mathcal{B}} = g^b$ (respectively $\hat{\mathcal{A}} = g^a$). In the first step, $\mathcal{A}$ chooses his ephemeral private key x and sends g^x to $\mathcal{B}$. Then, $\mathcal{B}$ receives the message from $\mathcal{A}$ as X and chooses his own ephemeral private key y. He then sends g^y to $\mathcal{A}$ and computes his key. $\mathcal{A}$ receives the message from $\mathcal{B}$ as Y and also computes the key. The shared key is $h(g^{ab}, g^{xy}, \mathcal{A}, \mathcal{B}, X, Y)$, i.e., the hash of the static DH key, the ephemeral DH key, the identities of both participants, and the exchanged messages.

$\mathcal{A}$		$\mathcal{B}$
long-term key pair: $\big(a, \hat{\mathcal{A}} = g^a\big)$		long-term key pair: $\big(b, \hat{\mathcal{B}} = g^b\big)$
choose random exponent x		
$X := g^x$	$\xrightarrow{X}$	
		choose random exponent y
		$Y := g^y$
	$\xleftarrow{Y}$	
compute key $h\big(\hat{\mathcal{B}}^a, Y^x, \mathcal{A}, \mathcal{B}, X, Y\big)$		compute key $h\big(\hat{\mathcal{A}}^b, X^y, \mathcal{A}, \mathcal{B}, X, Y\big)$

Figure 2.4. The *UM* protocol.

Note that the *UM* protocol provides resilience against leakage of ephemeral keys in certain scenarios where the *SIGDH* protocol does not. For example, even if x and y are revealed, the adversary cannot compute g^{ab} which is required as input to the hash function. As the *SIGDH* protocol, the *UM* protocol is resilient against UKS attacks. This is ensured by including the identities as inputs to the hash function. Unlike the *SIGDH* protocol, the *UM* protocol is not resilient against KCI attacks and does not provide PFS. To attack KCI, an adversary can use a to compute g^{ab} from $\mathcal{B}$'s public key and impersonate $\mathcal{B}$ to $\mathcal{A}$. To attack PFS, the adversary can send g^z to $\mathcal{B}$ and reveal b after $\mathcal{B}$ has accepted the session key $h(g^{ab}, g^{yz}, \mathcal{A}, \mathcal{B}, X, Y)$. Then, the adversary can compute the session key using $\hat{\mathcal{A}}$, Y, b, and z. A similar generic attack which works for all two-round protocols that do not use the long-term private keys in the computation of the ephemeral public keys has been described in [103]. This attack can be prevented for *UM* by key confirmation messages where both participants prove that they can compute the session key and agree on it. The resulting protocol has three rounds instead of two. Note that the original *UM* protocol does not include the exchanged messages and identities in the key derivation function. But adding these is a common protocol transformation to achieve resistance against certain types of attacks, see [26, 95, 96, 128]. We use a one-pass version of the UM protocol as a running example in Chapter 3 and analyze the security properties achieved by the *UM* protocol in its different versions with our Tamarin tool in Section 3.5.

2.1.2.4 The eCK Model

The eCK security model [111] is defined in terms of a game played between the adversary and a challenger. The adversary is given access to a finite number of protocol oracles Π_t indexed by integers t which we call the (external) session identifiers. We assume that each oracle Π_t models a protocol execution by an actor A chosen from a fixed set of agents $\mathbb{A}$. We denote the actor of the session t with t_{actor}. We use $t_{\text{role}} \in \{\mathcal{A}, \mathcal{B}, \bot\}$ to denote t's role,

$t_{\text{peer}} \in \mathbb{A} \cup \{\bot\}$ to denote t's peer, and $t_{\text{sent}}, t_{\text{received}} \in (\{0, 1\}^*)^*$ to denote t's exchanged messages. The $\bot$ is used to denote that the corresponding value is not determined yet. In addition to the protocol oracles, the challenger also maintains a mapping from identities to long-term keys that is used by the protocol oracles.

Queries. The adversary can perform the following queries on the challenger:

– *Start*(t, R, B): The oracle Π_t starts role R with peer B. The execution proceeds until Π_t waits for a message, fails, or accepts a session key. This query returns all messages sent by the session in the activation and a value that denotes if the session waits for another message, failed, or accepted a session key.

– *Send*(t, m): If Π_t waits for the reception of a message, then the message m is delivered to Π_t. The execution then proceeds until Π_t waits for a message, fails, or accepts a session key. This query returns all messages sent by the session in the activation and a value that denotes if the session waits for another message, failed, or accepted a session key.

– *RevealEphk*(t): This query returns the ephemeral private key of Π_t.

– *RevealSessk*(t): If t has accepted a session key k, this query returns k. Otherwise, it fails.

– *RevealLtk*(A): This query returns the long-term private key of agent A.

– *Test*(t): This query designates the session t as the *test session*. If t has accepted a session key k, then the challenger flips a bit b_C and returns k if $b_C = 0$ and a random bitstring of the right length otherwise. If t has not accepted or there is already a test session, the query fails.

We also assume that the adversary can query the challenger for the public keys of agents.

Clean sessions. For a given test session t, certain queries are forbidden to the adversary. To define the forbidden queries, we first define matching sessions. A session t *matches* a session t' if $\{t_{\text{role}}, t'_{\text{role}}\} = \{\mathcal{A}, \mathcal{B}\}$ and $(t_{\text{actor}}, t_{\text{peer}}, t_{\text{sent}}, t_{\text{received}}) = (t'_{\text{peer}}, t'_{\text{actor}}, t'_{\text{received}}, t'_{\text{sent}})$. A session t is *clean* if none of the following happened.

1. The adversary has performed *RevealSessk*(t).
2. The adversary has performed *RevealLtk*(t_{actor}) and *RevealEphk*(t).
3. There is a matching session t' and
   a) the adversary has performed *RevealSessk*(t'), or
   b) the adversary has performed *RevealLtk*(t'_{actor}) and *RevealEphk*(t').
4. There is no matching session and the adversary has performed *RevealLtk*(t_{peer}).

Game and Advantage. The eCK game is defined such that the adversary interacts with the challenger and outputs a bit b_A. The adversary wins if he successfully issued the query *Test*(t), the test session t is clean, and $b_C = b_A$. The advantage for a given adversary $\mathcal{M}$ is defined as $Adv^{\Pi}_{\text{eCK}}(\mathcal{M}) = \left| Pr\left[\mathcal{M} \text{ wins the game}\right] - \frac{1}{2} \right|$ where the probability is taken over the random choices of the challenger and $\mathcal{M}$. The advantage denotes how much better $\mathcal{M}$ is than an adversary that simply guesses b_C. A protocol Π is defined as eCK-*secure* if matching sessions compute the same key and the advantage for all adversaries is negligible in the security parameter k. The first condition ensures that if the adversary is passive, then the participants compute the same key. The second condition formalizes the secrecy of the session key. There is no way for the adversary to distinguish the *real* session key from a *random* session key.

Discussion. The eCK model captures UKS attacks, i.e., all protocols that are secure in the eCK model are resilient against UKS attacks as defined earlier. To see why this holds, consider the case where A thinks he is sharing a key k with B and B thinks he shares the same key k with $C \neq A$. Then these sessions are not matching and the adversary can choose A's session as the test session and reveal the session key of B's session to win the game. KCI attacks are also captured since the long-term key of the test session's actor can be revealed as long as the ephemeral key is not revealed. The eCK model does not capture PFS since secrecy of the test session's key in the case where both long-term keys are revealed is only guaranteed for the case captured by (3.), but not for case captured by (4.). If there is no matching session, i.e., if the adversary was not passive with respect to the test session, the long-term key of the peer can never be revealed. This notion is often called *weak PFS (wPFS)*. If PFS is desired, the strengthened variant eCK$^{\mathrm{PFS}}$ [110] of the eCK model can be obtained by replacing (4.) with the following condition.

4. There is no matching session and the adversary has performed $\mathit{RevealLtk}(t_{\mathrm{peer}})$ before the session t has accepted.

Here, PFS is captured since in both cases (3.) and (4.), the long-term keys of the test session's actor and peer can be revealed *after* the test session has accepted the session key.

The first protocol proved secure in the eCK model was the NAXOS protocol [111]. We formalize NAXOS security in a symbolic version of the eCK model in Section 3.5.2 and use Tamarin to obtain an unbounded security proof. We also use Tamarin to find an attack on NAXOS in the eCK$^{\mathrm{PFS}}$ model.

2.1.3 Tripartite and Identity-Based Key Exchange Protocols

In this section, we first present the mathematical primitives employed by many tripartite and identity-based key exchange protocols. Then, we consider the Joux protocol and the RYY protocol as examples of such protocols.

2.1.3.1 Bilinear Pairings

We assume given two DH groups $\mathbb{G}_1 = \langle P \rangle$ and $\mathbb{G}_2 = \langle g \rangle$ of the same prime order p. We write $\mathbb{G}_1$ as an additive group and $\mathbb{G}_2$ as a multiplicative group. All elements of $\mathbb{G}_1$ can be written as $[n]P$ and all elements of $\mathbb{G}_2$ can be written as g^n where $n \in \mathbb{Z}_p$ in both cases. Here, $[n]P$ denotes the n-fold sum $\Sigma_{i=1}^{n} P$ of P. The additive group $\mathbb{G}_1$ is usually an elliptic curve group and $[n]P$ is also called multiplication of the point P with the scalar n. The multiplicative group $\mathbb{G}_2$ is usually a subgroup of the multiplicative group of a finite field $\mathbb{F}_{q^l}$ for a prime q.

We assume that there is a map $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ such that $\hat{e}([n]P, [m]P) = \hat{e}(P, P)^{nm}$, i.e., it is bilinear. Note that this implies that $\hat{e}(R, S) = \hat{e}(S, R)$ for all $R, S \in \mathbb{G}_1$. We also assume that the map is non-degenerate, i.e., $\hat{e}(P, P)$ is a generator of $\mathbb{G}_2$. We call such a map a bilinear pairing. Such maps have proved very useful in the design of cryptographic primitives and key exchange protocols. For example, the first identity-based encryption scheme was based on bilinear pairings. In the design of key agreement protocols, bilinear pairings are mostly used for tripartite key exchange protocols and identity-based key exchange protocols.

2.1.3.2 Tripartite Key Agreement and the Joux Protocol

The goal of a tripartite key exchange protocol is to establish a key that is shared between three agents. There have been several protocols based on DH groups as shown before, but all of them require more than one round. The Joux protocol was the first single-round protocol and uses bilinear pairings.

The Joux protocol with signatures shown in Figure 2.5 can be seen as a generalization of the Diffie-Hellman protocol to three parties. Each party signs the concatenation of the intended partners and his ephemeral public key $[u]P$ for $u \in \{x, y, z\}$. Then, the ephemeral public keys from the peers are received and combined using $\hat{e}$. The result is exponentiated with the ephemeral private key to obtain the shared group element $\hat{e}(P, P)^{xyz} \in \mathbb{G}_2$. Finally, this group element is hashed together with the concatenation of the participating agents (in the right order).

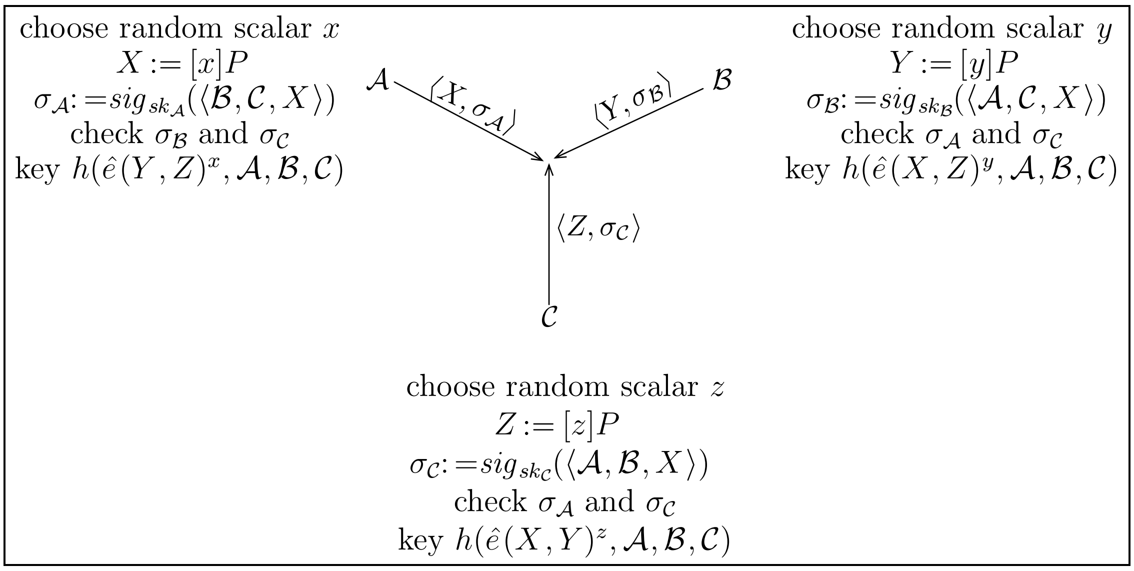

choose random scalar x
$X := [x]P$
$\sigma_{\mathcal{A}} := sig_{sk_{\mathcal{A}}}(\langle \mathcal{B}, \mathcal{C}, X \rangle)$
check $\sigma_{\mathcal{B}}$ and $\sigma_{\mathcal{C}}$
key $h(\hat{e}(Y, Z)^x, \mathcal{A}, \mathcal{B}, \mathcal{C})$
$\mathcal{A}$
$\langle X, \sigma_{\mathcal{A}} \rangle$
$\langle Y, \sigma_{\mathcal{B}} \rangle$
$\mathcal{B}$
choose random scalar y
$Y := [y]P$
$\sigma_{\mathcal{B}} := sig_{sk_{\mathcal{B}}}(\langle \mathcal{A}, \mathcal{C}, X \rangle)$
check $\sigma_{\mathcal{A}}$ and $\sigma_{\mathcal{C}}$
key $h(\hat{e}(X, Z)^y, \mathcal{A}, \mathcal{B}, \mathcal{C})$
$\langle Z, \sigma_{\mathcal{C}} \rangle$
$\mathcal{C}$
choose random scalar z
$Z := [z]P$
$\sigma_{\mathcal{C}} := sig_{sk_{\mathcal{C}}}(\langle \mathcal{A}, \mathcal{B}, X \rangle)$
check $\sigma_{\mathcal{A}}$ and $\sigma_{\mathcal{C}}$
key $h(\hat{e}(X, Y)^z, \mathcal{A}, \mathcal{B}, \mathcal{C})$

Figure 2.5. The *SIGJOUX* protocol.

We formalize perfect forward secrecy of the *SIGJOUX* protocol in Section 3.4.2 and perform an automated analysis with TAMARIN in Section 3.5.

2.1.3.3 Identity-based Key Exchange and the *RYY* Protocol

The *RYY* protocol is an identity-based key exchange protocol that uses bilinear pairings as described in the previous section. Additionally, we assume given a hash function $H: \{0,1\}^k \rightarrow \mathbb{G}_1$. The setting for this protocol is as follows. There is a key generation center (KGC) that knows a master key $s \in \mathbb{Z}_p$. The public key for an arbitrary identity ID is just $H(ID)$. Hence, knowledge of the hash function H and the group $\mathbb{G}_1$ is sufficient to encrypt messages for an arbitrary identity. The owner of the identity ID can obtain his private key $d_{ID} = [s]H(ID)$ from the KGC after proving that he is ID. Note that $\mathcal{A}$ and $\mathcal{B}$ can compute a common key $\hat{e}(H(\mathcal{A}), d_{\mathcal{B}}) = \hat{e}(H(\mathcal{A}), H(\mathcal{B}))^s = \hat{e}(d_{\mathcal{A}}, H(\mathcal{B}))$. To compute this key, it is sufficient to know one of $d_{\mathcal{A}}$, $d_{\mathcal{B}}$, and s. This means that even when there is no active attacker, the KGC can later decrypt all encrypted messages between $\mathcal{A}$ and $\mathcal{B}$. To achieve resilience against such attacks, the *RYY* protocol as shown in Figure 2.6 combines the identity-based key computation with a standard Diffie-Hellman key exchange. The resulting key is the hash of the identity-based key $\hat{e}(H(\mathcal{B}), H(\mathcal{A}))^s$, the ephemeral DH key g^{xy}, the identities, and the exchanged messages.

$\mathcal{A}$		$\mathcal{B}$
long-term secret: $d_{\mathcal{A}} = [s]H(\mathcal{A})$		long-term secret: $d_{\mathcal{B}} = [s]H(\mathcal{B})$
choose random exponent x		
$X := g^x$	$\xrightarrow{X}$	
		choose random exponent y
	$\xleftarrow{Y}$	$Y := g^y$
key $h(Y^x, \hat{e}(d_{\mathcal{A}}, H(\mathcal{B})), \mathcal{A}, \mathcal{B}, X, Y)$		key $h(X^y, \hat{e}(H(\mathcal{A}), d_{\mathcal{B}}), \mathcal{A}, \mathcal{B}, X, Y)$

Figure 2.6. The RYY protocol with key derivation.

The RYY protocol can be considered an identity-based version of the UM protocol. We formalize weak perfect forward secrecy of RYY and perform an automated analysis with Tamarin in Section 3.5.

2.1.4 Physical Protocols

In this section, we describe two scenarios where physical properties of the environment must be established securely. We then describe protocols to achieve these desired goals.

2.1.4.1 Scenario: Contactless Car Entry Systems

Modern cars commonly use contactless entry systems. The car doors are unlocked whenever the corresponding key is sufficiently close to the car. The protocol executed between the car and key establishes two facts. First, the protocol ensures that the car communicates with the right key. This is a classical authentication property, which is usually ensured by executing a protocol that uses a shared secret or public key cryptography. Second, the protocol ensures that the key is sufficiently close to the car. Here, car manufacturers relied on the limited communication range of the keys. The car can only communicate with the key if it is sufficiently close. As shown in [82], such a protocol can be attacked by employing a *fake key* that is close to the car and a *leech* that is close to the real key. The fake key can initiate the protocol and relay all challenges from the car to the leech who queries the real key for the responses.

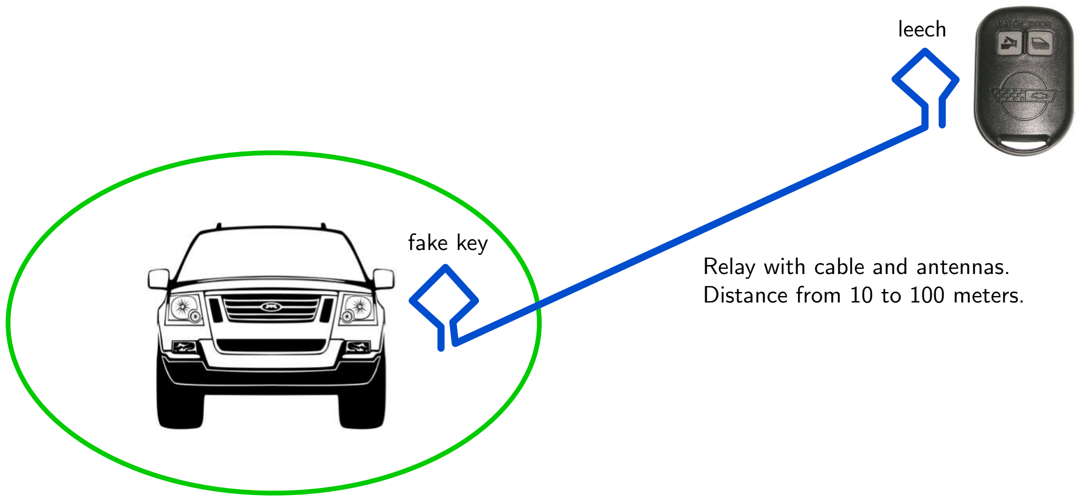

leech
fake key
Relay with cable and antennas.
Distance from 10 to 100 meters.

Figure 2.7. Relay attack on contactless car entry system.

This type of attack was first considered in [66] under the name Mafia fraud. In [34], an attack scenario with fake ATMs is described. Here, a man-in-the-middle attack is employed to forward the messages back and forth between the real ATM and the real card that is inserted into the fake ATM.

2.1.4.2 Authenticated Ranging Protocols

To establish the required facts for a contactless car entry system, an authenticated ranging protocol can be used. An authenticated ranging protocol is executed between a verifier $\mathcal{V}$ and a prover $\mathcal{P}$. The goal of the protocol is for the verifier $\mathcal{V}$ to establish an upper bound on the distance to an honest prover $\mathcal{P}$. Most protocols that achieve authenticated ranging are based on roundtrip-time measurements by the verifier. In the previous example, the car would be the verifier and the key would be the prover. After establishing an upper bound on the distance to the key, the car would decide if the key is close enough to unlock the door. Figure 2.8 shows an example protocol. Here, $\mathcal{V}$ send a fresh nonce nv as a challenge at time $t_S^{\mathcal{V}}$. The prover $\mathcal{P}$ notes the time $t_R^{\mathcal{P}}$ when he received the challenge and decides on a time $t_S^{\mathcal{P}}$ when to send the response, based on the required computation time. Then, $\mathcal{P}$ computes a signature on the challenge nv and the computation delay $t_S^{\mathcal{P}} - t_R^{\mathcal{P}}$, which he sends back to $\mathcal{V}$ at time $t_S^{\mathcal{P}}$. $\mathcal{V}$ notes the reception time $t_R^{\mathcal{V}}$, checks the signature, and computes the upper bound on the distance as $(t_R^{\mathcal{V}} - t_S^{\mathcal{V}} - \delta) * \frac{c}{2}$, where c is the speed of light. Since the adversary can delay the transmissions, but not speed up the transmissions with respect to the assumed line-of-sight transmission with speed of light, the computed value is an upper bound on the distance.

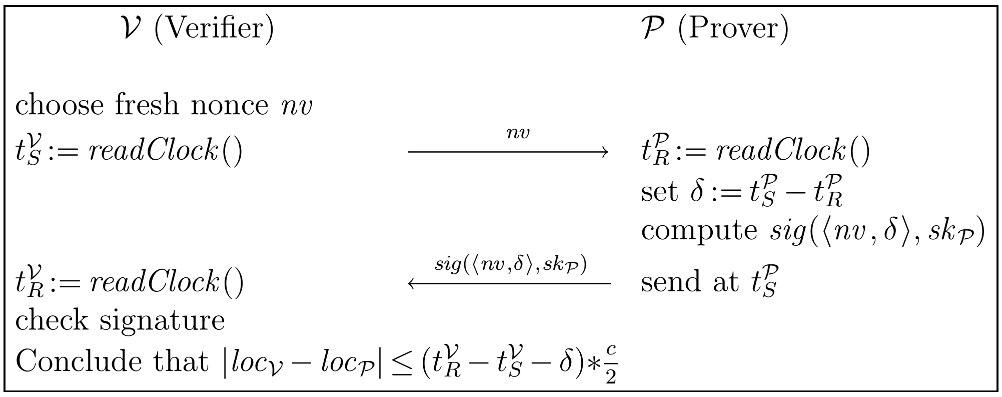

$\mathcal{V}$ (Verifier)
$\mathcal{P}$ (Prover)
choose fresh nonce nv
$t_S^{\mathcal{V}} := readClock()$
nv
$t_R^{\mathcal{P}} := readClock()$
set $\delta := t_S^{\mathcal{P}} - t_R^{\mathcal{P}}$
compute $sig(\langle nv, \delta\rangle, sk_{\mathcal{P}})$
$sig(\langle nv, \delta\rangle, sk_{\mathcal{P}})$
$t_R^{\mathcal{V}} := readClock()$
send at $t_S^{\mathcal{P}}$
check signature
Conclude that $|loc_{\mathcal{V}} - loc_{\mathcal{P}}| \leq (t_R^{\mathcal{V}} - t_S^{\mathcal{V}} - \delta) * \frac{c}{2}$

Figure 2.8. Authenticated Ranging Protocol.

2.1.4.3 Scenario: Location-Based Access Control

Consider now a slightly different scenario where wireless access to sensitive data is controlled by the following access control policy. First, the user, identified by a shared secret or as the holder of a certain private key, must be allowed to access the data. Second, the user must be located in a certain building to access the data. Access to the building is secured by guards who check the identity of the user. In contrast to an access control policy that only takes the first condition into account, this policy also tolerates the loss of key material to outsiders that cannot gain access to the building. This is the main difference to the previous scenario where the owner of the real key (material) is always assumed to be honest.

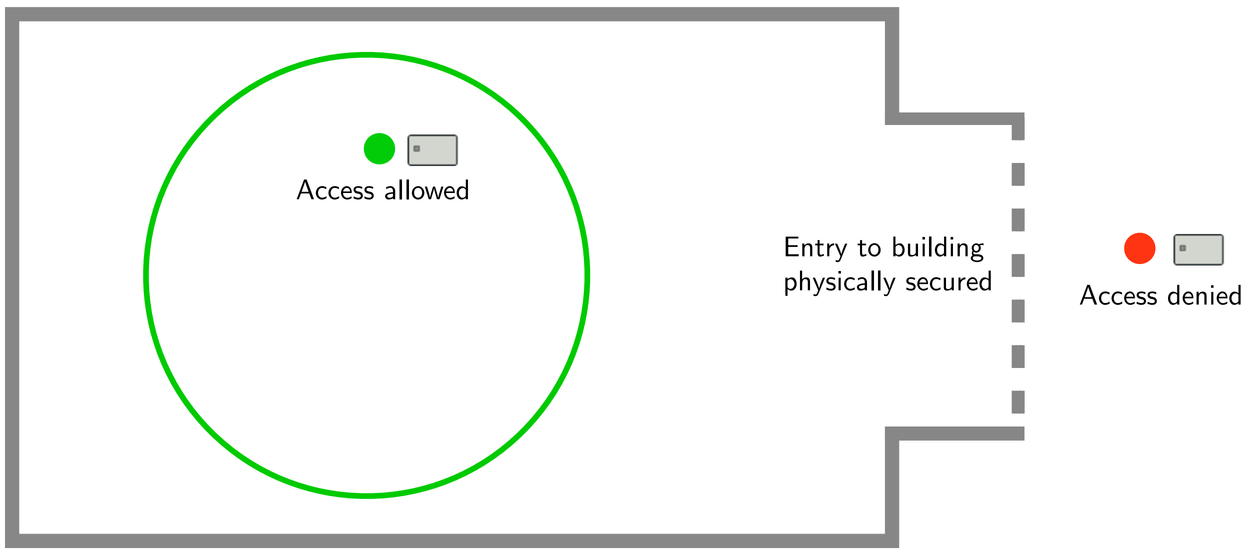


Figure 2.9. Location based access control to sensitive data.

2.1.4.4 Distance Bounding Protocols

For the location-based access control scenario, an authenticated ranging protocol is not the right choice. For example, consider the protocol from Figure 2.8. An outsider who knows the signing key of an authorized user can simply increase δ to appear closer to the verifier than he really is. To handle this additional scenario, *distance bounding protocols* have been introduced in [34]. Usually, the following two types of attacks have been considered in addition to Mafia fraud. In a *distance fraud*, a lone dishonest prover claims to be closer to the verifier than he really is. Later on, resistance against *terrorist fraud* [67] has also been stated as a desirable goal. Here a dishonest prover P helps another dishonest prover P' to appear closer than he really is. Since P' does not trust P, P' is not allowed to give up his long-term keys to P or even help P in future fraud attempts, depending on the exact definition of terrorist fraud. A formal computational definition of terrorist fraud is given in [81] and several protocols are investigated with respect to this notion in [74]. Since no published protocol is secure with respect to this definition, it is still an open question if this computational definition is the right one.

2.2 Notational Preliminaries

In the following, we use the following mathematical notation. We use S^* to denote the set of sequences over S. For a sequence s, we write s_i for the i-th element, $|s|$ for the length of s, and $idx(s) = \{1, \ldots, |s|\}$ 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 where $|s| = k$, and $s \cdot s'$ to denote the concatenation of the sequences s and s'. $S^\sharp$ denotes the set of finite multisets with elements from S. We annotate the usual set operations with $\sharp$ to denote the corresponding multiset operations, e.g., we use $\cup^\sharp$ to denote multiset-union. For a sequence s, $mset(s)$ denotes the corresponding multiset and $set(s)$ the corresponding set. For a function f, we write $f[a \mapsto b]$ to denote the function that maps a to b and c to $f(c)$, for all $c \neq a$. For a relation $\rightarrow$ with $\rightarrow \subseteq S \times S$, we use $\rightarrow^+$ to denote the transitive closure and $\rightarrow^*$ to denote the transitive-reflexive closure.

2.3 Term Rewriting

As previously stated, we abstract cryptographic messages and operations on them by terms and the properties of these operations by equations. This naturally allows us to use variables to reason symbolically about sets of messages. Most methods for this approach come from the area of term rewriting. We therefore recall some standard notions from term rewriting following [65], [86], and [79].

2.3.1 Standard Notions

Order sorted term algebras. An *order-sorted signature* $\mathbf{\Sigma} = (S, \leq, \Sigma)$ consists of a set of sorts S, a partial order $\leq$ on S, and a set Σ of function symbols associated with sorts with the following two properties. First, for every $s \in S$, the connected component C of s in $(S, \leq)$ has a *top sort* denoted $top(s)$ such that $c \leq top(s)$ for all $c \in C$. Second, for every $f\colon s_1 \times \ldots \times s_k \to s$ in Σ with $k \geq 1$, there is an $f\colon top(s_1) \times \ldots \times top(s_k) \to s$ in Σ called the *top sort overloading* of f. If the sort hierarchy is clear from the context, we often use Σ instead of $\mathbf{\Sigma}$.

If there is only one sort, we say the corresponding signature is *unsorted* and we write Σ^k to denote all k-ary function symbols in Σ. We assume that there are countably infinite sets of variables $\mathcal{V}_s$ for all sorts $s \in S$ and define $\mathcal{V} = \biguplus_{s \in S} \mathcal{V}_s$. We use $x{:}s$ to denote variables from $\mathcal{V}_s$. For a signature $\mathbf{\Sigma}$ and an arbitrary subset $\mathcal{V}' \subseteq \mathcal{V}$, $\mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V}')_s$ denotes the set of well-sorted terms of sort s constructed over $\Sigma \cup \mathcal{V}'$. We use $\mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V}')$ to denote the set of *all* well-sorted terms. For a set of constants $\mathcal{C}$, we also use $\mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V}' \cup \mathcal{C})$ to denote $\mathcal{T}_{\mathbf{\Sigma} \cup \mathcal{C}}(\mathcal{V}')$.

Positions, subterms, vars, and contexts. A *position* is a sequence of natural numbers. For a term t and a position p, we denote the *subterm of t at position p* with $t|_p$. Formally, $t|_p$ is defined as (a) t if $p = [\,]$, (b) $t_i|_{p'}$ if $p = [i] \cdot p'$ and $t = f(t_1, \ldots, t_k)$ for $1 \leq i \leq k$, and (c) undefined otherwise. We say p is a *valid position in t* if $t|_p$ is defined. For two positions p and p', *p is above p'* if p is a proper prefix of p'. In this case, *p' is below p*. If p is unequal to p' and neither above nor below p', then both positions are *incomparable*. We say p and p' are *siblings* if $|p| = |p'|$, $p_i = p'_i$ for $1 \leq i < |p|$, and $p_{|p|} \neq p'_{|p|}$.

Furthermore, we use $t[s]_p$ to denote the term t where the occurrence of the subterm $t|_p$ at position p has been replaced by s. We use $root(t)$ to denote f if $t = f(t_1, \ldots, t_k)$ for some $f \in \Sigma$ and t itself otherwise. The set $St(t)$ of *syntactic subterms* of a term t is defined in the usual way as $\{t|_p \mid p \textit{ valid position in } t\}$. For a term t, we define $vars(t) = St(t) \cap \mathcal{V}$. A term t is *ground* if $vars(t) = \emptyset$. We use $fvars(\varphi)$ to denote the free variables in a formula φ.

A context C is a term with holes. Holes are distinct variables x_i that occur exactly once in C. We use the notation $C[x_1, \ldots, x_n]$ to indicate that C has the holes $x_1, \ldots, x_n$ and $C[t_1, \ldots, t_n]$ to denote the term where the holes have been replaced by the terms t_i.

Substitutions and replacements. A *substitution* σ is a well-sorted function from $\mathcal{V}$ to $\mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V})$ that corresponds to the identity function except on a finite set of variables which we denote with $dom(\sigma)$. We use $range(\sigma)$ to denote the image of $dom(\sigma)$ under σ and define $vrange(\sigma) = vars(range(\sigma))$. We identify σ with its usual extension to an endomorphism on $\mathcal{T}_{\mathbf{\Sigma}}(\mathcal{V})$ and use the notation $t\sigma$ for the application of σ to the term t. Furthermore, we denote the substitution σ with $dom(\sigma) = \{x_1, \ldots, x_k\}$ and $x_i\sigma = t_i$ by $\{t_1/x_1, \ldots, t_k/x_k\}$. We use id to denote the identity substitution.

A replacement ρ is a function from a finite set of terms to $\mathcal{T}_{\Sigma}(\mathcal{V})$ such that $dom(\rho) \cap St(range(\rho)) = \emptyset$. For a term t, we define t^{ρ} as the unique term where all occurrences of subterms $s = t|_p$ of t, with $s \in dom(\rho)$ such that there is no position p' above p with $t|_{p'}$ in $dom(\rho)$ are replaced by $\rho(s)$.

Equations and equational theories. An *equation* over a signature Σ is an unordered pair $\{s, t\}$ of terms $t, s \in \mathcal{T}_{\Sigma}(\mathcal{V})$ denoted $s \simeq t$. An *equational presentation* is a tuple (Σ, E) where Σ is a signature and E is a set of equations. Given an equational presentation $\mathcal{E} = (\Sigma, E)$, we define the *equational theory* $=_{\mathcal{E}}$ as the smallest Σ-congruence containing all instances of equations of E. We often identify the equational presentation $\mathcal{E}$ and the corresponding equational theory $=_{\mathcal{E}}$. An equation $s \simeq t$ is *regular* if $vars(s) = vars(t)$ and *sort-preserving* if for all substitutions σ, it holds that $s\sigma \in \mathcal{T}_{\Sigma}(\mathcal{V})_s$ if and only if $t\sigma \in \mathcal{T}_{\Sigma}(\mathcal{V})_s$. An equational presentation is regular (respectively sort-preserving) if all its equations are.

Unification and matching. An *$\mathcal{E}$-unifier* of two terms s and t is a substitution σ such that $s\sigma =_{\mathcal{E}} t\sigma$. For $W \subseteq \mathcal{V}$, we use $unif_{\mathcal{E}}^{W}(s, t)$ to denote an $\mathcal{E}$-unification algorithm that returns a set of unifiers of s and t such that for all $\sigma \in unif_{\mathcal{E}}^{W}(s, t)$, $vrange(\sigma) \cap W = \emptyset$. The unification algorithm is *complete* if for all $\mathcal{E}$-unifiers σ of s and t, there is a $\tau \in unif_{\mathcal{E}}^{W}(s, t)$ and a substitution θ such that for all $x \in vars(s, t)$, $(x\tau)\theta =_{\mathcal{E}} x\sigma$. The unification algorithm is *finitary* if for all s and t, it terminates and returns a finite set of unifiers.

An *$\mathcal{E}$-matcher* of two terms t and p is a substitution σ such that $t =_{\mathcal{E}} p\sigma$. The notions of complete and finitary coincide with those for $\mathcal{E}$-unification. We use $match_{\mathcal{E}}(t, p)$ to denote a finitary and complete $\mathcal{E}$-matching algorithm.

Rewriting. A *rewrite rule* over a signature Σ is an ordered pair of terms (l, r) with $l, r \in \mathcal{T}_{\Sigma}(\mathcal{V})$ denoted $l \to r$. A *rewrite system* $\mathcal{R}$ is a set of rewrite rules. $\mathcal{R}$ defines a rewrite relation $\to_{\mathcal{R}}$ with $s \to_{\mathcal{R}} t$ if there is a position p in s, a 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* if there is no infinite sequence $(t_i)_{i \in \mathbb{N}}$ of terms with $t_i \to_{\mathcal{R}} t_{i+1}$. A rewrite system $\mathcal{R}$ is *confluent* if for all terms t, s_1, 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* if it is terminating and confluent. In this case, we use $t{\downarrow_{\mathcal{R}}}$ to denote the unique 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 $t' \to_{R} t''$. A rewriting rule $l \to r$ is *sort-decreasing* if for all substitutions σ, $r\sigma \in \mathcal{T}_{\Sigma}(C)_s$ implies $l\sigma \in \mathcal{T}_{\Sigma}(C)_s$. A rewrite system is sort-decreasing if all its rules are.

A rewrite system $\mathcal{R}$ is *subterm-convergent* if 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 normal form with respect to $\mathcal{R}$. We say a function symbol f is *irreducible* with respect to such a rewriting system $\mathcal{R}$ if there is no $l \to r \in \mathcal{R}$ with $root(l) = f$. We use $\mathcal{R}^{\simeq}$ to denote the set of equations obtained from a rewrite system by replacing $\to$ with $\simeq$ in the rewrite rules.

2.3.2 Rewriting Modulo and Finite Variants

Rewriting modulo. Rewriting is often used to reason about equality modulo an equational theory $\mathcal{E}$. If the equations in $\mathcal{E}$ can be oriented to obtain a convergent rewriting system $\mathcal{R}$, then $\mathcal{R}$ can be used to check equality since $t =_{\mathcal{E}} s$ if and only if $t{\downarrow_{\mathcal{R}}} = s{\downarrow_{\mathcal{R}}}$. To cope with equational theories containing equations that cannot be oriented such as commutativity, the notion of *$\mathcal{R}$,$\mathcal{AX}$-rewriting* for a rewrite system $\mathcal{R}$ and an equational theory $\mathcal{AX}$

has been introduced in [142]. The rewriting relation $\rightarrow_{\mathcal{R},\mathcal{AX}}$ is defined as $s \rightarrow_{\mathcal{R},\mathcal{AX}} t$ if there is a position p in s, a rewriting rule $l \rightarrow r \in \mathcal{R}$, and a substitution σ such that $s|_p =_{\mathcal{AX}} l\sigma$ and $s[r\sigma]_p = t$. If $\mathcal{AX}$-matching is decidable, then the relation $\rightarrow_{\mathcal{R},\mathcal{AX}}$ is also decidable. To use rewriting to reason about equality modulo $\mathcal{E}$, the following properties are required.

Coherence and decomposition. We say $\mathcal{R}$,$\mathcal{AX}$ is *convergent* if the relation $\rightarrow_{\mathcal{R},\mathcal{AX}}$ is convergent. In this case, we denote the unique normal form of t with respect to $\mathcal{R}$,$\mathcal{AX}$-rewriting by $t{\downarrow_{\mathcal{R},\mathcal{AX}}}$. We say $\mathcal{R}$,$\mathcal{AX}$ is *coherent* if for all t_1, t_2, and t_3, it holds that $t_1 \rightarrow_{R,\mathcal{AX}} t_2$ and $t_1 =_{\mathcal{AX}} t_3$ implies that there are t_4 and t_5 such that $t_2 \rightarrow^*_{R,\mathcal{AX}} t_4$, $t_3 \rightarrow^+_{R,\mathcal{AX}} t_5$, and $t_4 =_{\mathcal{AX}} t_5$. If $(\boldsymbol{\Sigma}, \mathcal{R} \cup \mathcal{AX})$ is an equational presentation of $=_{\mathcal{E}}$ and $\mathcal{R}$,$\mathcal{AX}$ is convergent and coherent, then $t =_{\mathcal{E}} s$ if and only if $t{\downarrow_{\mathcal{R},\mathcal{AX}}} = s{\downarrow_{\mathcal{R},\mathcal{AX}}}$. See [142, 94] for details.

We call $(\boldsymbol{\Sigma}, \mathcal{R}, \mathcal{AX})$ a *decomposition* of $\mathcal{E}$ if the following holds:

1. $(\boldsymbol{\Sigma}, \mathcal{R}^{\simeq} \cup \mathcal{AX})$ is an equational presentation of $=_{\mathcal{E}}$.
2. $\mathcal{AX}$ is regular, sort-preserving and all equations contain only variables of top-sort.
3. $\mathcal{R}$ is sort-decreasing and $\mathcal{R}$,$\mathcal{AX}$ is convergent and coherent.
4. There is a complete and finitary $\mathcal{AX}$-unification algorithm.

Example 2.1. Consider the unsorted equational theory $ACUN$ that models Xor with $\Sigma_{ACUN} = \{\mathsf{0}, +\}$ and $E_{ACUN} = AC \cup \{x + \mathsf{0} \simeq x, x + x \simeq \mathsf{0}\}$ for $AC = \{(x + y) + z \simeq x + (y + z), x + y \simeq y + x\}$. The last two equations in E_{ACUN} can be oriented to obtain $\mathcal{R}_{ACUN} = \{x + \mathsf{0} \rightarrow x, x + x \rightarrow \mathsf{0}\}$ and $\mathcal{R}_{ACUN}$,AC is convergent. It is easy to see that $x + (x + y) =_{ACUN} y$, but

$$(x + (x + y)){\downarrow_{\mathcal{R}_{ACUN},AC}} = (x + (x + y)) \neq_{AC} y = y{\downarrow_{\mathcal{R}_{ACUN},AC}} .$$

The problem is that $\mathcal{R}_{ACUN}$,AC is not coherent and the relation $\leftrightarrow^*_{\mathcal{R}_{ACUN},AC}$ is therefore strictly smaller than $=_{ACUN}$. It is possible to complete $\mathcal{R}_{ACUN}$ with the missing rewrite rule $x + (x + y) \rightarrow y$ to obtain $\mathcal{R}'_{ACUN}$. Then $(\Sigma, \mathcal{R}'_{ACUN}, AC)$ is a decomposition of $ACUN$.

Finite variants. For an equational theory $\mathcal{E}$, we define the $\mathcal{E}$-instances of a term t as $insts_{\mathcal{E}}(t) = \{t' | \exists \sigma . t\sigma =_{\mathcal{E}} t'\}$. We use $ginsts_{\mathcal{E}}(t)$ to denote the set of ground $\mathcal{E}$-instances of t. To reason about $\mathcal{E}$-instances using a decomposition $(\Sigma, \mathcal{R}, \mathcal{AX})$ of $\mathcal{E}$, the finite variant property is often useful.

A decomposition $(\Sigma, \mathcal{R}, \mathcal{AX})$ of an equational theory $\mathcal{E}$ has the *finite variant property* if 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 a substitution θ and $i \in \{1, \ldots, k\}$ with

1. $(t\sigma){\downarrow_{\mathcal{R},\mathcal{AX}}} =_{\mathcal{AX}} (t\tau_i){\downarrow_{\mathcal{R},\mathcal{AX}}}\theta$ and
2. $x\sigma{\downarrow_{\mathcal{R},\mathcal{AX}}} =_{\mathcal{AX}} (x\tau_i){\downarrow_{\mathcal{R},\mathcal{AX}}}\theta$ for all $x \in vars(t)$.

We call such a set of substitutions a *complete set of* $\mathcal{R}$,$\mathcal{AX}$*-variants of* t. For a decomposition with the finite variant property, we can use folding variant narrowing [79] to compute a complete and minimal set of $\mathcal{R}$,$\mathcal{AX}$-variants of a term.

Example 2.2. Consider the equational theory $\mathcal{E} = (\Sigma, E)$ with $\Sigma = \{g/2, \mathsf{a}/0\}$ and $E = \{g(x, \mathsf{a}) \simeq \mathsf{a}\}$. Then $(\Sigma, \mathcal{R}, \emptyset)$ is a decomposition with the finite variant property for $\mathcal{R} = \{g(x, \mathsf{a}) \rightarrow \mathsf{a}\}$. Consider the term $t = g(x, y)$. The set $\{id, \{\mathsf{a}/y\}\}$ is a complete set of $\mathcal{R}$,$\emptyset$-variants of t. Let t' and σ arbitrary such that $t' = t'{\downarrow_{\mathcal{R}}}$ and $t' =_{\mathcal{E}} t\sigma$, i.e., t' is an irreducible $\mathcal{E}$-instance of t. Since $\{id, \{\mathsf{a}/y\}\}$ is a complete set of $\mathcal{R}$,$\emptyset$-variants of t, t' is either an instance of t (i.e. $t' \in insts_\emptyset((t)id)$) or an instance of $t\{\mathsf{a}/y\}{\downarrow_{\mathcal{R}}} = \mathsf{a}$, i.e., equal to a. This is already ensured by requirement (1) for a complete set of variants. Requirement (2) additionally ensures that σ factorizes through id in the first case and through $\{\mathsf{a}/y\}$ in the second case. If we would drop requirement (2), then $\{id, \{\mathsf{a}/x, \mathsf{a}/y\}\}$ would also be a complete set of variants, even though $\sigma = \{\mathsf{a}/y\}$ would not factorize through $\{\mathsf{a}/x, \mathsf{a}/y\}$. If complete sets of variants are used to perform $\mathcal{E}$-unification [79], both requirements are necessary.

2.3.3 Ordered Completion

We use ordered completion [65] to define the normalization of ground terms with respect to an arbitrary equational theory $\mathcal{E}$. In Chapter 4, we use this notion in our combination proof for impossibility. This technique has been used in similar contexts to prove the correctness of combination results for unification [15] and deducibility [46, 56]. In the following, we assumed an unsorted signature Σ and a set of constant symbols $\mathcal{N}$ different from Σ^0 called names. Let $\succ$ be a total simplification order on ground terms, i.e., for ground terms s_1 and s_2 and a nonempty context C, we have that (i) $s_1 \succ s_2$ or $s_2 \succ s_1$, (ii) $C[s_1] \succ s_1$, and (iii) $s_1 \succ s_2$ implies $C[s_1] \succ C[s_2]$. Additionally, we assume for all $n \in \mathcal{N}$, $c \in \Sigma^0$, and $t \in \mathcal{T}_\Sigma(\mathcal{N}) \setminus (\mathcal{N} \cup \Sigma^0)$ that $c \succ n$ and $t \succ c$. We then use n_{min} to denote the minimum for $\succ$, which is a name. The lexicographic path ordering constructed from a total ordering on $\mathcal{N} \cup \Sigma$, where names are smaller than constants from Σ^0 and constants are smaller than non-constant function symbols, always has these properties (see [65]).

For a given equational theory $\mathcal{E}$ and a total simplification ordering on ground terms $\succ$, we define the ordered rewrite relation $\rightarrow_{(\succ,\mathcal{E})}$ as follows: $t \rightarrow_{(\succ,\mathcal{E})} t'$ if there is a position p of t, an equation $l \simeq r$ in $\mathcal{E}$, and a substitution σ such that $t|_p = l\sigma$, $t' = t[r\sigma]_p$, and $t \succ t'$. We use *ordered completion* for a given equational theory $\mathcal{E}$ to obtain a (possibly infinite) set of equations $O_\mathcal{E}$ such that $=_{O_\mathcal{E}}$ equals $=_\mathcal{E}$ and $\rightarrow_{(\succ,O_\mathcal{E})}$ is convergent on ground terms. For a ground term t, we use $t{\downarrow}_\mathcal{E}^\succ$ to denote the normal form of t with respect to $\rightarrow_{(\succ,O_\mathcal{E})}$.

2.4 Isabelle/HOL and the Inductive Approach

In Chapter 5, we use the theorem prover Isabelle [136]. Moreover, we build on the inductive approach to protocol verification by Paulson [140] to reason about physical protocols. The following paragraphs provide a small summary of Isabelle/HOL and the inductive approach.

Isabelle/HOL. Isabelle is a generic theorem prover with a specialization for higher-order logic (HOL). We will avoid Isabelle-specific details as much as possible or explain them in context as needed. Here we limit ourselves to few comments on typing. A function f from type α to β is denoted $f\colon \alpha \rightarrow \beta$ and $f x \equiv t$ defines the function f with parameter x as the term t. We write $\alpha \times \beta$ for the product type of α and β. We use the predefined list type α *list* where $xs.x$ denotes the list xs extended by the element x. Algebraic data types are defined using the **datatype** declaration. Central to our work is the ability to define (parameterized) inductively defined sets. These sets are defined by sets of rules and denote the least set closed under the rules. Given an inductive definition, Isabelle generates a rule for proof by induction.

Inductive Approach. Paulson has introduced an inductive approach to security protocol verification. The approach is based on a trace-based interleaving semantics, which gives a semantics to distributed systems as the set of traces describing all possible interleaved events executed by the involved agents. In particular, protocols are modeled by rules describing the protocol steps executed by honest agents and possible intruder actions. The rules constitute an inductive definition that defines the protocol's semantics as an infinite set of communication traces, each trace being a finite list of communication events. Security properties are specified as predicates on traces. Protocol security is then proved by induction on traces using an induction principle derived from the protocol rules. Paulson formalized his approach within higher-order logic in the Isabelle/HOL system and used it to prove security properties for a wide range of security protocols.

Chapter 3
Automated Protocol Analysis

In this chapter, we present a method to analyze security protocols that use DH exponentiation and bilinear pairings without bounding the number of protocol sessions. We have implemented the method in a tool, called the TAMARIN prover, and demonstrated its effectiveness in numerous case studies from the area of key exchange protocols, tripartite group key exchange protocols, and identity-based key exchange protocols. Given a specification of the protocol and the adversary model, TAMARIN automatically searches for a proof that the protocol is secure in the given model with respect to an unbounded number of sessions. If TAMARIN terminates without succeeding in the proof, it provides a counterexample, i.e., an attack on the protocol with respect to the given adversary model.

In the following, we first define our security protocol model based on multiset rewriting and many-sorted first-order logic. Then, we present our verification theory, which simplifies reasoning about the security protocol model. Next, we present the constraint solving rules that we use to analyze protocols. Then, we extend the previously introduced approach from DH exponentiation to bilinear pairings and AC operators. Finally, we describe the TAMARIN tool and the case studies.

3.1 Security Protocol Model

We model the execution of a security protocol in the context 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. The adversary and the protocol interact by updating network messages and freshness information. Adversary capabilities and protocols are specified jointly as a (labeled) multiset rewriting system. Security properties are modeled as trace properties of the transition system. In the following, we first describe cryptographic messages. Then, we define how protocols are specified and executed. Finally, we define our property specification language and illustrate our protocol model with an example.

3.1.1 Cryptographic Messages

To model cryptographic messages, we use an order-sorted term algebra and an equational theory. The function symbols in the signature model the algorithms that can be applied to messages such as encryption or decryption and the equational theory models the properties of the algorithms. We use the set of sorts consisting of the top sort *msg* and two incomparable subsorts *fr* and *pub* for fresh and public names. We assume there are two countably infinite sets FN and PN of *fresh* and *public names*. We use fresh names to model random messages such as keys or nonces and public names to model known constants such as agent identities. To model DH exponentiation, we define the unsorted signature

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

where all function symbols are of sort $msg \times \ldots \times msg \rightarrow msg$. Here, $g\hat{}a$ denotes exponentiation of the message g with the exponent a. The remaining function symbols model the exponents as an abelian group. The algebraic properties of DH exponentiation are modeled by the equations

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

which capture that repeated exponentiation in a DH group corresponds to multiplication of the exponents, exponentiation with 1 is the identity, and the exponents form an abelian group. We define the equational theory for DH as $\mathcal{DH} = (\Sigma_{\mathcal{DH}}, E_{\mathcal{DH}})$.

For additional cryptographic operators, we assume given an unsorted signature $\Sigma_{\mathcal{ST}}$ disjoint from $\Sigma_{\mathcal{DH}}$ and a subterm-convergent rewrite system $\mathcal{R}_{\mathcal{ST}}$ over $\Sigma_{\mathcal{ST}}$. We define the combined signature $\Sigma_{\mathcal{DH}_e} = \Sigma_{\mathcal{ST}} \cup \Sigma_{\mathcal{DH}}$ and use $\mathcal{T}$ to denote $\mathcal{T}_{\Sigma_{\mathcal{DH}_e} \cup \mathsf{FN} \cup \mathsf{PN}}(\mathcal{V})$ and $\mathcal{M}$ to denote the ground terms in $\mathcal{T}$ which we call *messages*. We define the combined equational theory as $\mathcal{DH}_e = (\Sigma_{\mathcal{DH}} \cup \Sigma_{\mathcal{ST}}, E_{\mathcal{DH}} \cup \mathcal{R}_{\mathcal{ST}}^{\simeq})$. In the remainder of this chapter, we use g to denote a public name that is used as a fixed generator of a DH group and a, b, c, and k to denote fresh names.

Example 3.1. 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. This is equal to $\mathsf{g}\hat{}((\mathsf{a}*\mathsf{b})*\mathsf{a}^{-1})$ and can be further simplified to $\mathsf{g}\hat{}\mathsf{b}$.

Note that our equational theory does not support protocols that perform multiplication in the DH group $\mathbb{G}$. To define such protocols, an additional function symbol $\cdot$ 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 $(\mathsf{g}\hat{}\mathsf{a} \cdot \mathsf{g}\hat{}\mathsf{b})\hat{}\mathsf{c} = (\mathsf{g}\hat{}\mathsf{a})\hat{}\mathsf{c} \cdot (\mathsf{g}\hat{}\mathsf{b})\hat{}\mathsf{c}$ holds in all DH groups, but does usually not hold if we replace $\cdot$ by $*$. We further discuss our choice of the equational theory $\mathcal{DH}$ in Section 6.1.1.

Example 3.2. As an example of a subterm-convergent theory that can be used, consider $\mathcal{DY} = (\Sigma_{\mathcal{DY}}, \mathcal{R}_{\mathcal{DY}})$ defined by

$$\Sigma_{\mathcal{DY}} = \left\{ \begin{array}{llll} \langle\_,\_\rangle & \mathit{fst}(\_) & \mathit{snd}(\_) & \\ \mathit{pk}(\_) & \mathit{enc}(\_,\_) & \mathit{dec}(\_,\_) & \\ \mathit{true} & \mathit{sig}(\_,\_) & \mathit{verify}(\_,\_) & h(\_) \end{array} \right\}$$

and

$$\mathcal{R}_{\mathcal{DY}} = \left\{ \begin{array}{ll} \mathit{fst}(\langle x, y\rangle) \rightarrow x & \mathit{snd}(\langle x, y\rangle) \rightarrow y \\ \mathit{dec}(\mathit{enc}(x, \mathit{pk}(y)), y) \rightarrow x & \\ \mathit{verify}(\mathit{sig}(x, y), x, \mathit{pk}(y)) \rightarrow \mathit{true} & \end{array} \right\}.$$

This theory models pairing and projection, public key encryption, message-hiding signatures, and a hash function. Note that we often use $\langle t_1, t_2 \ldots, t_{k-1}, t_k\rangle$ as an abbreviation for $\langle t_1, \langle t_2, \ldots \langle t_{k-1}, t_k\rangle \ldots\rangle\rangle$ and $h(t_1, \ldots, t_k)$ as an abbreviation for $h(\langle t_1, \ldots, t_k\rangle)$.

3.1.2 Labeled Multiset Rewriting with Fresh Names

We use multiset rewriting to specify the concurrent execution of protocol and adversary. Multiset rewriting is commonly used to model concurrent systems since 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 verification by Cervesato et al. [41] and in Maude-NPA [77]. Like these two approaches, we extend standard multiset rewriting with support for creating fresh names. Additionally, we introduce persistent facts and enrich rewriting rules with actions to obtain *labeled* transition systems. Note that it is possible to give a translation from processes in the applied pi calculus as, for example, used in ProVerif [28] into our dialect of multiset rewriting.

State

We model the states of our transition system as finite multisets of facts. Facts are built from terms over a fact signature. Formally, we assume given an unsorted signature $\Sigma_{\mathcal{F}}$ partitioned into *linear* and *persistent* fact symbols. We define the set of *facts* as the set $\mathcal{F}$ consisting of all facts $F(t_1, \ldots, t_k)$ such that $t_i \in \mathcal{T}$ and $F \in \Sigma_{\mathcal{F}}^k$. We denote the set of *ground facts* by $\mathcal{G}$. We say that a fact $F(t_1, \ldots, t_k)$ is *linear* if F is linear and *persistent* if F is persistent. Linear facts model resources that can only be consumed once, whereas persistent facts model inexhaustible resources that can be consumed arbitrarily often. We assume that the linear fact symbol Fr is included in $\Sigma_{\mathcal{F}}^1$. The semantics of Fr is fixed and the fact $\mathsf{Fr}(n)$ denotes that the fresh name n is freshly generated. The semantics of the remaining fact symbols are defined by the multiset rewriting rules that generate and consume them. In our examples, we prefix all persistent protocol fact symbols with !.

Rules

We use labeled multiset rewriting to define the possible transitions of our transition system. A *labeled multiset rewriting rule* is a triple (l, a, r) with $l, a, r \in \mathcal{F}^*$, denoted $l \dashv\![a]\!\rightarrowtail r$. We often use inference rule notation for rules, i.e., we use

$$\frac{l_1 \quad \ldots \quad l_k}{r_1 \quad \ldots \quad r_n}[a_1 \quad \ldots \quad a_m] \quad \text{or} \quad \frac{l_1 \quad \ldots \quad l_k}{r_1 \quad \ldots \quad r_n} \quad \text{if } a \text{ is empty.}$$

For $ru = l \dashv\![a]\!\rightarrowtail r$, we define the *premises* as $prems(ru) = l$, the *actions* as $acts(ru) = a$, and the *conclusions* as $concs(ru) = r$. We call a set of labeled multiset rewriting rules a *labeled multiset rewriting system*. In the following, we often drop the "labeled" qualifier.

To deal with fresh name generation, we define the rule

$$\textsc{Fresh} = [] \dashv\![]\!\rightarrowtail [\mathsf{Fr}(x{:}fr)],$$

which is the only rule that produces Fr facts. In the operational semantics, we ensure that applications of this rule are unique and that the same fresh name is never generated twice.

Labeled Operational Semantics

To define the labeled operational semantics of a multiset rewrite system R, we first define the labeled transition relation $steps(R) \subseteq \mathcal{G}^\sharp \times (ginsts_{\mathcal{DH}_e}(R \cup \{\textsc{Fresh}\})) \times \mathcal{G}^\sharp$ as

$$\frac{\begin{array}{c} l \dashv\![a]\!\rightarrowtail r \in ginsts_{\mathcal{DH}_e}(R \cup \{\textsc{Fresh}\}) \quad S' = (S \setminus^\sharp lfacts(l)) \cup^\sharp mset(r) \\ lfacts(l) \subseteq^\sharp S \quad set(pfacts(l)) \subseteq set(S) \end{array}}{(S, l \dashv\![a]\!\rightarrowtail r, S') \in steps(R)},$$

where *lfacts*(l) and *pfacts*(l) denote the multisets of linear and persistent facts in l, respectively, and each transition is labeled with the applied rule instance.

A transition rewrites the state S with a ground instance of Fresh or a rule from R. Since we perform multiset rewriting modulo $\mathcal{DH}_e$, any applicable ground $\mathcal{DH}_e$-instance of a rule in $R \cup \{\textsc{Fresh}\}$ can be used. An instance $l \text{-}[a]\!\!\rightarrow r$ is applicable to S if the multiset of linear facts in l is included in S with respect to multiset inclusion ($\subseteq^\sharp$) and the set of persistent facts in l is included in S with respect to set inclusion. To obtain the successor state S', the consumed linear facts are removed and the generated facts are added.

Definition 3.3. An *execution* of R is an alternating sequence

$$e = [S_0, (l_1 \text{-}[a_1]\!\!\rightarrow r_1), S_1, \ldots, S_{k-1}, (l_k \text{-}[a_k]\!\!\rightarrow r_k), S_k]$$

of states and multiset rewriting rule instances such that the following conditions hold:

E1. $S_0 = \emptyset^\sharp$

E2. For all $i \in \{1, \ldots, k\}$, $(S_{i-1}, (l_i \text{-}[a_i]\!\!\rightarrow r_i), S_i) \in \mathit{steps}(R)$.

E3. For all $i, j \in \{1, \ldots, k\}$ and $n \in \mathsf{FN}$ where $(l_i \text{-}[a_i]\!\!\rightarrow r_i) = (l_j \text{-}[a_j]\!\!\rightarrow r_j) = ([\,] \text{-}[\,]\!\!\rightarrow \mathsf{Fr}(n))$, it holds that $i = j$.

We denote the set of executions of R with *execs*(R). We define the *trace* of such an execution e as $\mathit{trace}(e) = [\mathit{set}(a_1), \ldots, \mathit{set}(a_k)]$, i.e., the trace is the sequence of sets of actions of the multiset rewriting rule instances. We define the *observable trace* $\overline{tr}$ of a trace tr as the subsequence of all actions in tr that are not equal to $\emptyset$.

The Conditions **E1** and **E2** ensure that an execution starts with the empty multiset and each step is valid. The Condition **E3** ensures that the same fresh name is never generated twice. To ensure that fresh name generation works as intended, we will restrict ourselves to multiset rewriting systems R where rules do not introduce Fr-facts or create fresh names themselves. To formally define what it means to "create a fresh name", we define the *stable fresh names* of a term t as

$$\mathit{fnames}_{st}(t) = \bigcap_{t =_{\mathcal{DH}_e} t'} (\mathit{St}(t') \cap \mathsf{FN})$$

The definition of *fnames*$_{st}$ accounts for the fact that a term can contain spurious fresh names if it is not reduced with respect to the cancellation equations in $\mathcal{DH}_e$. For example, the term $\mathsf{a}*\mathsf{b}*\mathsf{b}^{-1}$ contains the name b, but $\mathit{fnames}_{st}(\mathsf{a}*\mathsf{b}*\mathsf{b}^{-1}) = \{\mathsf{a}\}$ since b does not occur in all terms that are equal to $\mathsf{a}*\mathsf{b}*\mathsf{b}^{-1}$ modulo $\mathcal{DH}_e$. We say a multiset rewriting system R *does not create fresh names* if for all $(S, ri, S') \in \mathit{steps}(R)$, either ri is an instance of Fresh or $\mathit{fnames}_{st}(S') \subseteq \mathit{fnames}_{st}(S)$ considering states as terms.

3.1.3 Protocol and Adversary Specification

To define the protocol and message deduction rules, we assume that $\Sigma_{\mathcal{F}}$ consists of Fr, an arbitrary number of protocol-specific fact symbols to describe the protocol state, and the following special fact symbols. 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.

Protocol Rules

We now define our formal notion of a protocol which encompasses both the rules executed by the honest participants and the adversary's capabilities, like revealing long-term keys.

Definition 3.4. A *protocol rule* is a multiset rewriting rule $l \;\text{-}[a]\!\!\rightarrow r$ such that

P1. l, a, and r do not contain fresh names,

P2. l does not contain K and Out facts,

P3. r does not contain K, In, and Fr facts,

P4. the argument of an Fr-fact is always of sort *fr*,

P5. r does not contain the function symbol $*$, and

P6. $l \;\text{-}[a]\!\!\rightarrow r$ satisfies (a) $\mathit{vars}(r) \subseteq \mathit{vars}(l) \cup \mathcal{V}_{pub}$ and (b) l only contains irreducible function symbols from $\Sigma_{\mathcal{ST}}$ or it is an instance of a rule that satisfies (a) and (b).

A *protocol* is a finite set of protocol rules.

Condition **P1** prevents protocol rules from directly using fresh names. Conditions **P2** and **P3** prevent protocol rules from using the special facts in unintended ways. Condition **P4** simplifies later definitions and is not a restriction since Fr-facts in reachable states always have a fresh name as argument. Condition **P5** ensures that multiplication is not directly used and simplifies reasoning about products since protocol rules introduce new products only as exponents. Condition **P6** is a syntactic criterion ensuring together with **P1** that protocols do not create fresh names, i.e., all fresh names originate from instances of the FRESH rule. Note that condition **P5** is similar to those of previous work such as [108] and [45] and is not a restriction in practice. Protocols that use multiplication in the group of exponents such as MTI/C0 [119] can usually be specified by using repeated exponentiation. Moreover, protocols that use multiplication in the DH group, such as MQV [113], cannot be specified anyway since $*$ denotes multiplication in the group of exponents.

Example 3.5. There are several kinds of rules that are forbidden by **P1**–**P6**. For example, $[\mathsf{In}(x)] \;\text{-}[]\!\!\rightarrow [\mathsf{Out}(y)]$ is forbidden since y is neither of sort *pub* nor does it occur in the premises. We do not allow this rule since it can send arbitrary terms. This is problematic since those terms can contain fresh names that are identical to fresh names chosen later on. The rule $[\mathsf{In}(\mathit{fst}(\langle z, y\rangle))] \;\text{-}[]\!\!\rightarrow [\mathsf{Out}(y)]$ is equivalent to this rule and demonstrates the problem with allowing reducible function symbols in the premises.

Another forbidden rule is $[\mathsf{In}(x), \mathsf{Fr}(y{:}\mathit{fr})] \;\text{-}[]\!\!\rightarrow [\mathsf{Out}(x*(y{:}\mathit{fr}))]$ which contains $*$. We do not allow this rules since it violates **P5**. The rule $[\mathsf{In}(x), \mathsf{Fr}(y{:}\mathit{fr})] \;\text{-}[]\!\!\rightarrow [\mathsf{Out}(\mathsf{g}\hat{\ }(x*(y{:}\mathit{fr})))]$ is also forbidden, but the equivalent rule $[\mathsf{In}(x), \mathsf{Fr}(y{:}\mathit{fr})] \;\text{-}[]\!\!\rightarrow [\mathsf{Out}((\mathsf{g}\hat{\ }x)\hat{\ }(y{:}\mathit{fr}))]$ is allowed. Finally, the rule $[\mathsf{In}(\mathsf{g}\hat{\ }(y{:}\mathit{fr})), \mathsf{St}(y{:}\mathit{fr}, x)] \;\text{-}[]\!\!\rightarrow [\mathsf{Out}((\mathsf{g}\hat{\ }(y{:}\mathit{fr}))\hat{\ }x)]$ is allowed since it is an instance of $[\mathsf{In}(z), \mathsf{St}(y{:}\mathit{fr}, x)] \;\text{-}[]\!\!\rightarrow [\mathsf{Out}(z\hat{\ }x)]$.

Message Deduction Rules

The set of messages deduction rules is defined as

$$MD = \left\{ \begin{array}{llll} \dfrac{\mathsf{Out}(x)}{\mathsf{K}(x)} & \dfrac{\mathsf{K}(x)}{\mathsf{In}(x)}[\mathsf{K}(x)] & \dfrac{\mathsf{Fr}(x{:}\mathit{fr})}{\mathsf{K}(x{:}\mathit{fr})} & \dfrac{}{\mathsf{K}(x{:}\mathit{pub})} \\ \dfrac{K(x_1) \quad \ldots \quad \mathsf{K}(x_k)}{K(f(x_1,\ldots x_k))} & \multicolumn{3}{l}{\text{for all } f \in \Sigma^k_{\mathcal{DH}_e}} \end{array} \right\}.$$

The rules in the first line 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 rules in the second line allow the adversary to learn public names and freshly generated names. The remaining rules allow the adversary to apply functions from $\Sigma_{\mathcal{DH}_e}$ to known messages. Note that MD does not create fresh names.

Example 3.6. Consider the protocol

$$P_{Msg} = \left\{ \frac{\mathsf{Fr}(x{:}fr) \quad \mathsf{Fr}(y{:}fr)}{\mathsf{St}(x) \quad \mathsf{Out}(\langle (\mathsf{g}\hat{\ }x)\hat{\ }y, y^{-1}\rangle)}[\mathsf{Start}()], \quad \frac{\mathsf{St}(x) \quad \mathsf{In}(\mathsf{g}\hat{\ }x)}{}[\mathsf{Fin}()] \right\}.$$

which encodes the message deducibility problem "is $\mathsf{g}\hat{\ }x$ deducible from $\langle (\mathsf{g}\hat{\ }x)\hat{\ }y, y^{-1}\rangle$ for fresh names x and y". This problem has a solution if and only if the protocol has the trace $[\{\mathsf{Start}()\}, \{\mathsf{K}(m)\}, \{\mathsf{Fin}()\}]$ for some message m. In this and the following examples, we never use the same variable name for variables of different sorts in a rule and therefore include the sort annotation only for the first occurrence of a variable.

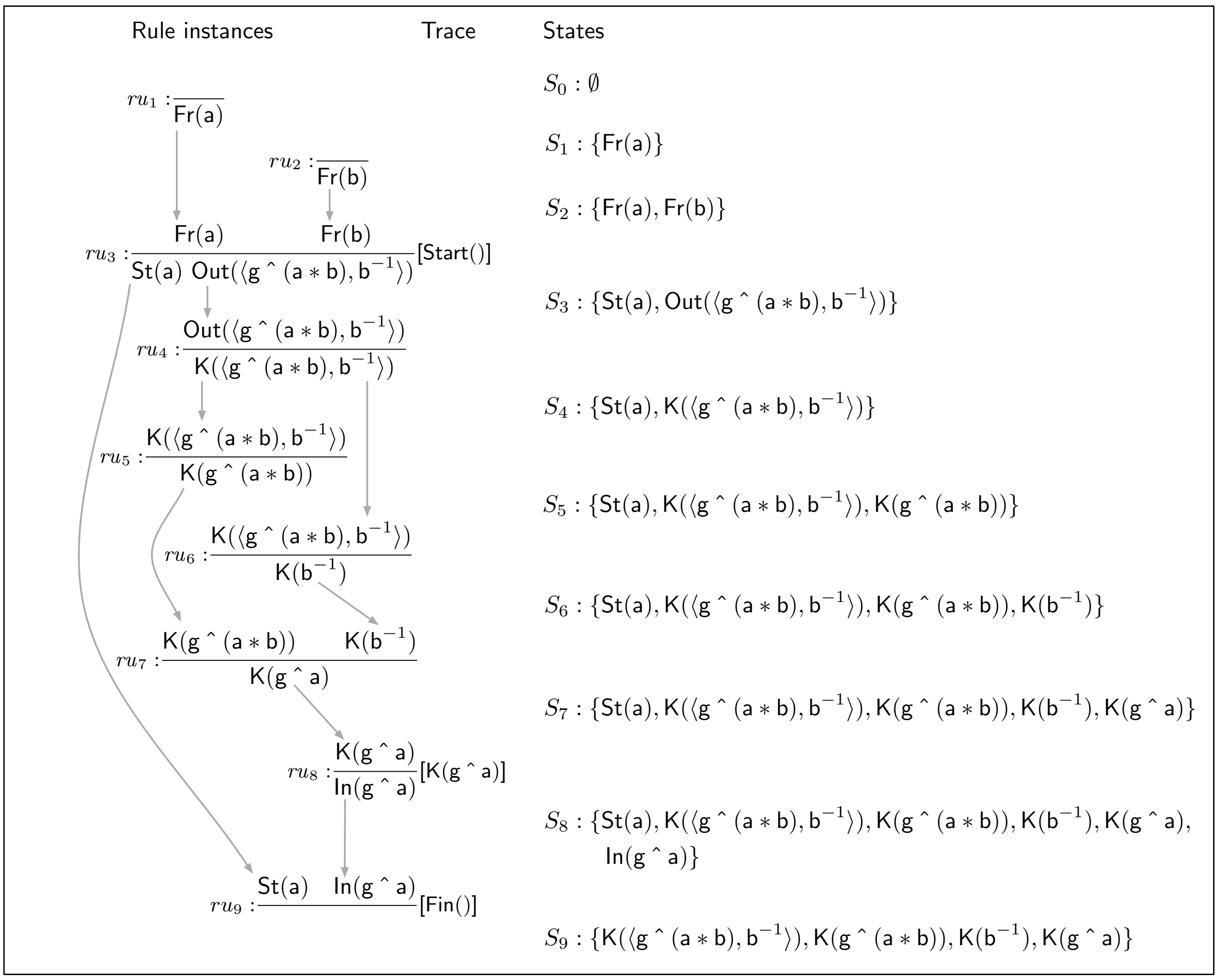
The protocol has the execution shown in Figure 3.1 with the observable trace $[\{\mathsf{Start}()\}, \{\mathsf{K}(\mathsf{g}\hat{\ }\mathsf{a})\}, \{\mathsf{Fin}()\}]$. We show the states on the right and the $\mathcal{DH}_e$-instances of the rules used to rewrite the state on the left. The first rule applications add fresh facts which are consumed by the first protocol rule instance ru_3. This is the first observable step. Rule instance ru_4 models the reception of the message sent by the protocol and ru_5, ru_6, and ru_7 denote message deduction steps. Here, we chose those $\mathcal{DH}_e$-instances of the rules that simplify the terms in the conclusion. For instance, ru_5 is a $\mathcal{DH}_e$-instance of $\mathsf{K}(x) \xrightarrow{[]} \mathsf{K}(fst(x))$ and ru_7 is a $\mathcal{DH}_e$-instance of $[\mathsf{K}(x_1), \mathsf{K}(x_2)] \xrightarrow{[]} \mathsf{K}(x_1\hat{\ }x_2)$. The last two rule instances model the sending of the deduced message $\mathsf{g}\hat{\ }\mathsf{a}$ by the adversary and the execution of the second protocol step.

Note that the sequence of rule instances is already sufficient to characterize the execution since the trace and the sequence of states can be reconstructed from it. To denote the causal dependencies between transitions, the figure contains grey arrows from conclusions to premises of rule instances. Such an arrow denotes that the corresponding fact is generated by the conclusion that is the source of the arrow and consumed by the premise that is the target of the arrow. The causal dependencies allow us to find permutations of the sequence of rule instances that correspond to valid executions of P_{Msg}. More precisely, any permutation of the rule instances that respects the causal dependencies corresponds to a valid execution of P_{Msg}. We can therefore represent a *set* of executions by a sequence of rule instances and their causal dependencies. We take advantage of these observations in the next section when we define dependency graphs as sequences of rule instances together with their causal dependencies.

3.1.4 Trace Formulas

We use many-sorted first-order logic with a sort for timepoints [47] 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. In the following, we use i and j for temporal variables, f for facts, and t_1, t_2 for terms.

A *trace atom* is either a *term equality* $t_1 \approx t_2$, a *timepoint ordering* $i \prec j$, a *timepoint equality* $i \approx j$, or an *action* $f @ i$. A *trace formula* is a first-order formula over trace atoms. In the following, we also use a fact f as a formula to abbreviate the formula $\exists i. f @ i$.



Rule instances
Trace
States
$ru_1 : \frac{}{\mathsf{Fr(a)}}$
$ru_2 : \frac{}{\mathsf{Fr(b)}}$
$ru_3 : \frac{\mathsf{Fr(a)} \quad \mathsf{Fr(b)}}{\mathsf{St(a)} \quad \mathsf{Out}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle)}$ [Start()]
$ru_4 : \frac{\mathsf{Out}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle)}{\mathsf{K}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle)}$
$ru_5 : \frac{\mathsf{K}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle)}{\mathsf{K}(\mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}))}$
$ru_6 : \frac{\mathsf{K}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle)}{\mathsf{K}(\mathsf{b}^{-1})}$
$ru_7 : \frac{\mathsf{K}(\mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b})) \quad \mathsf{K}(\mathsf{b}^{-1})}{\mathsf{K}(\mathsf{g} \,\hat{}\, \mathsf{a})}$
$ru_8 : \frac{\mathsf{K}(\mathsf{g} \,\hat{}\, \mathsf{a})}{\mathsf{In}(\mathsf{g} \,\hat{}\, \mathsf{a})}$ [K(g ^ a)]
$ru_9 : \frac{\mathsf{St(a)} \quad \mathsf{In}(\mathsf{g} \,\hat{}\, \mathsf{a})}{}$ [Fin()]
$S_0 : \emptyset$
$S_1 : \{\mathsf{Fr(a)}\}$
$S_2 : \{\mathsf{Fr(a)}, \mathsf{Fr(b)}\}$
$S_3 : \{\mathsf{St(a)}, \mathsf{Out}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle)\}$
$S_4 : \{\mathsf{St(a)}, \mathsf{K}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle)\}$
$S_5 : \{\mathsf{St(a)}, \mathsf{K}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle), \mathsf{K}(\mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}))\}$
$S_6 : \{\mathsf{St(a)}, \mathsf{K}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle), \mathsf{K}(\mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b})), \mathsf{K}(\mathsf{b}^{-1})\}$
$S_7 : \{\mathsf{St(a)}, \mathsf{K}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle), \mathsf{K}(\mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b})), \mathsf{K}(\mathsf{b}^{-1}), \mathsf{K}(\mathsf{g} \,\hat{}\, \mathsf{a})\}$
$S_8 : \{\mathsf{St(a)}, \mathsf{K}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle), \mathsf{K}(\mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b})), \mathsf{K}(\mathsf{b}^{-1}), \mathsf{K}(\mathsf{g} \,\hat{}\, \mathsf{a}), \mathsf{In}(\mathsf{g} \,\hat{}\, \mathsf{a})\}$
$S_9 : \{\mathsf{K}(\langle \mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b}), \mathsf{b}^{-1} \rangle), \mathsf{K}(\mathsf{g} \,\hat{}\, (\mathsf{a} * \mathsf{b})), \mathsf{K}(\mathsf{b}^{-1}), \mathsf{K}(\mathsf{g} \,\hat{}\, \mathsf{a})\}$

Figure 3.1. Execution of the protocol P_{Msg}. The grey arrows denote causal dependencies.

To define the semantics of trace formulas, we associate a domain DOM_s with each sort s. The domain for timepoints is $DOM_{temp} = \mathbb{Q}$ and the domains for messages are $DOM_{msg} = \mathcal{M}$, $DOM_{fr} = \mathsf{FN}$, and $DOM_{pub} = \mathsf{PN}$. We say a function θ from $\mathcal{V}$ to $\mathbb{Q} \cup \mathcal{M}$ is a *valuation* if it respects sorts, i.e., $\theta(\mathcal{V}_s) \subseteq DOM_s$ 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 $\mathcal{E}$, the *satisfaction relation* $(tr, \theta) \vDash_{\mathcal{E}} \varphi$ between traces tr, valuations θ, and trace formulas φ is defined inductively by the following rules:

$$
\begin{array}{ll}
(tr,\theta) \vDash_{\mathcal{E}} f@i & \text{if } \theta(i) \in idx(tr) \text{ and } f\theta \in_{\mathcal{E}} tr_{\theta(i)} \\
(tr,\theta) \vDash_{\mathcal{E}} i \prec j & \text{if } \theta(i) < \theta(j) \\
(tr,\theta) \vDash_{\mathcal{E}} i \approx j & \text{if } \theta(i) = \theta(j) \text{ and } i, j \text{ of sort } temp \\
(tr,\theta) \vDash_{\mathcal{E}} t_1 \approx t_2 & \text{if } t_1\theta =_{\mathcal{E}} t_2\theta \text{ and } t_1, t_2 \text{ of sort } msg \\
(tr,\theta) \vDash_{\mathcal{E}} \neg\varphi & \text{if not } (tr,\theta) \vDash_{\mathcal{E}} \varphi \\
(tr,\theta) \vDash_{\mathcal{E}} \varphi \wedge \psi & \text{if } (tr,\theta) \vDash_{\mathcal{E}} \varphi \text{ and } (tr,\theta) \vDash_{\mathcal{E}} \psi \\
(tr,\theta) \vDash_{\mathcal{E}} \exists x{:}s.\, \varphi & \text{if there is } u \in DOM_s \text{ such that } (tr, \theta[x \mapsto u]) \vDash_{\mathcal{E}} \varphi
\end{array}
$$

The semantics of the remaining logical connectives and quantifiers are defined by translation to the given fragment as usual. For closed formulas φ, we overload notation as follows. We write $tr \vDash_{\mathcal{E}} \varphi$ if $(tr, \theta) \vDash_{\mathcal{E}} \varphi$ for some θ. We call such a trace tr a model of φ. For a set of traces *Tr*, we write $Tr \vDash_{\mathcal{E}} \varphi$ if $tr \vDash_{\mathcal{E}} \varphi$ for all $tr \in Tr$. We say that φ is *P,$\mathcal{E}$-valid*, written $P \vDash_{\mathcal{E}} \varphi$, if $trace(execs(P \cup MD)) \vDash_{\mathcal{E}} \varphi$. We say that a formula φ is *P,$\mathcal{E}$-satisfiable* if there is $tr \in trace(execs(P \cup MD))$ such that $tr \vDash_{\mathcal{E}} \varphi$. Note that a closed formula φ is P,$\mathcal{E}$-valid if and only if $\neg\varphi$ is not P,$\mathcal{E}$-satisfiable.

We use the rationals instead of the natural numbers as the domain for temporal variables to achieve the following property for the satisfaction of formulas by traces.

Lemma 3.7. *For all trace formulas φ and all traces tr_1 and tr_2 with $\overline{tr_1} = \overline{tr_2}$, it holds that $tr_1 \vDash_{\mathcal{E}} \varphi$ if and only if $tr_2 \vDash_{\mathcal{E}} \varphi$.*

The lemma holds because $\mathbb{Q}$ is dense which makes it impossible to formalize the next-operator in our logic. Therefore, a formula cannot distinguish between traces that only differ in the number of silent steps between two observable steps. We give a full proof of this Lemma in [157].

3.1.5 Formalization of the one-pass UM Protocol

As a running example, we use the one-pass UM protocol and an adversary model with long-term and ephemeral key reveals, but no session key reveals. In Section 3.5.2, we will present a formalization of NAXOS with respect to the full eCK model. The rules for the protocol are given in Figure 3.2. The protocol can be obtained from the two-pass *UM* protocol given in Figure 2.4 by leaving out the message from the responder to the initiator and using the responder's long-term public key as his ephemeral public key.

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

Initiator:
$$\frac{\mathsf{Fr}(x{:}fr) \quad !\mathsf{Ltk}(A{:}pub, a{:}fr) \quad !\mathsf{Pk}(B{:}pub, \bar{B})}{\mathsf{Out}(X) \quad !\mathsf{Ephk}(sid, x)}[\,\mathsf{Accept}(sid, key)\,]$$

where $\bar{B} = \mathsf{g}\hat{\ }(b{:}fr)$
$X = \mathsf{g}\hat{\ }x$
$sid = \langle A, B, X, \mathcal{I}\rangle$
$key = h(\bar{B}\hat{\ }x, \bar{B}\hat{\ }a, A, B, X)$

Responder:
$$\frac{\mathsf{In}(X) \quad !\mathsf{Pk}(A{:}pub, \bar{A}) \quad !\mathsf{Ltk}(B{:}pub, b{:}fr)}{}[\,\mathsf{Accept}(sid, key)\,]$$

where $\bar{A} = \mathsf{g}\hat{\ }(a{:}fr)$
$sid = \langle B, A, X, \mathcal{R}\rangle$
$key = h(X\hat{\ }b, \bar{A}\hat{\ }b, A, B, X)$

Long-Term reveal:
$$\frac{!\mathsf{Ltk}(A, a)}{\mathsf{Out}(a)}[\,\mathsf{RevealLtk}(A)\,]$$

Ephemeral reveal:
$$\frac{!\mathsf{Ephk}(sid, x)}{\mathsf{Out}(x)}[\,\mathsf{RevealEphk}(sid)\,]$$

Figure 3.2. Rules P_{UM} defining the one-pass *UM* protocol.

To model the protocol, we use the subterm-convergent theory $\mathcal{DY}$ as $\mathcal{ST}$. We use the public name g to denote the generator of the DH group and the public names $\mathcal{I}$ and $\mathcal{R}$ to denote the two roles in the session identifier.

The first rule models the generation of keys. A fresh name a is chosen and used as the long-term private key of an agent A. The corresponding public long-term key is $\mathsf{g}\hat{}a$. The rule generates !Ltk and !Pk facts that can be used to look up the private and the public key of an agent. Additionally, the public key is sent on the network and can be received by the adversary. We do not model key registration by the adversary in this example; all keys are honestly generated. Note that an agent can have more than one key pair. The second rule models the initiator's actions. He generates a fresh name x that is used as his ephemeral private key, he looks up his own long-term private key a, and B's public key $\bar{B}$. Then he computes his public ephemeral key X and sends it. He also provides his ephemeral key for reveals. Finally, he accepts the session key *key* with the session id *sid*. Note that the pattern matching on the !Ltk and !Pk facts exploits that if such a fact is included in the state, then the arguments always have a certain form. For example, the second argument of !Ltk is always of sort *fr*. The third rule models the responder's actions. The responder receives a message X, looks up the required keys, and computes the session key. The last two rules model key reveals by the adversary. These reveals can be executed whenever the corresponding long-term or ephemeral private key exists, but they are observable. The security property can therefore restrict reveals for certain agents and sessions.

$$
\begin{aligned}
&\varphi_{\mathit{UM\text{-}exec}} = \\
&\quad \exists A\ B\ X\ \mathit{key}. \\
&\qquad \text{// An initiator session and a matching responder session accept } \mathit{key}, \\
&\qquad \mathsf{Accept}(\langle A, B, X, \mathcal{I}\rangle, \mathit{key}) \wedge \mathsf{Accept}(\langle B, A, X, \mathcal{R}\rangle, \mathit{key}) \wedge \\
&\qquad \text{// the initiator accepts before the responder, and} \\
&\qquad i \prec j \wedge \\
&\qquad \text{// the adversary did not perform any reveals.} \\
&\qquad \neg(\exists C. \mathsf{RevealLtk}(C)) \wedge \neg(\exists\, \mathit{sid}. \mathsf{RevealEphk}(\mathit{sid}))
\end{aligned}
$$

$$
\begin{aligned}
&\varphi_{\mathit{UM\text{-}sec}} = \\
&\quad \forall A B X\, \mathit{key}\, \mathit{msid}. \\
&\qquad \text{// If the key of a responder test session with matching session } \mathit{msid} \text{ is known,} \\
&\qquad \mathsf{Accept}(\langle B, A, X, \mathcal{R}\rangle, \mathit{key}) \wedge \mathsf{K}(\mathit{key}) \wedge \mathit{msid} \approx \langle A, B, X, \mathcal{I}\rangle \\
&\qquad \text{// then the session is not clean, i.e., one of the following happened:} \\
&\qquad \Longrightarrow (\text{// 1. } B\text{'s long-term key was revealed.} \\
&\qquad\qquad \mathsf{RevealLtk}(B) \\
&\qquad\quad \text{// 2. There is a long-term key reveal for the peer of the test session} \\
&\qquad\quad \text{// } \quad \text{and an ephemeral key reveal for a matching session.} \\
&\qquad\quad \vee (\mathsf{RevealLtk}(A) \wedge \mathsf{RevealEphk}(\mathit{msid})) \\
&\qquad\quad \text{// 3. There is no matching session and a long-term key reveal for the} \\
&\qquad\quad \text{// } \quad \text{peer of the responder session.} \\
&\qquad\quad \vee (\neg(\exists \mathit{key}'. \mathsf{Accept}(\mathit{msid}, \mathit{key}')) \wedge \mathsf{RevealLtk}(A)))
\end{aligned}
$$

Figure 3.3. Properties for the P_{UM} protocol.

Figure 3.3 shows two properties for our formalization of the UM one-pass protocol. The first property characterizes traces where two matching sessions compute the same key and the adversary does not perform any reveals. This property is satisfiable for the protocol P_{UM}, i.e., there are traces of P_{UM} that satisfy this property and hence key agreement for matching sessions is reachable. We will later show how our constraint solving algorithm can prove the satisfiability of this property. The second property formalizes the desired security property for the responder. Whenever a responder session accepts a key that is known to the adversary, then the session is not clean, i.e., the adversary performed a forbidden key reveal. We will later show how our constraint solving algorithm can prove the validity of this property for P_{UM}, i.e., that all traces of P_{UM} satisfy this property.

3.2 Verification Theory

For symbolic attack-search algorithms, there are several drawbacks to the multiset rewriting semantics given in the previous section. First, incrementally constructing attacks is difficult with executions, as they do not contain causal dependencies between steps. Second, symbolic reasoning modulo $\mathcal{DH}_e$ is difficult because $\mathcal{DH}_e$ contains cancellation equations. For example, if the adversary knows $t = na{*}x$ for a fresh name na, we cannot conclude that na has been used in the construction of t, as x could be equal to na^{-1}. Third, the message deduction rules allow for redundant steps such as first encrypting a cleartext and then decrypting the resulting ciphertext. For search algorithms, it is useful to impose normal-form conditions on message deduction to avoid exploring such redundant steps.

We take the following approach. First, we define dependency graphs. They consist of the sequence of rewriting rule instances corresponding to a protocol execution and their causal dependencies, similar to strand spaces [83]. Afterwards, we show that we can use dependency graphs modulo AC, an equational theory without cancellation equations, instead of dependency graphs modulo $\mathcal{DH}_e$. Next, we define normal message deductions and the corresponding normal dependency graphs. We also show that normal dependency graphs are weakly trace equivalent to the multiset rewriting semantics. Finally, we define the fragment of guarded trace properties, which ensures that variables in formulas refer to terms in the considered traces.

3.2.1 Dependency Graphs

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

Example 3.8. The dependency graph for the execution of protocol P_{Msg} from Figure 3.1 is shown in Figure 3.4. 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. Persistent conclusions can be consumed multiple times and can therefore have multiple outgoing edges. For example, the conclusion of node 4 is consumed by nodes 5 and 6. Since this is a dependency graph modulo $\mathcal{DH}_e$, the nodes are labeled with ground $\mathcal{DH}_e$-instances of rules from $P_{Msg} \cup MD \cup \{\textsc{Fresh}\}$.

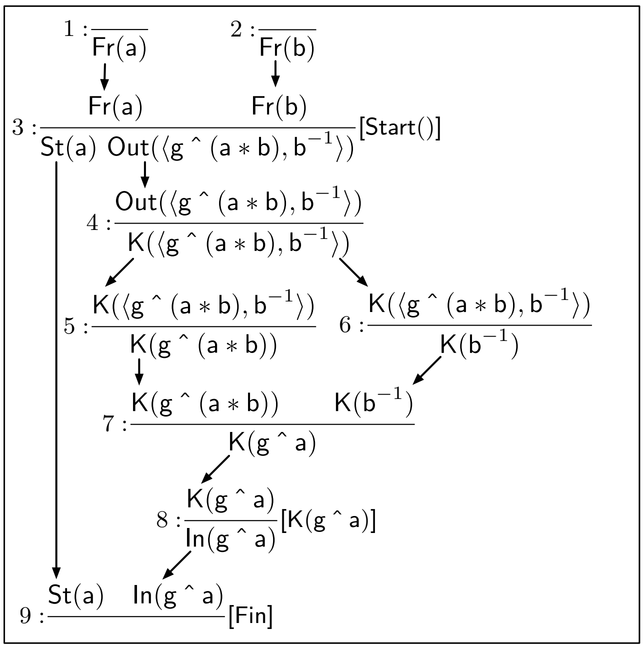

1 : Fr(a)
2 : Fr(b)
3 : Fr(a) Fr(b) / St(a) Out(⟨g ^ (a ∗ b), b⁻¹⟩) [Start()]
4 : Out(⟨g ^ (a ∗ b), b⁻¹⟩) / K(⟨g ^ (a ∗ b), b⁻¹⟩)
5 : K(⟨g ^ (a ∗ b), b⁻¹⟩) / K(g ^ (a ∗ b))
6 : K(⟨g ^ (a ∗ b), b⁻¹⟩) / K(b⁻¹)
7 : K(g ^ (a ∗ b)) K(b⁻¹) / K(g ^ a)
8 : K(g ^ a) / In(g ^ a) [K(g ^ a)]
9 : St(a) In(g ^ a) / [Fin]

Figure 3.4. Dependency graph for the execution from Example 3.6.

Definition 3.9. Let $\mathcal{E}$ be an equational theory and R be a set of labeled multiset rewriting rules. We say that the pair $dg = (I, D)$ is a *dependency graph modulo* $\mathcal{E}$ *for* R if $I \in ginsts_{\mathcal{E}}(R \cup \{\textsc{Fresh}\})^*$, $D \in \mathbb{N}^2 \times \mathbb{N}^2$, and dg satisfies the Conditions **DG1–4** listed below. To state these conditions, we introduce the following definitions. We call $idx(I)$ the *nodes* of dg and D the *edges* of dg. We write $(i,u) \rightarrowtail (j,v)$ for the edge $((i,u),(j,v))$. A *conclusion* of dg is a pair $(i,\, u)$ such that i is a node of dg and $u \in idx(concs(I_i))$. The corresponding *conclusion fact* is $(concs(I_i))_u$. A *premise* of dg is a pair (i,u) such that i is a node of dg and $u \in idx(prems(I_i))$. The corresponding *premise fact* is $(prems(I_i))_u$. A conclusion or premise is *linear* if its fact is linear.

DG1. For every edge $(i,u) \rightarrowtail (j,v) \in D$, it holds that $i < j$ and the conclusion fact of (i,u) is syntactically equal 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 $\textsc{Fresh}$ instances are unique.

We denote the set of all dependency graphs modulo $\mathcal{E}$ for R by $dgraphs_{\mathcal{E}}(R)$.

Let $dg = (I, D)$ and $I = [l_1 \text{-}[a_1]\!\rightarrow r_1, \ldots, l_k \text{-}[a_k]\!\rightarrow r_k]$, overloading notation, we define the *trace* of dg as $trace(dg) = [set(a_1), \ldots, set(a_k)]$. We can show that the multiset rewriting semantics given in Section 3.1 and the dependency graphs are trace equivalent in the following sense

Lemma 3.10. *For all protocols P,* $trace(execs(P \cup MD)) = trace(dgraphs_{\mathcal{DH}_e}(P \cup MD))$.

Proof. We prove by induction that the sequences of rule instances of executions and dependency graphs coincide. Since the traces are determined by the sequences of rule instances, this completes the proof. □

3.2.2 Dependency Graphs Modulo AC

We now switch to a semantics based on dependency graphs modulo associativity and commutativity. To achieve this, we define the set of equations AC as

$$AC = \{x{*}(y{*}z) \simeq (x{*}y){*}z, x{*}y \simeq y{*}x\}$$

and the rewriting system $\mathcal{RDH}$ as

$$\mathcal{RDH} = \left\{ \begin{array}{lll} (x\hat{\ }y)\hat{\ }z \to x\hat{\ }(y{*}z) & x\hat{\ }1 \to x & \\ & & \\ (x^{-1}{*}y)^{-1} \to x{*}y^{-1} & 1^{-1} \to 1 & x^{-1}{*}y^{-1} \to (x{*}y)^{-1} \\ x{*}(x{*}y)^{-1} \to y^{-1} & x{*}1 \to x & (x^{-1})^{-1} \to x \\ x^{-1}{*}(y^{-1}{*}z) \to (x{*}y)^{-1}{*}z & x{*}(x^{-1}{*}y) \to y & x{*}x^{-1} \to 1 \\ (x{*}y)^{-1}{*}(y{*}z) \to x^{-1}{*}z & & \end{array} \right\}.$$

We define $\mathcal{RDH}_e = \mathcal{RDH} \cup \mathcal{R}_{\mathcal{ST}}$. Then $(\Sigma_{\mathcal{DH}_e}, \mathcal{RDH}_e, AC)$ is a decomposition of $\mathcal{DH}_e$ with the finite variant property for the following reasons. First, $(\Sigma_{\mathcal{DH}_e}, \mathcal{RDH}_e^{\simeq} \cup AC)$ is an equational presentation of $=_{\mathcal{DH}_e}$. Second, AC is regular, sort-preserving, and all variables are of sort *msg*. Third, $\mathcal{RDH}_e$ is sort-decreasing and $\mathcal{RDH}_e$,AC-rewriting is convergent and coherent. Since $\mathcal{RDH}$ and $\mathcal{ST}$ have disjoint signatures and neither contains duplicating rules [35], it suffices to consider both rewrite systems individually. Since $\mathcal{ST}$ is a subterm-convergent theory, both properties easily follow. For $\mathcal{RDH}$,AC-rewriting, we have used the AProVE termination tool [85] and the Maude Church-Rosser and Coherence Checker [73] to verify both properties. Fourth, there is a complete and finitary AC-unification algorithm. Finally, the finite variant property can be established individually for $\mathcal{RDH}$,AC and $\mathcal{ST}$, which has been done in [51].

Given a term t, we therefore have a unique normal form with respect to $\mathcal{RDH}_e$,AC-rewriting, which we denote with $t{\downarrow_{\mathcal{RDH}_e}}$. We use folding variant narrowing to compute a complete set of $\mathcal{RDH}_e$,AC-variants, which we denote by $\lceil t \rceil^{\mathcal{RDH}_e}_{substs}$. We use $\lceil t \rceil^{\mathcal{RDH}_e}_{insts}$ to denote the set $\{(t\tau){\downarrow_{\mathcal{RDH}_e}} | \tau \in \lceil t \rceil^{\mathcal{RDH}_e}_{substs}\}$ of normalized instances corresponding to the variants. We say that t is ${\downarrow_{\mathcal{RDH}_e}}$-normal if $t =_{AC} t{\downarrow_{\mathcal{RDH}_e}}$. We say a dependency graph $dg = (I, D)$ is ${\downarrow_{\mathcal{RDH}_e}}$-normal if all rule instances in I are. It is straightforward to extend the notion of $\mathcal{RDH}_e$,AC-variants to multiset rewriting rules by considering rules as terms and the required new function symbols as free. We can then show that we can take the $\mathcal{RDH}_e$,AC-variants of the multiset rewriting rules and use dependency graphs modulo AC.

Lemma 3.11. *For all sets of multiset rewriting rules R,*

$$dgraphs_{\mathcal{DH}_e}(R){\downarrow_{\mathcal{RDH}_e}} =_{AC} \{dg \mid dg \in dgraphs_{AC}(\lceil R \rceil^{\mathcal{RDH}_e}_{insts}) \wedge dg\ {\downarrow_{\mathcal{RDH}_e}}\text{-}normal\}.$$

Proof. The lemma is a consequence of the fact that all ${\downarrow_{\mathcal{RDH}_e}}$-normal $\mathcal{DH}_e$-instances of rules in R are ${\downarrow_{\mathcal{RDH}_e}}$-normal AC-instances of rules in $\lceil R \rceil^{\mathcal{RDH}_e}_{insts}$. □

Example 3.12. The dependency graph from Figure 3.4 is already an element of $dgraphs_{AC}(\lceil P_{Msg} \cup MD \rceil^{\mathcal{RDH}_e}_{insts})$ and ${\downarrow_{\mathcal{RDH}_e}}$-normal. For example, the label of node 5 is ${\downarrow_{\mathcal{RDH}_e}}$-normal and an AC-instance of the second rule in the set

$$\lceil \mathsf{K}(x) \dashv[]\!\!\rightarrow \mathsf{K}(fst(x)) \rceil^{\mathcal{RDH}_e}_{insts} = \{\mathsf{K}(x) \dashv[]\!\!\rightarrow \mathsf{K}(fst(x)), \mathsf{K}(\langle y, z \rangle) \dashv[]\!\!\rightarrow \mathsf{K}(y)\}.$$

As another example, consider the label of node 7 which is $\downarrow_{\mathcal{RDH}_e}$-normal and an AC-instance of the third rule rule in the set

$$\lceil \mathsf{K}(x), \mathsf{K}(y) \;\text{-[]}\!\!\rightarrow \mathsf{K}(x\,\hat{}\,y) \rceil^{\mathcal{RDH}_e}_{insts} = \left\{ \begin{array}{l} \mathsf{K}(x), \mathsf{K}(y) \;\text{-[]}\!\!\rightarrow \mathsf{K}(x\,\hat{}\,y) \\ \mathsf{K}(u\,\hat{}\,v), \mathsf{K}(v^{-1}) \;\text{-[]}\!\!\rightarrow \mathsf{K}(u) \\ \mathsf{K}(u\,\hat{}\,(v{*}w)), \mathsf{K}(w^{-1}) \;\text{-[]}\!\!\rightarrow \mathsf{K}(u\,\hat{}\,v) \\ \ldots \quad \text{44 other variants} \quad \ldots \end{array} \right\}.$$

The rule for exponentiation has 47 $\mathcal{RDH}_e$,AC-variants that cover all $\downarrow_{\mathcal{RDH}_e}$-normal-forms of $x\,\hat{}\,y$ for instantiations of x and y with arbitrary terms. To compute the set of variants, we have used TAMARIN, which implements folding variant narrowing.

Note that the rewriting system resulting from orienting the non-AC equations in $E_{\mathcal{DH}}$ from left to right is not coherent for similar reasons as the rewrite system from Example 2.1. After completion, the equations containing the inverse operator are usually oriented such that the inverse operator is pushed inwards, e.g., $(x{*}y)^{-1} \rightarrow x^{-1}{*}y^{-1}$. The problem with this orientation is that there is no finite set of variants for the term x^{-1} since for all k, the normal form of $(\mathsf{a}_1{*}\ldots{*}\mathsf{a}_k)^{-1}$ is $\mathsf{a}_1^{-1}{*}\ldots{*}\mathsf{a}_k^{-1}$. In $\mathcal{RDH}$, we therefore use an orientation due to Lankford which solves this problem by pushing the inverse operator outwards.

3.2.3 Normal Message Deduction Rules

In dependency graphs modulo AC, we use K-facts and ground AC-instances of $\lceil MD \rceil^{\mathcal{RDH}_e}_{insts}$ to model message deduction by the adversary. In the following, we introduce new fact symbols and a new set of rules to model *normal message deduction*. The new rules are sound and complete with respect to the old rules and enforce restrictions that are similar to those for normal natural deduction proofs [146]. In the setting of message deduction, there are several works that take a similar approach such as [48], [165], and [52]. We first focus on the deduction of products, then we show the rules for communication and $\Sigma_{\mathcal{DH}}$, and finally, we show how to handle the rules for $\Sigma_{\mathcal{ST}}$.

Multiplication

Conditions **P1**–**P6** from Definition 3.4 ensure that protocol rules do not build new products in extractable positions. More precisely, we define the *non-inverse* factors of a term t as

$$nifactors(t) = \begin{cases} nifactors(a) \cup nifactors(b) & \text{if } t = a{*}b \\ nifactors(s) & \text{if } t = s^{-1} \\ \{t\} & \text{otherwise} \end{cases}$$

and prove in Lemma A.4 in the Appendix that for all products t that can be extracted from messages sent by the protocol, the adversary can already deduce $nifactors(t)$ before the message is sent. Intuitively, this means that all extractable products are simple variations of products constructed by the adversary himself. For example, the adversary can send $\mathsf{a}^{-1}{*}\mathsf{b}$ to a protocol that inverts received messages. The protocol replies with $\mathsf{a}{*}\mathsf{b}^{-1}$, which is a different product, but has the same non-inverse factors.

Rules for Communication and $\mathcal{DH}$

Let us first assume that $\Sigma_{\mathcal{ST}} = \emptyset$. Then we can partition the rules in $\lceil MD \rceil_{insts}^{\mathcal{RDH}_e}$ into five subsets: communication rules for sending and receiving messages, deconstruction rules that extract a subterm from an argument, exponentiation rules that modify an exponentiation by adding and removing (some, but not all) exponents, construction rules that apply a function symbol from $\Sigma_{\mathcal{DH}}$ to arguments, and multiplication rules consisting of all $\mathcal{RDH}_e$,AC-variants of the rule for multiplication. Note that the set of exponentiation rules consists of all $\mathcal{RDH}_e$,AC-variants of the rule $[\mathsf{K}(x), \mathsf{K}(y)] \dashv[]\!\!\rightarrow \mathsf{K}(x\,\hat{}\,y)$ that are neither deconstruction nor construction rules.

We want to enforce *normal message deductions*, which satisfy the following normal-form condition. All messages are deduced by extracting subterms from messages sent by the protocol, optionally modifying the exponent of extracted exponentiations, and finally applying function symbols to these messages. This allows us to search for message deductions by applying deconstruction and exponentiation rules to messages sent by the protocol top-down and applying construction rules to messages received by the protocol bottom-up until both meet in the middle.

To achieve this, we introduce three new fact symbols. $\mathsf{K}^{\Downarrow \mathsf{d}}(m)$ denotes that m has been extracted from a message sent by the protocol. $\mathsf{K}^{\Downarrow \mathsf{e}}(m)$ denotes that m has been deduced by modifying the exponent of an extracted message. We call a rule instance with conclusion $\mathsf{K}^{\Downarrow \mathsf{d}}(m)$ or $\mathsf{K}^{\Downarrow \mathsf{e}}(m)$ a *deconstruction of* m. Finally, $\mathsf{K}^{\Uparrow}(m)$ denotes that m has been deduced using a normal message deduction. We call a rule instance with conclusion $\mathsf{K}^{\Uparrow}(m)$ a *normal deduction of* m.

Figure 3.5 shows the *normal message deduction rules ND* that have been obtained by replacing each K-fact in the previously defined sets of rules with the appropriate new fact symbol. The RECV rule generates a $\mathsf{K}^{\Downarrow \mathsf{d}}$-fact, which captures that a message sent by the protocol can be extracted. The SEND rule allows the adversary to send a message that has a normal deduction to the protocol. The deconstruction rules allow to extract a subterm from a message that has been extracted itself. This extraction might require an additional message. For example, the second deconstruction rule requires the inverse of the exponent to extract the base of an exponentiation. For these additional premises, we use $\mathsf{K}^{\Uparrow}$-facts. The exponentiation rules allow us to modify the exponent of an extracted message and use a $\mathsf{K}^{\Downarrow \mathsf{e}}$-fact for the conclusion. The COERCE rule reflects that both extracted subterms themselves and extracted subterms with modified exponent have a normal deduction. The resulting $\mathsf{K}^{\Uparrow}$-fact of this rule can then be used by construction rules or SEND. The construction rules use $\mathsf{K}^{\Uparrow}$ for both premises and conclusions. There are construction rules for all function symbols, for fresh names, and for public names. Note that we replace all multiplication rules with n,l-ary construction rules for multiplication. This is possible because we can construct all deducible products from their non-inverse factors.

Example 3.13. Figure 3.6 shows three message deduction subgraphs for exponentiation. Subfigure (a) shows a message deduction where $\mathsf{a}\,\hat{}\,\mathsf{b}$ is first constructed from a and b, and immediately afterwards, a is extracted. This is not a normal deduction because, as Subfigure (b) shows, a deconstruction rule for exponentiation cannot be applied to $\mathsf{K}^{\Uparrow}(\mathsf{a}\,\hat{}\,\mathsf{b})$. Subfigure (c) shows that the *ND* rules also forbid repeated exponentiations. The deduction in (c) can be replaced by the single rule instance $\mathsf{K}^{\Downarrow \mathsf{d}}(\mathsf{a}\,\hat{}\,\mathsf{b}), \mathsf{K}^{\Uparrow}(\mathsf{b}^{-1} {*} \mathsf{d}) \dashv[]\!\!\rightarrow \mathsf{K}^{\Downarrow \mathsf{e}}(\mathsf{a}\,\hat{}\,\mathsf{d})$. Note that the deduction in (c) includes the term c which occurs neither in the conclusion of i_2

Communication rules: $\textsc{Recv}\,\dfrac{\mathsf{Out}(x)}{\mathsf{K}^{\Downarrow \mathsf{d}}(x)}$ $\quad\textsc{Send}\,\dfrac{\mathsf{K}^{\Uparrow}(x)}{\mathsf{In}(x)}$

Deconstruction rules:

$$\dfrac{\mathsf{K}^{\Downarrow \mathsf{d}}(x^{-1})}{\mathsf{K}^{\Downarrow \mathsf{d}}(x)} \quad \dfrac{\mathsf{K}^{\Downarrow \mathsf{d}}(x\hat{\ }y) \quad \mathsf{K}^{\Uparrow}(y^{-1})}{\mathsf{K}^{\Downarrow \mathsf{d}}(x)} \quad \dfrac{\mathsf{K}^{\Downarrow \mathsf{d}}(x\hat{\ }(y^{-1})) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Downarrow \mathsf{d}}(x)} \quad \dfrac{\mathsf{K}^{\Downarrow \mathsf{d}}(x\hat{\ }(y{*}z^{-1})) \quad \mathsf{K}^{\Uparrow}(y^{-1}{*}z)}{\mathsf{K}^{\Downarrow \mathsf{d}}(x)}$$

Exponentiation rules:

$$\dfrac{\mathsf{K}^{\Downarrow \mathsf{d}}(x\hat{\ }y) \quad \mathsf{K}^{\Uparrow}(z)}{\mathsf{K}^{\Downarrow \mathsf{e}}(x\hat{\ }(y{*}z))} \quad \dfrac{\mathsf{K}^{\Downarrow \mathsf{d}}(x\hat{\ }y) \quad \mathsf{K}^{\Uparrow}(y^{-1}{*}z)}{\mathsf{K}^{\Downarrow \mathsf{e}}(x\hat{\ }z)} \quad \dots \quad \dfrac{\mathsf{K}^{\Downarrow \mathsf{d}}(x\hat{\ }(y_1{*}y_2^{-1})) \quad \mathsf{K}^{\Uparrow}(z_1{*}z_2^{-1})}{\mathsf{K}^{\Downarrow \mathsf{e}}(x\hat{\ }(y_1{*}z_1{*}(y_2{*}z_2)^{-1}))}$$

Coerce rule: $\textsc{Coerce}\,\dfrac{\mathsf{K}^{\Downarrow y}(x)}{\mathsf{K}^{\Uparrow}(x)}$

Construction rules:

$$\dfrac{\mathsf{K}^{\Uparrow}(x)}{\mathsf{K}^{\Uparrow}(x^{-1})} \quad \dfrac{\mathsf{K}^{\Uparrow}(x) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Uparrow}(x\hat{\ }y)} \quad \dfrac{}{\mathsf{K}^{\Uparrow}(1)} \quad \dfrac{\mathsf{K}^{\Uparrow}(x_1) \quad \dots \quad \mathsf{K}^{\Uparrow}(x_n) \quad \mathsf{K}^{\Uparrow}(x_{n+1}) \quad \dots \quad \mathsf{K}^{\Uparrow}(x_l)}{\mathsf{K}^{\Uparrow}((x_1{*}\dots{*}x_n){*}(x_{n+1}{*}\dots x_l)^{-1})}$$

$$\dfrac{\mathsf{Fr}(x{:}fr)}{\mathsf{K}^{\Uparrow}(x{:}fr)} \quad \dfrac{}{\mathsf{K}^{\Uparrow}(x{:}pub)}$$

Figure 3.5. The normal message deduction rules *ND*. We use y in the Coerce to denote that either $\mathsf{K}^{\Downarrow \mathsf{d}}(x)$ or $\mathsf{K}^{\Downarrow \mathsf{e}}(x)$ can be used as premise. There are construction rules for multiplication for all n and l such that $n > 0$ and $l > 1$. There are 42 exponentiation rules computed from the 47 $\mathcal{RDH}_e,AC$-variants of the exponentiation rule in *MD*. The remaining 5 variants correspond to the construction rule for exponentiation, the 3 deconstruction rules for exponentiation, and the variant where the exponent is 1 for which no message deduction rule is required.

nor in the first premise of i_1. Also note that it is possible to extend the deduction in (c) by adding an unbounded number of additional steps that remove the previously added exponent and add a new exponent. Since our search algorithm exploits locality of message deductions [121], it is important to exclude this redundancy.

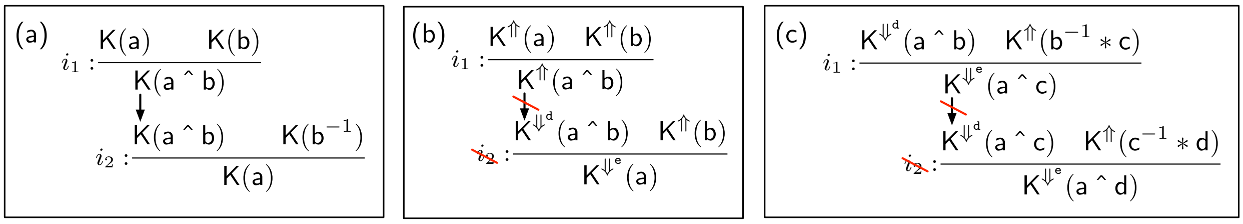

(a)
i_1 : K(a) K(b) / K(a ^ b)
i_2 : K(a ^ b) K(b^{-1}) / K(a)
(b)
i_1 : K$^{\Uparrow}$(a) K$^{\Uparrow}$(b) / K$^{\Uparrow}$(a ^ b)
i_2 : K$^{\Downarrow d}$(a ^ b) K$^{\Uparrow}$(b) / K$^{\Downarrow e}$(a)
(c)
i_1 : K$^{\Downarrow d}$(a ^ b) K$^{\Uparrow}$(b^{-1} * c) / K$^{\Downarrow e}$(a ^ c)
i_2 : K$^{\Downarrow d}$(a ^ c) K$^{\Uparrow}$(c^{-1} * d) / K$^{\Downarrow e}$(a ^ d)

Figure 3.6. Message deduction subgraphs for exponentiation. The crossed edges are not allowed since source and target are not equal.

Rules for $\mathcal{ST}$

To explain how the normal message deduction rules for $\Sigma_{\mathcal{ST}}$ are computed, we first show the rules for a simple theory. Then we explain the general case.

Example 3.14. Consider the subterm theory $\mathcal{DY}$ defined in Example 3.2. This theory has the construction rules

$$\frac{\mathsf{K}^{\Uparrow}(x) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Uparrow}(\langle x, y\rangle)} \qquad \frac{\mathsf{K}^{\Uparrow}(x)}{\mathsf{K}^{\Uparrow}(\mathit{fst}(x))} \qquad \frac{\mathsf{K}^{\Uparrow}(x)}{\mathsf{K}^{\Uparrow}(\mathit{snd}(x))} \qquad \frac{\mathsf{K}^{\Uparrow}(x)}{\mathsf{K}^{\Uparrow}(\mathit{pk}(x))} \qquad \frac{\mathsf{K}^{\Uparrow}(x) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Uparrow}(\mathit{enc}(x, y))}$$

$$\frac{\mathsf{K}^{\Uparrow}(x) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Uparrow}(\mathit{dec}(x, y))} \qquad \frac{}{\mathsf{K}^{\Uparrow}(\mathit{true})} \qquad \frac{\mathsf{K}^{\Uparrow}(x) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Uparrow}(\mathit{sig}(x, y))} \qquad \frac{\mathsf{K}^{\Uparrow}(x) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Uparrow}(\mathit{verify}(x, y))} \qquad \frac{\mathsf{K}^{\Uparrow}(x)}{\mathsf{K}^{\Uparrow}(h(x))}$$

and the deconstruction rules

$$\frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\langle x, y\rangle)}{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(x)} \qquad \frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\langle x, y\rangle)}{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(y)} \qquad \frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\mathit{enc}(x, \mathit{pk}(y))) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(x)}\,.$$

For this theory, the deconstruction rules can be computed by taking the non-trivial $\mathcal{RDH}_e$,AC-variants of the message deduction rules and replacing the K-facts such that the conclusion and the premise that contains the conclusion-message as a subterm use $\mathsf{K}^{\Downarrow^{\mathsf{d}}}$-facts and the remaining premises use $\mathsf{K}^{\Uparrow}$-facts. For example, the only non-trivial $\mathcal{RDH}_e$,AC-variant of the rule for *dec* is $[\mathsf{K}(\mathit{enc}(x,\, \mathit{pk}(y))), \mathsf{K}(y)] \,\text{-}[]\!\!\rightarrow \mathsf{K}(x)$ and replacing the fact symbols as described above yields the given destruction rule.

For arbitrary rewrite rules, the set of deconstruction rules computed as described above is unfortunately not complete. For example, consider the rewrite theory

$$\mathcal{ST} = (\{f(\_, \_), g(\_), e(\_, \_)\}, \{f(g(e(x_1, x_2)), x_2) \rightarrow x_1\}).$$

Note that given $e(\mathsf{a}, \mathsf{b})$ and b, the fresh name a is deducible for this theory since we can first apply g to $e(\mathsf{a}, \mathsf{b})$ and then apply f to the result and b to obtain $f(g(e(\mathsf{a}, \mathsf{b})), \mathsf{b})$, which reduces to a. Using the method described above to compute the normal message deduction rules *ND*, we obtain the single deconstruction rule $\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(g(e(x_1, x_2))), \mathsf{K}^{\Uparrow}(x_2)\right] \,\text{-}[]\!\!\rightarrow \mathsf{K}^{\Downarrow^{\mathsf{d}}}(x_1)$ for f and construction rules for f, g, and e. These rules are not sufficient to deduce $\mathsf{K}^{\Uparrow}(\mathsf{a})$ from $\mathsf{K}^{\Downarrow^{\mathsf{d}}}(e(\mathsf{a}, \mathsf{b}))$ and $\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\mathsf{b})$. To apply the deconstruction rule, the fact $\mathsf{K}^{\Downarrow^{\mathsf{d}}}(g(e(\mathsf{a}, \mathsf{b})))$ is required. Using the construction rule for g and COERCE, it is only possible to deduce $\mathsf{K}^{\Uparrow}(g(e(\mathsf{a}, \mathsf{b})))$, which cannot be used with the deconstruction rule for f. We therefore need another deconstruction rule $\mathsf{K}^{\Downarrow^{\mathsf{d}}}(e(x_1, x_2)), \mathsf{K}^{\Uparrow}(x_2) \,\text{-}[]\!\!\rightarrow \mathsf{K}^{\Downarrow^{\mathsf{d}}}(x_1)$ which performs the required construction and deconstruction in one step.

We therefore compute the normal message deduction rules for $\mathcal{ST}$ as follows. For each n-ary function symbol $f \in \Sigma_{\mathcal{ST}}$, the construction rule is

$$[\mathsf{K}^{\Uparrow}(x_1), \ldots, \mathsf{K}^{\Uparrow}(x_n)] \,\text{-}[]\!\!\rightarrow \mathsf{K}^{\Uparrow}(f(x_1, \ldots, x_n)).$$

We compute the destruction rules from the rewriting system $\mathcal{R}_{\mathcal{ST}}$. For each rewriting rule $l \rightarrow r \in \mathcal{R}_{\mathcal{ST}}$, we distinguish two cases. If r is ground, then we do not require a deconstruction rule for this rewriting rule since r can be directly deduced using only construction rules. If there is a position p with $l|_p = r$, then we use the function *drules* to compute the corresponding deconstruction rules

$$\mathit{drules}(l, p) = \left\{\left(\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(l|_{p'})\right] \cdot \mathit{cprems}(l, p')\right) \,\text{-}[]\!\!\rightarrow \left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(l|_p)\right] \mid p' \text{ strictly above } p \text{ and } p' \neq [\,]\right\}.$$

Intuitively, *drules* returns one deconstruction rule for each subterm of l that contains the position p except for $l|_p$ and l itself. The *drules* function uses the function *cprems* to compute the required $\mathsf{K}^{\Uparrow}$-premises. We use the function *seq* that converts sets into sequences to define *cprems* as

$$\mathit{cprems}(l,p) = \mathit{seq}(\{\mathsf{K}^{\Uparrow}(l|_{p'}) \mid p' \neq [\,] \wedge p' \text{ has sibling that is above or equal to } p\}).$$

The rules returned by $\mathit{drules}(l,p)$ express that the adversary can deduce an instance m of l to extract the corresponding subterm $m|_p$.

Example 3.15. To see how the computation of the deconstruction rules works, consider $\mathit{drules}(l,[1,1])$ for $l = \mathit{dec}(\mathit{enc}(x,\mathit{pk}(y)),y)$. The only position $p' \neq [\,]$ strictly above $[1,1]$ is $[1]$. We therefore obtain the only rule $\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\mathit{enc}(x,\,\mathit{pk}(y)))\right] \cdot [\mathsf{K}^{\Uparrow}(y)] \;-\!\![]\!\!\rightarrow \mathsf{K}^{\Downarrow^{\mathsf{d}}}(y)$. The second premise is equal to $\mathit{cprems}(l,[1])$ since $l|_{[2]}{=}y$ and the position $[2]$ is the only sibling of the position $[1]$ or of a position above $[1]$ that is not equal to $[\,]$.

Example 3.16. We can now turn to the problematic case described earlier. The right-hand side of the rewrite rule $f(g(e(x_1,x_2)),x_2) \rightarrow x_1$ occurs at position $[1,1,1]$. We compute the deconstruction rule for this rewrite rule with $\mathit{drules}(l,\,[1,\,1,\,1])$ for $l = f(g(e(x_1,x_2)),x_2)$. There are two positions $p' \neq [\,]$ strictly above $[1,1,1]$, namely $[1,1]$ and $[1]$. We therefore obtain the two deconstruction rules

$$\left(\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(g(e(x_1,x_2)))\right] \cdot \mathit{cprems}(l,[1])\right) -\!\![]\!\!\rightarrow \left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(x_1)\right]$$

and

$$\left(\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(e(x_1,x_2))\right] \cdot \mathit{cprems}(l,[1,1])\right) -\!\![]\!\!\rightarrow \left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(x_1)\right]$$

where the remaining $\mathsf{K}^{\Uparrow}$-premises are computed by $\mathit{cprems}(l,[1])$ and $\mathit{cprems}(l,[1,1])$. The first case captures that the adversary extracts x_1 by applying f to $g(e(x_1,\,x_2))$ and the remaining arguments computed by *cprems*. The second case captures that the adversary extracts x_1 by applying g to $e(x_1,\,x_2)$ and f to the result. Here, *cprems* computes the required arguments for both the application of g and the application of f. For the first rule, $\mathit{cprems}(l,[1])$ returns the sequence $[\mathsf{K}^{\Uparrow}(x_2)]$ since $l|_{[2]}{=}x_2$ and $[2]$ is the position of the the required second argument for the application of f. For the second rule, $\mathit{cprems}(l,[1,1])$ also returns the sequence $[\mathsf{K}^{\Uparrow}(x_2)]$ since $[1,\,1]$ has no siblings in l. This reflects that the application of g performed by this rule does not require any arguments except for $e(x_1,x_2)$. Intuitively, $\mathit{cprems}(l,p)$ starts at the position p and traverses l upwards and collects all siblings, which correspond to the required arguments for the performed function applications.

Example 3.17. Figure 3.7 shows four message deduction subgraphs. In (a), the adversary decrypts a message that he earlier encrypted himself. Instead of performing these deductions, the adversary can directly use the conclusion $\mathsf{K}(\mathsf{a})$ that is used by i_1. The deduction from (a) is not possible with the normal message deduction rules *ND* as depicted in (b). Subfigures (c) and (d) show two different ways to deduce $\mathsf{K}^{\Uparrow}(\langle\mathsf{a},\mathsf{b}\rangle)$ from $\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\langle\mathsf{a},\mathsf{b}\rangle)$. We will later introduce an additional normal form condition that forbids deduction (c).

3.2.4 Normal Dependency Graphs

We now define normal dependency graphs. They use normal message deduction rules and enforce further normal-form conditions. To state the normal-form conditions, we first define the subset of invertible function symbols. An n-ary function

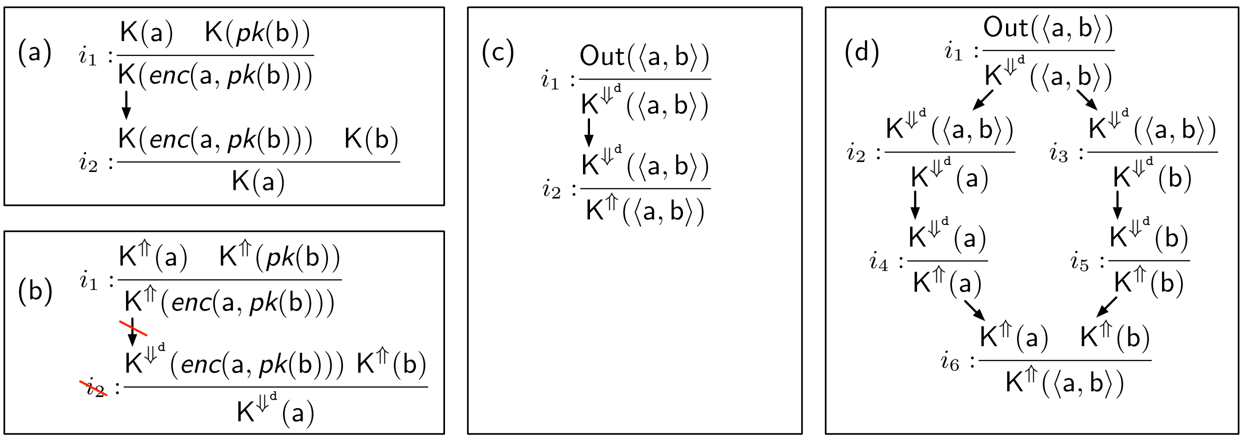


Figure 3.7. Message deduction subgraphs for $\mathcal{DY}$. The crossed edge is not allowed since source and target are not equal.

symbol $f \in \Sigma_{\mathcal{ST}} \cup \Sigma_{\mathcal{DH}}$ is *invertible* if for arbitrary terms t_i and for all $1 \leq i \leq n$, $\mathsf{K}^{\Downarrow^{\mathsf{d}}}(f(t_1, \dots, t_n)) \,\text{-}[]\!\!\rightarrow \mathsf{K}^{\Downarrow^{\mathsf{d}}}(t_i) \in \mathit{ginsts}_{AC}(ND)$, i.e., there is an AC-instance of a deconstruction rule that extracts t_i. For $\mathcal{DH}$, the function symbol $\_^{-1}$ is the only invertible function symbol and for the $\mathcal{DY}$ theory, $\langle \_, \_ \rangle$ is the only invertible function symbol. A message where the outermost function symbol is invertible can always be deduced by applying the corresponding construction rule.

Definition 3.18. A *normal dependency graph for a protocol P* is a dependency graph dg such that $dg \in \mathit{dgraphs}_{AC}(\lceil P \rceil^{\mathcal{RDH}_e}_{\mathit{insts}} \cup ND)$ and the following conditions are satisfied.

N1. The dependency graph dg is $\downarrow_{\mathcal{RDH}_e}$-normal.

N2. There is no multiplication rule that has a premise fact of the form $\mathsf{K}^{\Uparrow}(s{*}t)$ and all conclusion facts of the form $\mathsf{K}^{d}(s{*}t)$ are conclusions of a multiplication rule.

N3. If there are two conclusions c and c' with conclusion facts $\mathsf{K}^{d}(m)$ and $\mathsf{K}^{d'}(m')$ such that $m =_{AC} m'$ and *either* $d = d' = \Uparrow$ *or* $d = \Downarrow^{y}$ and $d' = \Downarrow^{y'}$ for $y, y' \in \{\mathsf{d}, \mathsf{e}\}$, then $c = c'$.

N4. All conclusion facts $\mathsf{K}^{\Uparrow}(f(t_1, \dots, t_n))$ where f is an invertible function symbol are conclusions of the construction rule for f.

N5. If a node i has a conclusion $\mathsf{K}^{\Downarrow^{y}}(m)$ for $y \in \{\mathsf{d}, \mathsf{e}\}$ and a node j has a conclusion $\mathsf{K}^{\Uparrow}(m')$ with $m =_{AC} m'$, then $i < j$ and either $\mathit{root}(m)$ is invertible or the node j is an instance of Coerce.

N6. There is no node $\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(a), \mathsf{K}^{\Uparrow}(b)\right] \,\text{-}[]\!\!\rightarrow \mathsf{K}^{\Downarrow^{\mathsf{e}}}(c\,\hat{}\,d)$ where c does not contain any fresh names and $\mathit{nifactors}(d) \subseteq_{AC} \mathit{nifactors}(b)$.

We denote the set of all normal dependency graphs for P with $\mathit{ndgraphs}(P)$.

Condition **N1** ensures that all rule instances are $\downarrow_{\mathcal{RDH}_e}$-normal. Condition **N2** formalizes that the adversary constructs all products directly by multiplying their components. Condition **N3** ensures that the same message never has multiple $\mathsf{K}^{\Downarrow^{\mathsf{d}}}$, $\mathsf{K}^{\Downarrow^{\mathsf{e}}}$, or $\mathsf{K}^{\Uparrow}$ deductions.

Condition **N4** ensures that terms m where $root(m)$ is invertible are never deduced by Coerce. Condition **N5** forbids two types of redundancies. First, if there is already a normal deduction for a message, then there is no need for a later deconstruction of the same message. Second, if there is already a deconstruction of a message, then the Coerce rule should be used to create a normal deduction unless it is forbidden by condition **N4**. Condition **N6** forbids instances of exponentiation rules that can be directly replaced by the construction rule for exponentiation.

Note that normal dependency graphs have the same observable traces as dependency graphs modulo AC using the message deduction rules MD.

Lemma 3.19. *For all protocols P,*

$$\{\overline{trace(dg)} \mid dg \in dgraphs_{AC}(\lceil P \cup MD \rceil^{\mathcal{RDH}_e}_{insts}) \wedge dg \downarrow_{\mathcal{RDH}_e}\text{-}normal\} = \overline{trace(ndgraphs(P))}.$$

In Appendix A, we prove Lemma A.12, which is an extended version of this lemma for bilinear pairings and AC operators, which we will introduce later on. Combining this result with Lemma 3.10 and Lemma 3.11, we conclude that normal dependency graphs have the same observable traces as our multiset rewriting semantics.

Corollary 3.20. *For all protocols P,* $\overline{trace(execs(P \cup MD))\downarrow_{\mathcal{RDH}_e}} =_{AC} \overline{trace(ndgraphs(P))}$.

Example 3.21. The dependency graph from Figure 3.4 can be converted into the normal dependency graph depicted in Figure 3.8 as follows. First, the rules for message deduction are replaced by the corresponding rules from *ND* where possible. This requires a a Coerce node between Node 10 and 12 in the normal dependency graph. Between the conclusion of Node 6 and the second premise of Node 10 in the normal dependency graph, we cannot simply add a Coerce node because of condition **N4**. We therefore add the deconstruction rule for the inverse followed by Coerce and the construction rule for inverse. This is similar to the deductions (c) and (d) in Figure 3.7 where only deduction (d) is allowed.

Properties of Normal Dependency Graphs

We prove a property of normal dependency graphs that is crucial for our constraint solving algorithm. It states that every $\mathsf{K}^{\Downarrow y}(t)$-premise is deduced using a chain of deconstruction rules from a received message. We use here the *extended set of deconstruction rules* ND^{decon} that consists of the deconstruction and exponentiation rules from Figure 3.5 and the deconstruction rules for $\mathcal{ST}$.

Definition 3.22. Let $dg = (I, D)$ be a normal dependency graph for P. Its *deconstruction chain relation* $\twoheadrightarrow_{dg}$ is the smallest relation such that $i \twoheadrightarrow_{dg} p$ if $(i, 1)$ is a $\mathsf{K}^{\Downarrow y}$-conclusion in dg for $y \in \{\mathtt{d}, \mathtt{e}\}$ and (a) $(i, 1) \rightarrowtail p \in D$ or (b) there is a premise $(j, 1)$ such that $(i, 1) \rightarrowtail (j, 1) \in D$ and $j \twoheadrightarrow_{dg} p$.

Our algorithm exploits the following lemma to reason about the possible origins of $\mathsf{K}^{\Downarrow y}$-premises.

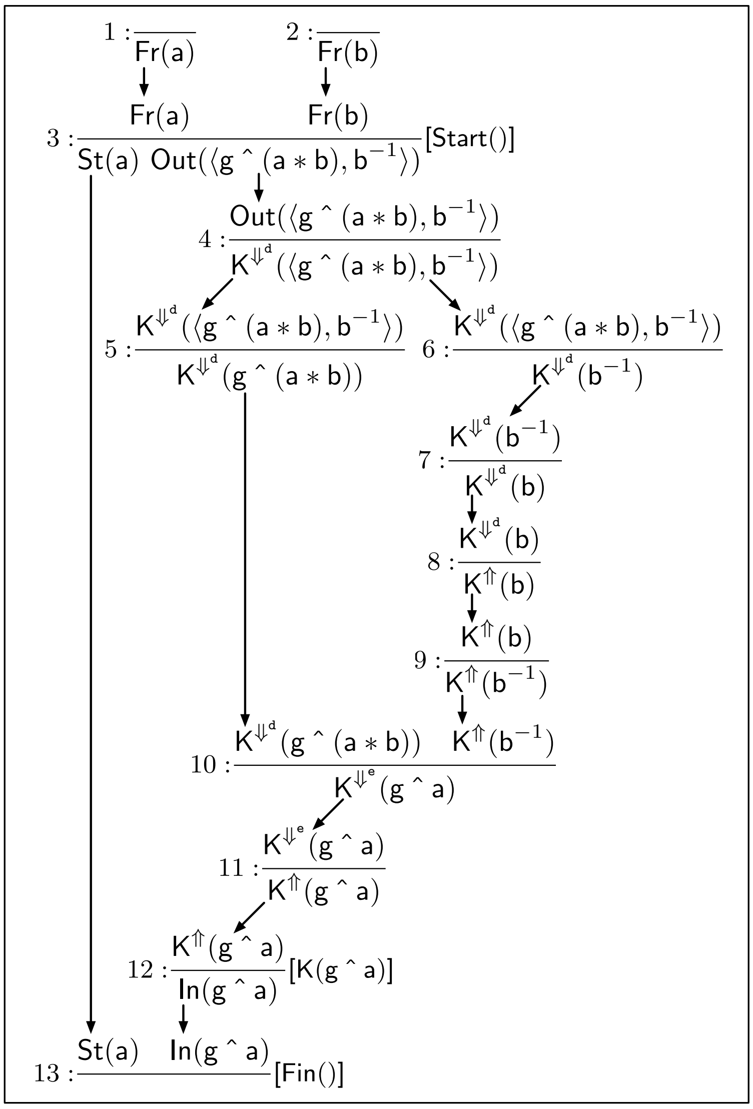

1 : Fr(a)
2 : Fr(b)
3 : Fr(a) Fr(b) / St(a) Out(⟨g ^ (a ∗ b), b⁻¹⟩) [Start()]
4 : Out(⟨g ^ (a ∗ b), b⁻¹⟩) / K⇓d(⟨g ^ (a ∗ b), b⁻¹⟩)
5 : K⇓d(⟨g ^ (a ∗ b), b⁻¹⟩) / K⇓d(g ^ (a ∗ b))
6 : K⇓d(⟨g ^ (a ∗ b), b⁻¹⟩) / K⇓d(b⁻¹)
7 : K⇓d(b⁻¹) / K⇓d(b)
8 : K⇓d(b) / K⇑(b)
9 : K⇑(b) / K⇑(b⁻¹)
10 : K⇓d(g ^ (a ∗ b)) K⇑(b⁻¹) / K⇓e(g ^ a)
11 : K⇓e(g ^ a) / K⇑(g ^ a)
12 : K⇑(g ^ a) / In(g ^ a) [K(g ^ a)]
13 : St(a) In(g ^ a) / [Fin()]

Figure 3.8. Normal dependency graph for P_{Msg}.

Lemma 3.23. *Let dg be a normal dependency graph. For every premise p of dg with fact* $\mathsf{K}^{\Downarrow^y}(m)$ *for* $y \in \{\mathsf{d}, \mathsf{e}\}$*, there is a node i in dg such that* $I_i \in ginsts_{AC}(\textsc{Recv})$ *and* $i \twoheadrightarrow_{dg} p$*.*

The lemma follows from the structure of the rules in *ND*. The only rule in *ND* that has a $\mathsf{K}^{\Downarrow^y}$-conclusion, but no $\mathsf{K}^{\Downarrow^y}$-premise is Recv.

Example 3.24. For example, in the dependency graph dg from Figure 3.8, $4 \twoheadrightarrow_{dg} (11, 1)$, $5 \twoheadrightarrow_{dg} (11, 1)$, and $5 \twoheadrightarrow_{dg} (10, 1)$, but not $6 \twoheadrightarrow_{dg} (11, 1)$ since conclusion $(8, 1)$ is a $\mathsf{K}^{\Uparrow}$-fact.

3.2.5 Guarded Trace Formulas

In the following, 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 $\bot$ and all other logical operators are $\wedge$, $\vee$, $\forall$, and $\exists$.

Definition 3.25. A trace formula φ in negation normal form is a *guarded trace formula* if all its quantifiers are of the form $\exists \vec{x}.g \wedge \psi$ or $\forall \vec{x}.\neg g \vee \psi$ such that

G1. $\vec{x} \subseteq \mathcal{V}_{msg} \cup \mathcal{V}_{temp}$ and

G2. either

a) g is an action $f@i$ and $\vec{x} \subseteq vars(f@i)$ or

b) g is an equality $s \approx t$, $vars(s) \cap \vec{x} = \emptyset$ and $\vec{x} \subseteq vars(t)$.

A guarded trace formula φ is a *guarded trace property* if it is closed and for all subterms t of φ, $root(t)$ is a variable, a public name, or an irreducible function symbol from $\Sigma_{\mathcal{ST}}$.

Intuitively, the guarding of quantified variables by actions ensures that for checking if a trace tr satisfies a guarded trace property φ, we only have to consider assignments of subterms of the trace to these variables. For variables guarded by equalities $s \approx t$, we only have to consider assignments of subterms of s to these variables. 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.

In our case studies, it was possible to automatically convert the specified security properties, including the properties of the *UM* model from Figure 3.3, to guarded trace properties. The conversion first rewrites the given formula to negation normal form and pushes quantifiers inward. Then, it replaces each body φ of an existential quantifier that is not a conjunction with $\varphi \wedge \neg\bot$. The rewriting for universal quantifiers is analogous.

Terms in guarded trace properties cannot use reducible function symbols. 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. Together with the requirement of guarding all quantified variables, this ensures that satisfaction of guarded trace properties modulo AC and satisfaction modulo $\mathcal{DH}_e$ coincide for $\downarrow_{\mathcal{RDH}_e}$-normal traces.

Lemma 3.26. *For all $\downarrow_{\mathcal{RDH}_e}$-normal traces tr and guarded trace properties φ, $tr \vDash_{\mathcal{DH}_e} \varphi$ if and only if $tr \vDash_{AC} \varphi$.*

In Appendix A, we prove Lemma A.20, which is an extended version of this lemma for bilinear pairings and AC operators that can be used in formulas. The following theorem allows us to switch from verification in a multiset rewriting semantics modulo $\mathcal{DH}_e$ to verification in a dependency graph semantics modulo AC.

Theorem 3.27. *For all protocols P and guarded trace properties φ,*

$$trace(execs(P \cup MD)) \vDash_{\mathcal{DH}_e} \varphi \quad \text{if and only if} \quad trace(ndgraphs(P)) \vDash_{AC} \varphi.$$

Proof. The proofs proceeds as follows.

$$\begin{array}{lll} & trace(execs(P \cup MD)) \vDash_{\mathcal{DH}_e} \varphi & \\ \text{iff} & \overline{trace(execs(P \cup MD))} \vDash_{\mathcal{DH}_e} \varphi & [\text{by Lemma 3.7}] \\ \text{iff} & \overline{trace(execs(P \cup MD))}\downarrow_{\mathcal{RDH}_e} \vDash_{\mathcal{DH}_e} \varphi & [\text{Definition of } \vDash_{\mathcal{DH}_e}] \end{array}$$

$$\begin{array}{lll}
\text{iff} & \overline{\mathit{trace}(\mathit{ndgraphs}(P))} \vDash_{\mathcal{DH}_e} \varphi & [\,\text{by Corollary 3.20}\,] \\
\text{iff} & \mathit{trace}(\mathit{ndgraphs}(P)) \vDash_{AC} \varphi & [\,\text{by Lemma 3.7 and 3.26}\,]
\end{array}$$

Note that the last step exploits that φ is a guarded trace property. □

3.3 Constraint Solving

In this section, we give an algorithm that determines for a guarded trace property φ and a protocol P if φ is P,$\mathcal{DH}_e$-satisfiable. Our algorithm uses constraint solving to search for traces of P that are models of φ. If it terminates, it has either found such a trace or, since the search is complete, it has proved that φ is unsatisfiable by traces of P. To analyze if *all* traces of a protocol satisfy a property φ, we must check for validity instead of satisfiability. Since these notions are dual, we can determine if φ is P,$\mathcal{DH}_e$-valid by using our algorithm to check if $\neg\varphi$ is P,$\mathcal{DH}_e$-satisfiable. If we find a trace, then this a counterexample to the validity of φ, i.e., there is an attack. Both problems are undecidable and our algorithm does not always terminate. Nevertheless, it terminates for many relevant protocols and properties. In the following, we define the syntax and semantics of constraints. Afterwards, we give our constraint solving algorithm and rules, explain two optimizations, and present several examples.

3.3.1 Syntax and Semantics of Constraints

In the remainder of this section, let *ri* and *ru* range over multiset rewriting rules, u and v and w over natural numbers, and φ over guarded trace formulas. A *graph constraint* is either a *node* $i\colon ri$, an *edge* $(i,u) \rightarrowtail (j,v)$, a *deconstruction chain* $i \twoheadrightarrow (j,v)$, or a *provides* $i \rhd f$ which denotes that f is the first conclusion of the node i. A *constraint* is a graph constraint or a guarded trace formula.

A *structure* is a tuple $(dg,\, \theta)$ of a dependency graph $dg = (I,\, D)$ and a valuation θ. We denote the application of the homomorphic extension of θ to a rule ru by $ru\theta$. The satisfaction relation $\Vdash$ between structures and constraints is inductively defined by the following rules:

$$\begin{array}{ll}
(dg,\theta) \Vdash i\colon ri & \text{if } \theta(i) \in idx(I) \text{ and } ri\theta =_{AC} I_{\theta(i)} \\
(dg,\theta) \Vdash (i,u) \rightarrowtail (j,v) & \text{if } (\theta(i),u) \rightarrowtail (\theta(j),v) \in D \\
(dg,\theta) \Vdash i \twoheadrightarrow (j,v) & \text{if } \theta(i) \twoheadrightarrow_{dg} (\theta(j),v) \\
(dg,\theta) \Vdash i \rhd f & \text{if } \theta(i) \in idx(I) \text{ and } (concs(I_{\theta(i)}))_1 =_{AC} f\theta \\
(dg,\theta) \Vdash \varphi & \text{if } (trace(dg),\theta) \vDash_{AC} \varphi
\end{array}$$

A *constraint system* Γ is a finite set of constraints. The structure (dg,θ) satisfies Γ, written $(dg,\theta) \Vdash \Gamma$, if (dg,θ) satisfies each constraint in Γ. We say that (dg,θ) is a P-solution of Γ, if dg is a normal dependency graph for P and $(dg,\,\theta) \Vdash \Gamma$. A P-model of Γ is a normal dependency graph dg for P such that there is a valuation θ with $(dg,\theta) \Vdash \Gamma$. Note that the free variables of a constraint system are therefore considered as existentially quantified. We use $models_P(\Gamma)$ to denote the set of all P-models of Γ.

Example 3.28. The dependency graph from Figure 3.8 satisfies the constraints

- $i\colon \mathsf{K}^{\Downarrow^{\mathsf{d}}}(x), \mathsf{K}^{\Uparrow}(\mathsf{b}^{-1}) \mathrel{-\!\![]\!\!\rightarrowtail} \mathsf{K}^{\Downarrow^{\mathsf{e}}}(\mathsf{g\hat{}a})$ for all valuations θ with $\theta(i)=10$ and $\theta(x)=_{AC}\mathsf{g\hat{}(a{*}b)}$,
- $(j,1)\rightarrowtail(k,2)$ for θ with $\theta(j)=2$ and $\theta(k)=3$,
- $i\twoheadrightarrow(l,1)$ for θ with $\theta(i)=10$ and $\theta(l)=11$,
- $i\vartriangleright\mathsf{K}^{\Uparrow}(\mathsf{b})$ for θ with $\theta(i)=8$, and
- $\forall i\, y.\neg\mathsf{K}(h(y))@i\vee\bot$ since there is no K-action whose argument is a hash.

3.3.2 Constraint Solving Algorithm

As mentioned before, validity can be reduced to satisfiability. Moreover, Theorem 3.27 allows us to use normal dependency graphs and satisfaction modulo AC. Given a guarded trace property φ and a protocol P, we can therefore focus on the problem of finding normal dependency graphs $dg\in \mathit{ndgraphs}(P)$ such that $\mathit{trace}(dg)\vDash_{AC}\varphi$. This is equivalent to the problem of finding P-models for the constraint system $\{\varphi\}$. To achieve this, we define a constraint solving relation $\rightsquigarrow_P$ that reduces a constraint system to a finite set of constraint systems by inferring new constraints and performing case splits. The relation $\rightsquigarrow_P$ is sound and complete in the following sense: whenever $\Gamma\rightsquigarrow_P\{\Gamma_1,\ldots,\Gamma_k\}$, then the set of P-models of Γ is equal to the union of the sets of P-models of the Γ_i. We overload notation and define the reflexive-transitive closure $\rightsquigarrow_P^*$ of $\rightsquigarrow_P$ as the least relation such that $\Gamma\rightsquigarrow_P^*\mathbf{\Delta}$ if (a) $\mathbf{\Delta}=\{\Gamma\}$ or (b) $\Gamma\rightsquigarrow_P\{\Gamma_1,\ldots,\Gamma_k\}$, $\Gamma_i\rightsquigarrow_P^*\mathbf{\Delta}_i$, and $\mathbf{\Delta}=\bigcup_{i=1}^k\mathbf{\Delta}_i$. We call $\Gamma\rightsquigarrow_P^*\mathbf{\Delta}$ a *constraint reduction*.

The relation $\rightsquigarrow_P^*$ can be used to check P,$\mathcal{DH}_e$-satisfiability of φ as follows. We search for a constraint reduction $\{\varphi\}\rightsquigarrow_P^*\emptyset$ or a constraint reduction $\{\varphi\}\rightsquigarrow_P^*\mathbf{\Delta}$ such that there is a constraint system $\Gamma\in\mathbf{\Delta}$ for which we can directly confirm that it has a P-model. In the first case, we have proved that φ is not satisfiable. In the second case, we can extract a P-model dg of φ from Γ. We call such a constraint system *solved* and will later formally define this notion.

To check P,$\mathcal{DH}_e$-validity of φ, we first rewrite $\neg\varphi$ into a guarded trace property $\hat{\varphi}$, which is always possible since φ is a guarded trace property. Then, a constraint reduction $\{\hat{\varphi}\}\rightsquigarrow_P^*\emptyset$ proves the validity of φ and a constraint reduction $\{\hat{\varphi}\}\rightsquigarrow_P^*\mathbf{\Delta}$, where at least one constraint system in $\mathbf{\Delta}$ is solved, provides a counterexample to φ.

An algorithm based on $\rightsquigarrow_P$ can therefore use the following approach. To check satisfiability of the constraint system $\Gamma=\{\varphi\}$, it maintains a set $\mathbf{\Delta}$ of constraint systems as its state. The algorithm starts with the initial state $\mathbf{\Delta}:=\{\Gamma\}$ and maintains the invariant that $\Gamma\rightsquigarrow_P^*\mathbf{\Delta}$. In each step, it checks first if $\mathbf{\Delta}=\emptyset$ and otherwise checks if $\mathbf{\Delta}$ contains a solved constraint system. If the first check succeeds, the algorithm has proved that φ is not satisfiable for P. If the second check succeeds, the algorithm has proved that φ is satisfiable for P and can construct a P-model of φ. If both checks fail, the algorithm can choose a $\Delta\in\mathbf{\Delta}$ and a constraint solving step $\Delta\rightsquigarrow_P\mathbf{\Delta}'$. Then, it can update its state to $\mathbf{\Delta}:=(\mathbf{\Delta}\setminus\Delta)\cup\mathbf{\Delta}'$ and proceed with the next step. This is a standard approach to obtain a constraint solving algorithm from a constraint solving relation by providing a specific control that chooses one of many possible steps. We will provide intuition for devising a sensible control strategy in the examples in this section. In Section 3.5.1, we will sketch the control strategy used in our implementation of the algorithm in the Tamarin prover and explain the interactive mode where the user can provide the control strategy.

3.3.3 Constraint Solving Rules

We will now define the constraint solving relation $\rightsquigarrow_P$ in three steps. First, we introduce notation and definitions required to formalize the constraint solving rules. Then, we present all the constraint solving rules except those for handling message deduction and show an example constraint reduction to provide intuition. Finally, we present the constraint solving rules for message deduction, properties of the constraint solving relation, and an example.

We first define auxiliary functions on constraint systems. The *actions of a constraint system* Γ are defined as

$$as(\Gamma) = \{f@i \mid \exists i\, ri.i{:}\, ri \in \Gamma \wedge f \in acts(ri)\}.$$

The *implicit ordering of a constraint system* Γ is defined as

$$\prec_\Gamma = \{(i,j) \mid (\exists uv.(i,u) \rightarrowtail (j,v) \in \Gamma) \vee (\exists v.i \twoheadrightarrow (j,v) \in \Gamma) \vee (i \prec j \in \Gamma)\}^+.$$

We define $\rightsquigarrow_P$ as the least relation closed under the rules given in Figures 3.9, 3.10, 3.15, and 3.16. In our presentation of the constraint solving relation, we use the following notation. For constraints c and constraint systems Γ, we write c, Γ to denote $\{c\} \cup \Gamma$. We distinguish between two types of rules, *Insertion rules* where

$$\frac{c_1, \ldots, c_l}{\Delta_1 | \ldots | \Delta_k}\mathcal{I} \qquad \text{denotes that} \qquad \Gamma \rightsquigarrow_P \{\Gamma \cup \Delta_1, \ldots, \Gamma \cup \Delta_k\} \text{ if } \{c_1, \ldots, c\} \subseteq_{AC} \Gamma$$

and *modification rules* where

$$\frac{\Gamma}{\Gamma'_1 | \ldots | \Gamma'_k}\mathcal{M} \qquad \text{denotes that} \qquad \Gamma \rightsquigarrow_P \{\Gamma'_1, \ldots, \Gamma'_k\}.$$

We assume that all sets of multiset rewriting rules that are used in the side conditions of rules are renamed away from Γ and freshly chosen variables. For some of the rules, we use the overloaded notation $a \approx b$ for facts, natural numbers, or rule instances a and b. This denotes the equality constraint resulting from encoding a and b as terms. In our constraints, we also use a binary fact symbol $\mathsf{K}^{\Downarrow_-}(\_)$ instead of $\mathsf{K}^{\Downarrow_\mathsf{e}}(\_)$ and $\mathsf{K}^{\Downarrow_\mathsf{d}}(\_)$. This allows us to encode $\mathtt{e}$ and $\mathtt{d}$ as terms and use a variable of sort *msg* to represent both alternatives.

Basic Constraint Solving Rules

The first set of rules given in Figure 3.9 deals with guarded formulas. Note that the insertion rules can be applied repeatedly without yielding any new constraints and we will later define a notion of redundancy. The rule $\mathcal{S}_@$ solves a constraint $f@i$ by performing a case split over all rules and their actions that might be equal to f. In each case, a node constraint $j{:}\, ru_i$ for a freshly chosen temporal variable j and a freshly renamed instance ru_i of a variant of a protocol rule or Send is added. Additionally, an equality constraint between f and one of the actions of ru_i is added. The rule $\mathcal{S}_\approx$ solves equality constraints by performing unification. Since AC-unification is not unitary, this might result in case splits. Note that the rule can also be used to solve timepoint equalities. The rule $\mathcal{S}_\exists$ insert the body of an existential quantification and replaces the bound variables with freshly chosen variables. The rule $\mathcal{S}_{\forall,@}$ solves a $\forall$-quantification that is guarded by an action. The bound variables in φ and $f@i$ are instantiated by σ such that $(f@i)\sigma$ holds in Γ and $\varphi\sigma$ is then added to the constraint system. The rule $\mathcal{S}_{\forall,\approx}$ solves a $\forall$-quantification that is guarded by an equality. The bound variables in t and φ are instantiated by σ such that the equality holds and $\varphi\sigma$

$$\mathcal{S}_{@}\colon \quad \frac{f@i}{i\colon ru_1, g_1 \approx f \,|\, \ldots \,|\, i\colon ru_l, g_l \approx f}\,\mathcal{I} \qquad \text{if } \{(ru_1, g_1), \ldots, (ru_l, g_l)\} = \{(ru, g) \mid ru \in \lceil P \rceil^{\mathcal{RDH}_e}_{insts} \cup \{\textsc{Send}\} \wedge g \in acts(ru)\}$$

$$\mathcal{S}_{\approx}\colon \quad \frac{s \approx t, \Gamma}{\Gamma\sigma_1 \,|\, \ldots \,|\, \Gamma\sigma_l}\,\mathcal{M} \qquad \text{if } \{\sigma_1, \ldots, \sigma_l\} = unif^{fvars(\Gamma)}_{AC}(s, t)$$

$$\mathcal{S}_{\exists}\colon \quad \frac{\exists\, \vec{x}\,.\varphi}{\varphi\{\vec{y}\,/\vec{x}\,\}}\,\mathcal{I} \qquad \text{if } \vec{y} \text{ freshly chosen variables such that } |\vec{x}\,| = |\vec{y}\,| \text{ and } x_i \text{ and } y_i \text{ have the same sort.}$$

$$\mathcal{S}_{\forall,@}\colon \quad \frac{\forall\, \vec{x}\,.\neg(f@i) \vee \varphi}{\varphi\sigma}\,\mathcal{I} \qquad \text{if } g\,@\,j \in as(\Gamma),\ \sigma \in match_{AC}(g@\,j, f@i), \text{ and } dom(\sigma) \subseteq \vec{x}$$

$$\mathcal{S}_{\forall,\approx}\colon \quad \frac{\forall\, \vec{x}\,.\neg(s \approx t) \vee \varphi}{\varphi\sigma}\,\mathcal{I} \qquad \text{if } \sigma \in match_{AC}(s, t) \text{ and } dom(\sigma) \subseteq \vec{x}$$

$$\mathcal{S}_{\vee}\colon \quad \frac{\varphi_1 \vee \varphi_2}{\varphi_1 \,|\, \varphi_2}\,\mathcal{I} \qquad \mathcal{S}_{\wedge}\colon \quad \frac{\varphi_1 \wedge \varphi_2}{\varphi_1, \varphi_2}\,\mathcal{I}$$

$$\mathcal{S}_{\not\approx}\colon \quad \frac{\neg(t \approx t)}{\perp}\,\mathcal{I} \qquad \mathcal{S}_{\perp}\colon \quad \frac{\perp, \Gamma}{}\,\mathcal{M} \qquad \text{(the result is the empty set of constraint systems)}$$

$$\mathcal{S}_{\neg @}\colon \quad \frac{\neg(f@i), \Gamma}{\perp}\,\mathcal{M} \qquad \text{if } f@i \in_{AC} as(\Gamma)$$

Figure 3.9. Constraint solving rules for guarded formulas.

is then added to the constraint system. The rule $\mathcal{S}_{\vee}$ performs a case distinctions on which disjunct is true. The rule $\mathcal{S}_{\wedge}$ adds both conjuncts to the constraint system. The rule $\mathcal{S}_{\perp}$ replaces $\perp$ with the empty case distinction. The remaining rules deal with negated atoms that cannot be false such as $\neg(t \approx t)$.

The second set of rules given in Figure 3.10 enforces several properties of normal dependency graphs by adding the corresponding constraints. The constraint solving rule $\mathcal{S}_{\mathbf{Prem}}$ solves a non-K^d and non-Fr premise of a node constraint by performing a case distinction over all rules and their conclusion indices that can provide such a conclusion. It adds a node constraint with a fresh temporal variable, a fresh instance of the rule, and an edge from the conclusion to the premise. $\mathcal{S}_{\rightarrowtail}$ can then be used to enforce that the target and source of the edge are equal by adding the corresponding equality. Usually, $\mathcal{S}_{\approx}$ is then directly used to solve this equality. In this case, all cases where the premise fact and the conclusion fact have no unifier disappear and a most general unifier of the two facts is applied to the constraint system in the other cases. $\mathcal{S}_{\mathbf{USrc}}$ ensures that each premise has at most one incoming edge. $\mathcal{S}_{\mathbf{UTgt}}$ ensures that there is only one outgoing edge for linear conclusions. $\mathcal{S}_{\mathbf{ULabel}}$ ensures that each node has a unique rule instance as its label. $\mathcal{S}_{\mathbf{Acyc}}$ ensures that the nodes can be linearly ordered without violating any constraints. $\mathcal{S}_{\mathbf{UFresh}}$ exploits the uniqueness of Fresh rules and ensures that the same Fr-premises is only used once. $\mathcal{S}_{\downarrow}$ ensures that the rule instances are $\downarrow_{\mathcal{RDH}_e}$-normal. So far, we have not shown the rules that deal with message deduction and the remaining conditions on normal dependency graphs. Before showing these, we will present an example constraint solving sequence where message deduction is not required.

$$\mathcal{S}_{\mathbf{Prem}}: \quad \frac{i\colon ri}{\begin{array}{l} j\colon ru_1, (j,v_1) \rightarrowtail (i,u) \\ \mid \ \ldots \\ \mid \ j\colon ru_l, (j,v_l) \rightarrowtail (i,u) \end{array}} \mathcal{I}$$

if $u \in idx(prems(ri))$, $(prems(ri))_u$ not K^d or Fr-fact, and $\{(ru_1, v_1), \ldots, (ru_l, v_l)\} = \{(ru, v) \mid ru \in \lceil P \rceil_{insts}^{\mathcal{RDH}_e} \cup \{\textsc{Send}\} \wedge v \in idx(concs(ru)\}$, and j freshly chosen

$$\mathcal{S}_{\rightarrowtail}: \quad \frac{i\colon ri, j\colon ru, (i,u) \rightarrowtail (j,v)}{(concs(ri))_u \approx (prems(ru))_v} \mathcal{I} \qquad \mathcal{S}_{\mathbf{USrc}}: \quad \frac{(i,u) \rightarrowtail p, (j,v) \rightarrowtail p}{i \approx j, u \approx v} \mathcal{I}$$

$$\mathcal{S}_{\mathbf{UTgt}}: \quad \frac{(i,u) \rightarrowtail (j,v), (i,u) \rightarrowtail (k,w), i\colon ri}{j \approx k, v \approx w} \mathcal{I} \quad \text{if } (concs(ri))_u \text{ linear}$$

$$\mathcal{S}_{\mathbf{ULabel}}: \quad \frac{i\colon ri, i\colon ru}{ri \approx ru} \mathcal{I} \qquad \mathcal{S}_{\mathbf{Acyc}}: \quad \frac{\Gamma}{\bot} \mathcal{M} \quad \text{if } i \prec_\Gamma i \text{ for some } i$$

$$\mathcal{S}_{\mathbf{UFresh}}: \quad \frac{i\colon ri, j\colon ru}{i \approx j, u \approx v} \mathcal{I} \qquad \text{if } (prem(ri))_u = (prem(ru))_v = \mathsf{Fr}(m)$$

$$\mathcal{S}_{\downarrow}: \quad \frac{i\colon ri}{\bot} \mathcal{I} \qquad \text{if } ri \text{ not } \downarrow_{\mathcal{RDH}_e}\text{-normal}$$

Figure 3.10. Constraint solving rules that ensure **DG1–4** and **N1**.

Note that most rules do not remove constraints and it is easy to see that rules such as $\mathcal{S}_@$ can be applied a second time without yielding any useful new constraints. We therefore define the following notion of redundancy.

Definition 3.29. *An application of a constraint solving rule is redundant if it does not add any new constraints except for trivial equalities $t \approx t'$ where $t =_{AC} t'$ or one of the following holds:*

- *The rule is $\mathcal{S}_@$ and $f @ i \in_{AC} as(\Gamma)$.*
- *The rule is $\mathcal{S}_\exists$ and there are terms $\vec{t}$ such that $\varphi\{\vec{t}/\vec{x}\} \in_{AC} \Gamma$.*
- *The rule is $\mathcal{S}_\vee$ and $\varphi_1 \in_{AC} \Gamma$ or $\varphi_2 \in_{AC} \Gamma$.*
- *The rule is $\mathcal{S}_{\mathbf{Prem}}$ and there is an edge $c \rightarrowtail (i,u)$ for some c.*

A constraint system is solved if all rule applications are redundant. We also call a constraint solved if all rule applications using the constraint are redundant.

Example 3.30. We will analyze a modified version of the protocol P_{UM} from Figure 3.2 that uses a private channel between participants. The modified version $P_{UM'}$ can be obtained by replacing the $\mathsf{Out}(X)$ fact in the initiator rule and the $\mathsf{In}(X)$ fact in the responder rule with $\mathsf{PChan}(X)$. We analyze if

$$\varphi_{UM\text{-}exec'} = \exists i\ j\ A\ B\ X\ key.\ \mathsf{Accept}(\langle A, B, X, \mathcal{I}\rangle, key) @ i \wedge \mathsf{Accept}(\langle B, A, X, \mathcal{R}\rangle, key) @ j$$

is satisfiable for $P_{UM'}$. We first compute $\lceil ru \rceil_{insts}^{\mathcal{RDH}_e}$ for all $ru \in P_{UM'}$. The responder rule has the six variants shown in Figure 3.11. For all other rules, $\lceil ru \rceil_{insts}^{\mathcal{RDH}_e} = \{ru{\downarrow_{\mathcal{RDH}_e}}\}$. To

1. $\tau_1 = id$

$$\begin{array}{l} \mathsf{PChan}(X) \\ !\mathsf{Pk}(A{:}pub, \mathsf{g}\,\hat{}\,(a{:}fr)) \\ !\mathsf{Ltk}(B{:}pub, b{:}fr) \\ \hline \end{array}[\mathsf{Accept}(sid, key)]$$

where $sid = \langle B, A, X, \mathcal{R}\rangle$
$key = h(X\,\hat{}\,b, \mathsf{g}\,\hat{}\,(a{*}b), A, B, X)$

2. $\tau_2 = \{u\,\hat{}\,v/X\}$

$$\begin{array}{l} \mathsf{PChan}(u\,\hat{}\,v) \\ !\mathsf{Pk}(A{:}pub, \mathsf{g}\,\hat{}\,(a{:}fr)) \\ !\mathsf{Ltk}(B{:}pub, b{:}fr) \\ \hline \end{array}[\mathsf{Accept}(sid, key)]$$

where $sid = \langle B, A, u\,\hat{}\,v, \mathcal{R}\rangle$
$key = h(u\,\hat{}\,(v{*}b), \mathsf{g}\,\hat{}\,(a{*}b), A, B, X)$

3. $\tau_3 = \{u\,\hat{}\,(b^{-1})/X\}$

$$\begin{array}{l} \mathsf{PChan}(u\,\hat{}\,(b^{-1})) \\ \quad\dots \\ \hline \end{array}[\mathsf{Accept}(sid, key)]$$

where $sid = \langle B, A, u\,\hat{}\,(b^{-1}), \mathcal{R}\rangle$
$key = h(u, \mathsf{g}\,\hat{}\,(a{*}b), A, B, X)$

4. $\tau_4 = \{u\,\hat{}\,(b^{-1}{*}v)/X\}$

$$\begin{array}{l} \mathsf{PChan}(u\,\hat{}\,(b^{-1}{*}v)) \\ \quad\dots \\ \hline \end{array}[\mathsf{Accept}(sid, key)]$$

where $sid = \langle B, A, u\,\hat{}\,(b^{-1}{*}v), \mathcal{R}\rangle$
$key = h(u\,\hat{}\,v, \mathsf{g}\,\hat{}\,(a{*}b), A, B, X)$

5. $\tau_5 = \{u\,\hat{}\,((b{*}v)^{-1})/X\}$

$$\begin{array}{l} \mathsf{PChan}(u\,\hat{}\,((b{*}v)^{-1})) \\ \quad\dots \\ \hline \end{array}[\mathsf{Accept}(sid, key)]$$

where $sid = \langle B, A, u\,\hat{}\,((b{*}v)^{-1}), \mathcal{R}\rangle$
$key = h(u\,\hat{}\,v^{-1}, \mathsf{g}\,\hat{}\,(a{*}b), A, B, X)$

6. $\tau_6 = \{u\,\hat{}\,((b{*}v)^{-1}{*}w)/X\}$

$$\begin{array}{l} \mathsf{PChan}(u\,\hat{}\,((b{*}v)^{-1}{*}w)) \\ \quad\dots \\ \hline \end{array}[\mathsf{Accept}(sid, key)]$$

where $sid = \langle B, A, u\,\hat{}\,((b{*}v)^{-1}{*}w), \mathcal{R}\rangle$
$key = h(u\,\hat{}\,(v^{-1}{*}w), \mathsf{g}\,\hat{}\,(a{*}b), A, B, X)$

Figure 3.11. Variant substitutions and normalized instances for the responder rule of $P_{UM'}$. We suppress the second and third premise in Variants 3–6 since they remain unchanged from Variant 1.

define the initial constraint system, we convert $\varphi_{UM\text{-}exec'}$ into the guarded trace property

$$\varphi = \exists\, i\ A\ B\ X\ key.\mathsf{Accept}(\langle A, B, X, \mathcal{I}\rangle, key)@i \wedge (\exists j.\mathsf{Accept}(\langle B, A, X, \mathcal{R}\rangle, key)@j \wedge \neg\bot).$$

We start with the constraint system $\Gamma_0 = \{\varphi\}$ and apply $\mathcal{S}_\exists$ and $\mathcal{S}_\wedge$ to obtain

$$\Gamma_1 = \left\{ \begin{array}{lll} \varphi & \multicolumn{2}{l}{\mathsf{Accept}(\langle A, B, X, \mathcal{I}\rangle, key)@i \wedge (\exists j.\mathsf{Accept}(\langle B, A, X, \mathcal{R}\rangle, key)@j \wedge \neg\bot)} \\ & \mathsf{Accept}(\langle A, B, X, \mathcal{I}\rangle, key)@i & \exists j.\ \mathsf{Accept}(\langle B, A, X, \mathcal{R}\rangle, key)@j \wedge \neg\bot \\ & & \mathsf{Accept}(\langle B, A, X, \mathcal{R}\rangle, key)@j \quad \neg\bot \end{array} \right\}.$$

Figure 3.12 shows the constraint reduction $\Gamma_1 \rightsquigarrow_P^* \{\Gamma_{1.1.1.1.1.1.1.1}\}$ resulting in the single solved constraint system $\Gamma_{1.1.1.1.1.1.1.1}$. We use grey arrows labeled with constraint solving rules to denote the successor(s) of a constraint system after applying the given rules, possibly multiple times. If such a grey arrow points to $\diamond$, this means that applying the constraint solving rule results in the empty set of constraint systems. We use edges between conclusions and premises of node constraints to denote edge constraints. We use a grey background to emphasize constraints that have been added or modified in the last step and a light-blue background as a hint that the next constraint solving step applies a rule to the given constraint. In some cases, we suppress solved constraints such as $f \approx f$ or φ in Γ_1.

Applying $\mathcal{S}_@$ to the first Accept-fact yields nine constraint systems of the form

$$\Gamma_1, i{:}\, ru, (act(ru))_1 \approx \mathsf{Accept}(\langle A, B, X, \mathcal{I}\rangle, key)@i,$$

one for each non-silent rule. We only show the complete constraint system $\Gamma_{1.1}$ for the initiator rule since the other constraint systems for the long-term reveal rule, the ephemeral reveal rule, and the variants of the responder rule can be shown to be contradictory by applying $\mathcal{S}_{\approx}$ to the added fact-equality. For $\Gamma_{1.1}$, there is only one unifier for the fact-equality and the result of applying $\mathcal{S}_{\approx}$ is the constraint system $\Gamma_{1.1.1}$. Note that the equality has been removed and the constraint $\mathsf{Accept}(SI, K)@i$ is solved since it is already in $as(\Gamma_{1.1.1})$.

Next, we solve the second and the third premise of the node i in three steps each. First, we use $\mathcal{S}_{\mathbf{Prem}}$ to add the node that provides the conclusion and the edge between conclusion and premise. Then we use $\mathcal{S}_{\rightarrowtail}$ to introduce the equality between source and target of the edge. Finally, we use $\mathcal{S}_{\approx}$ to solve the equality. This results in the only constraint system $\Gamma_{1.1.1.1}$.

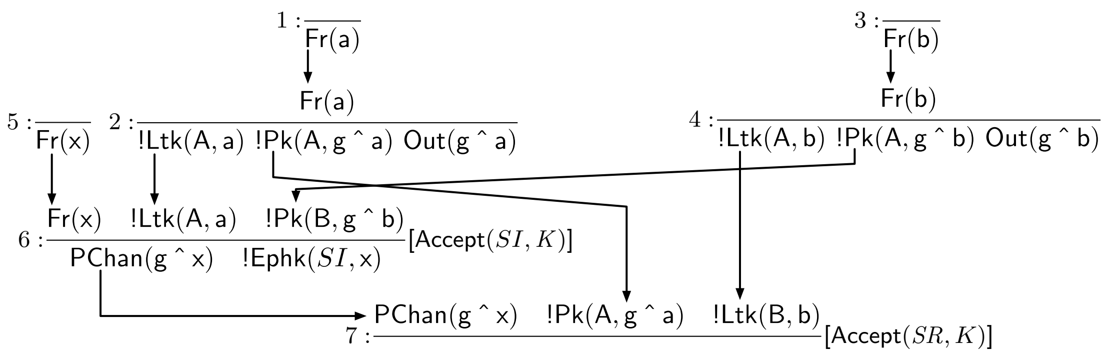
We now solve $\mathsf{Accept}(SR, K)@j$ with $\mathcal{S}_{@}$ followed by $\mathcal{S}_{\approx}$. This results in two intermediate constraint systems which we do not show. One where the first variant of the responder rule is added and one where the second variant is added. In both cases, the X in the rule is instantiated with $\mathsf{g}\hat{\ }(x{:}fr)$. In the first case, the node constraint contains the term

$$key = h((\mathsf{g}\hat{\ }(x{:}fr))\hat{\ }b, \mathsf{g}\hat{\ }(a*b), A, B, \mathsf{g}\hat{\ }(x{:}fr)),$$

which is not $\downarrow_{\mathcal{RDH}_e}$-normal. This case is therefore contradictory by $\mathcal{S}_{\downarrow}$. This reflects that the first variant is reserved for the case where X is not an exponentiation. In the second case, we continue by solving the third premise of the newly added responder node to obtain $\Gamma_{1.1.1.1.1}$.

In this constraint system, the nodes m and l have the same Fr-premise. We therefore use $\mathcal{S}_{\mathbf{UFresh}}$ to obtain $l \approx m$ followed by $\mathcal{S}_{\approx}$ and $\mathcal{S}_{\mathbf{ULabel}}$ to obtain $\Gamma_{1.1.1.1.1.1}$. Finally, we solve the remaining open premises and merge the new key generation nodes with the existing ones using $\mathcal{S}_{\mathbf{UFresh}}$ to obtain the solved constraint system $\Gamma_{1.1.1.1.1.1.1}$.

It is straightforward to obtain a P-model from this constraint system. For example, Figure 3.13 depicts one such model. We instantiate the fresh variables with distinct fresh names and the public variables with distinct public names. Then, we order the existing nodes such that the resulting sequence is compatible with $\prec_{\Gamma_{1.1.1.1.1.1.1}}$. This is possible since $\mathcal{S}_{\mathbf{Acyc}}$ is not applicable and the ordering relation is acyclic. Then, only Fr-premises do not have incoming edges. We therefore add the required FRESH instances immediately before the respective consumers. Finally, we instantiate node variables with the position in the sequence.



1 : Fr(a)
3 : Fr(b)
Fr(a)
Fr(b)
5 : Fr(x)
2 : !Ltk(A, a) !Pk(A, g ^ a) Out(g ^ a)
4 : !Ltk(A, b) !Pk(A, g ^ b) Out(g ^ b)
Fr(x) !Ltk(A, a) !Pk(B, g ^ b)
6 : PChan(g ^ x) !Ephk(SI, x)
[Accept(SI, K)]
PChan(g ^ x) !Pk(A, g ^ a) !Ltk(B, b)
7 :
[Accept(SR, K)]

Abbreviations: $K := h(\mathsf{g}\hat{\ }(\mathsf{b}{*}\mathsf{x}), \mathsf{g}\hat{\ }(\mathsf{b}{*}\mathsf{a}), \mathsf{A}, \mathsf{B}, \mathsf{g}\hat{\ }\mathsf{x})$, $SI := \langle \mathsf{A}, \mathsf{B}, \mathsf{g}\hat{\ }\mathsf{x}, \mathcal{I}\rangle$, $SR := \langle \mathsf{B}, \mathsf{A}, \mathsf{g}\hat{\ }\mathsf{x}, \mathcal{R}\rangle$

Figure 3.13. Extracted model from $\Gamma_{1.1.1.1.1.1.1}$ for $\mathsf{a}, \mathsf{b}, \mathsf{x} \in \mathsf{FN}$ and $\mathsf{A}, \mathsf{B}, \mathsf{g}, \mathcal{I}, \mathcal{R} \in \mathsf{PN}$.

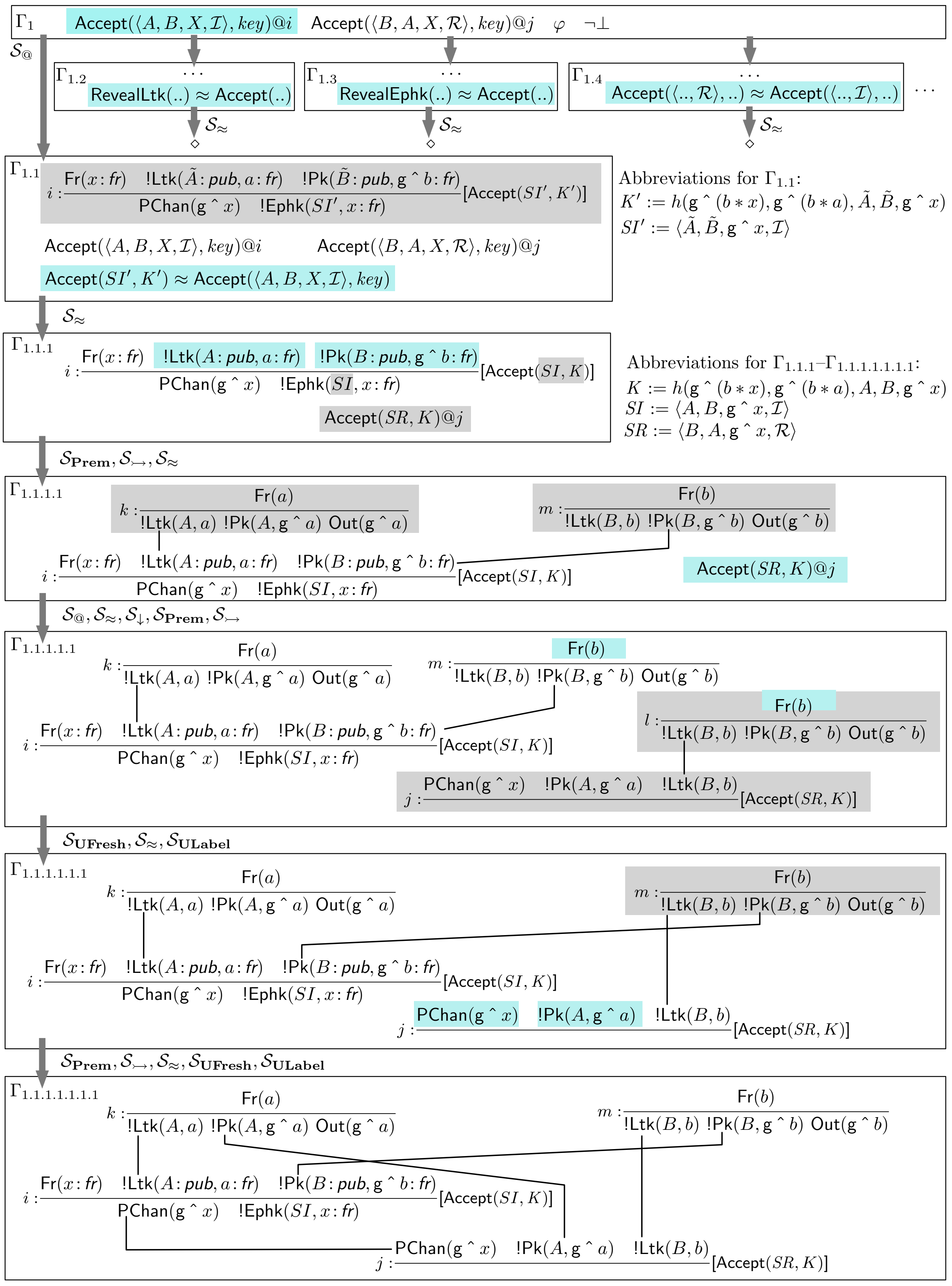

Γ_1 Accept($\langle A,B,X,\mathcal{I}\rangle$, key)@i Accept($\langle B,A,X,\mathcal{R}\rangle$, key)@j φ $\neg\bot$
$\mathcal{S}_@$
$\Gamma_{1.2}$ RevealLtk(..) $\approx$ Accept(..)
$\Gamma_{1.3}$ RevealEphk(..) $\approx$ Accept(..)
$\Gamma_{1.4}$ Accept($\langle .., \mathcal{R}\rangle$, ..) $\approx$ Accept($\langle .., \mathcal{I}\rangle$, ..)
$\mathcal{S}_\approx$
$\Gamma_{1.1}$
Abbreviations for $\Gamma_{1.1}$:
$K' := h(g\,\hat{}\,(b*x), g\,\hat{}\,(b*a), \tilde{A}, \tilde{B}, g\,\hat{}\,x)$
$SI' := \langle \tilde{A}, \tilde{B}, g\,\hat{}\,x, \mathcal{I}\rangle$
Accept(SI', K') $\approx$ Accept($\langle A,B,X,\mathcal{I}\rangle$, key)
$\Gamma_{1.1.1}$
Abbreviations for $\Gamma_{1.1.1}$–$\Gamma_{1.1.1.1.1.1.1.1}$:
$K := h(g\,\hat{}\,(b*x), g\,\hat{}\,(b*a), A, B, g\,\hat{}\,x)$
$SI := \langle A, B, g\,\hat{}\,x, \mathcal{I}\rangle$
$SR := \langle B, A, g\,\hat{}\,x, \mathcal{R}\rangle$
Accept(SR, K)@j
$\mathcal{S}_{\mathbf{Prem}}, \mathcal{S}_\rightarrowtail, \mathcal{S}_\approx$
$\Gamma_{1.1.1.1}$
$\mathcal{S}_@, \mathcal{S}_\approx, \mathcal{S}_\downarrow, \mathcal{S}_{\mathbf{Prem}}, \mathcal{S}_\rightarrowtail$
$\Gamma_{1.1.1.1.1}$
$\mathcal{S}_{\mathbf{UFresh}}, \mathcal{S}_\approx, \mathcal{S}_{\mathbf{ULabel}}$
$\Gamma_{1.1.1.1.1.1}$
$\mathcal{S}_{\mathbf{Prem}}, \mathcal{S}_\rightarrowtail, \mathcal{S}_\approx, \mathcal{S}_{\mathbf{UFresh}}, \mathcal{S}_{\mathbf{ULabel}}$
$\Gamma_{1.1.1.1.1.1.1}$
i : Fr(x : fr) !Ltk(A : pub, a : fr) !Pk(B : pub, g ^ b : fr) / PChan(g ^ x) !Ephk(SI, x : fr) [Accept(SI, K)]
k : Fr(a) / !Ltk(A, a) !Pk(A, g ^ a) Out(g ^ a)
m : Fr(b) / !Ltk(B, b) !Pk(B, g ^ b) Out(g ^ b)
l : Fr(b) / !Ltk(B, b) !Pk(B, g ^ b) Out(g ^ b)
j : PChan(g ^ x) !Pk(A, g ^ a) !Ltk(B, b) [Accept(SR, K)]

Figure 3.12. Constraint reduction for $P_{UM'}$.

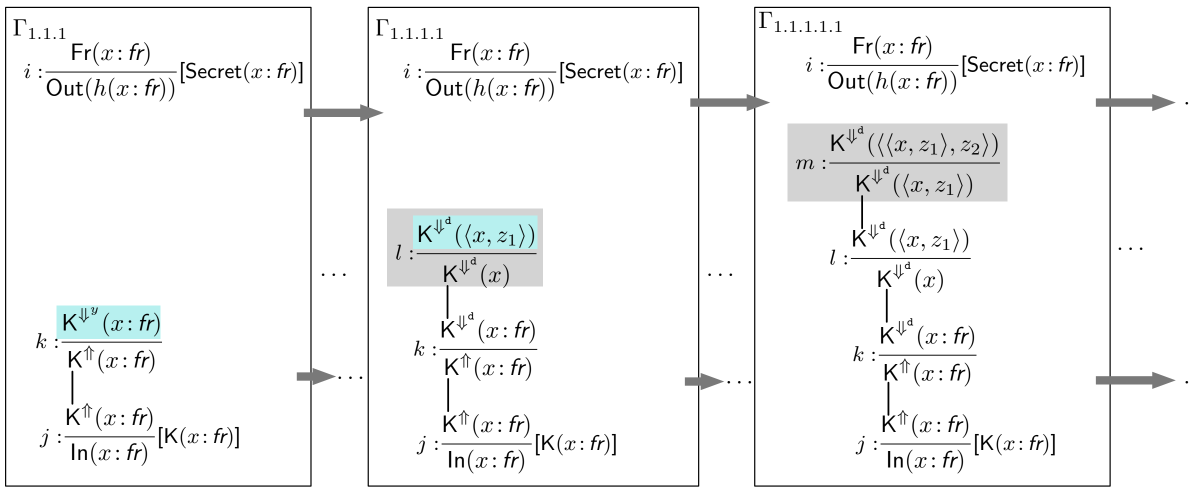
Constraint Solving Rules for Message Deduction

To handle message deduction, we require specialized rules to solve $\mathsf{K}^{\Uparrow}$-premises and $\mathsf{K}^{\Downarrow y}$-premises. Using $\mathcal{S}_{\mathbf{Prem}}$ for these would often result in nontermination, even for simple cases. We illustrate this in the following example.

Example 3.31. Consider the protocol

$$P_{MsgSimp} = \left\{ \frac{\mathsf{Fr}(x{:}fr)}{\mathsf{Out}(h(x))}[\mathsf{Secret}(x)] \right\}.$$

For $\mathcal{ST} = \mathcal{DY}$, we want to show that $\forall i\,x.\mathsf{Secret}(x)@i \Longrightarrow \neg(\exists j.\mathsf{K}(x)@j)$ is valid for $P_{MsgSimp}$. This is equivalent to showing that $\exists i\,j\,x.\mathsf{Secret}(x)@i \wedge \mathsf{K}(x)@j$ is not satisfiable for $P_{MsgSimp}$. After some constraint solving steps where we use $\mathcal{S}_{\mathbf{Prem}}$ also for $\mathsf{K}^{\Uparrow}$-premises, we obtain the constraint system $\Gamma_{1.1.1}$ depicted in Figure 3.14. If we solve the $\mathsf{K}^{\Downarrow y}$-premise of j with $\mathcal{S}_{\mathbf{Prem}}$, one of the resulting cases is shown in $\Gamma_{1.1.1.1}$. Again, we solve the $\mathsf{K}^{\Downarrow \mathsf{d}}$-premise of l with $\mathcal{S}_{\mathbf{Prem}}$ and get a constraint system with another unsolved $\mathsf{K}^{\Downarrow \mathsf{d}}$-premise with a larger pair. We can proceed indefinitely this way without obtaining a contradiction. The problem is that using only a backwards/bottom-up search, we do not take the messages *sent* by the protocol into account. Here, we can see that the protocol never sends a pair and we can therefore never deduce a fact of the form $\mathsf{K}^{\Downarrow \mathsf{d}}(\langle s, t\rangle)$.



$\Gamma_{1.1.1}$
$i : \frac{\mathsf{Fr}(x{:}\,fr)}{\mathsf{Out}(h(x{:}\,fr))}[\mathsf{Secret}(x{:}\,fr)]$
$k : \frac{\mathsf{K}^{\Downarrow y}(x{:}\,fr)}{\mathsf{K}^{\Uparrow}(x{:}\,fr)}$
$j : \frac{\mathsf{K}^{\Uparrow}(x{:}\,fr)}{\mathsf{In}(x{:}\,fr)}[\mathsf{K}(x{:}\,fr)]$
...
$\Gamma_{1.1.1.1}$
$i : \frac{\mathsf{Fr}(x{:}\,fr)}{\mathsf{Out}(h(x{:}\,fr))}[\mathsf{Secret}(x{:}\,fr)]$
$l : \frac{\mathsf{K}^{\Downarrow \mathsf{d}}(\langle x, z_1\rangle)}{\mathsf{K}^{\Downarrow \mathsf{d}}(x)}$
$k : \frac{\mathsf{K}^{\Downarrow \mathsf{d}}(x{:}\,fr)}{\mathsf{K}^{\Uparrow}(x{:}\,fr)}$
$j : \frac{\mathsf{K}^{\Uparrow}(x{:}\,fr)}{\mathsf{In}(x{:}\,fr)}[\mathsf{K}(x{:}\,fr)]$
...
$\Gamma_{1.1.1.1.1}$
$i : \frac{\mathsf{Fr}(x{:}\,fr)}{\mathsf{Out}(h(x{:}\,fr))}[\mathsf{Secret}(x{:}\,fr)]$
$m : \frac{\mathsf{K}^{\Downarrow \mathsf{d}}(\langle\langle x, z_1\rangle, z_2\rangle)}{\mathsf{K}^{\Downarrow \mathsf{d}}(\langle x, z_1\rangle)}$
$l : \frac{\mathsf{K}^{\Downarrow \mathsf{d}}(\langle x, z_1\rangle)}{\mathsf{K}^{\Downarrow \mathsf{d}}(x)}$
$k : \frac{\mathsf{K}^{\Downarrow \mathsf{d}}(x{:}\,fr)}{\mathsf{K}^{\Uparrow}(x{:}\,fr)}$
$j : \frac{\mathsf{K}^{\Uparrow}(x{:}\,fr)}{\mathsf{In}(x{:}\,fr)}[\mathsf{K}(x{:}\,fr)]$
...

Figure 3.14. Looping constraint solving for message deduction.

To prevent the problem explained in Example 3.31, we exploit the properties of normal message deduction. We solve bottom-up from messages received by the protocol with construction rules and top-down from messages sent by the protocol with deconstruction rules until the two meet in the middle with COERCE.

This strategy is formalized by the constraint solving rules given in Figure 3.15. First, note that message deduction usually arises from solving an action $\mathsf{K}(t)@i$ or a premise fact $\mathsf{In}(t)$. Both are solved by adding the corresponding instance of RECV. Then, the message deduction starts by solving the premise $\mathsf{K}^{\Uparrow}(t)$ of RECV. The corresponding rule is $\mathcal{S}_{\mathbf{Prem},\mathsf{K}^{\Uparrow}}$, which adds the provides constraint $j \rhd \mathsf{K}^{\Uparrow}(t)$ and the ordering constraint $j \prec i$

$\mathcal{S}_{\mathbf{Prem},\mathsf{K}^{\Uparrow}}$:	$\dfrac{i\colon ri}{j \rhd \mathsf{K}^{\Uparrow}(m),\, j \lessdot i}\,\mathcal{I}$	if $(prems(ri))_u = \mathsf{K}^{\Uparrow}(m)$ for some u and j freshly chosen
$\mathcal{S}_{\rhd,\mathsf{K}^{\Uparrow}}$:	$\dfrac{i \rhd \mathsf{K}^{\Uparrow}(m)}{i\colon [\mathsf{K}^{\Uparrow}(t_1),\dots,\mathsf{K}^{\Uparrow}(t_k)] -[]\!\!\rightarrow \mathsf{K}^{\Uparrow}(m) \mid i\colon \mathsf{K}^{\Downarrow_y}(m) -[]\!\!\rightarrow \mathsf{K}^{\Uparrow}(m)}\,\mathcal{I}$	if $m = f(t_1,\dots,t_k)$, f not invertible, $f \neq *$, and y freshly chosen
$\mathcal{S}_{\rhd,fr}$:	$\dfrac{i \rhd \mathsf{K}^{\Uparrow}(m)}{i\colon \mathsf{Fr}(m) -[]\!\!\rightarrow \mathsf{K}^{\Uparrow}(m) \mid i\colon \mathsf{K}^{\Downarrow_y}(m) -[]\!\!\rightarrow \mathsf{K}^{\Uparrow}(m)}\,\mathcal{I}$	if m of sort fr and y freshly chosen
$\mathcal{S}_{\rhd,inv}$:	$\dfrac{i \rhd \mathsf{K}^{\Uparrow}(m)}{i\colon [\mathsf{K}^{\Uparrow}(t_1),\dots,\mathsf{K}^{\Uparrow}(t_k)] -[]\!\!\rightarrow \mathsf{K}^{\Uparrow}(m)}\,\mathcal{I}$	if $m = f(t_1,\dots,t_k)$ and f invertible
$\mathcal{S}_{\rhd,*}$:	$\dfrac{i \rhd \mathsf{K}^{\Uparrow}(m)}{j \rhd \mathsf{K}^{\Uparrow}(t),\, j \lessdot i}\,\mathcal{I}$	if $root(m) = *$, $t \in nifactors(m) \setminus \mathcal{V}_{msg}$, and j freshly chosen
$\mathcal{S}_{\mathbf{Prem},\mathsf{K}^{\Downarrow_y}}$:	$\dfrac{i\colon ri}{j \twoheadrightarrow (i,u),\, j\colon \mathsf{Out}(z) -[]\!\!\rightarrow \mathsf{K}^{\Downarrow_{\mathsf{d}}}(z)}\,\mathcal{I}$	if $(prems(ri))_u = \mathsf{K}^{\Downarrow_y}(m)$ for some u and y and j, z freshly chosen
$\mathcal{S}_{\twoheadrightarrow}$:	$\dfrac{i \twoheadrightarrow (k,w),\, i\colon ri,\, \Gamma}{i\colon ri, (i,1) \rightarrowtail (k,w), \Gamma \mid i\colon ri, (i,1) \rightarrowtail (j,1), j\colon ru_1, j \twoheadrightarrow (k,w), \Gamma \mid \dots \mid i\colon ri, (i,1) \rightarrowtail (j,1), j\colon ru_l, j \twoheadrightarrow (k,w), \Gamma}\,\mathcal{M}$	if $\{ru_1,\dots,ru_l\} = ND^{decon}$, $(concs(ri))_1 \neq \mathsf{K}^{\Downarrow_{\mathsf{d}}}(x)$ for all $x \in \mathcal{V}_{msg}$, and j freshly chosen

Figure 3.15. Constraint solving rules for message deduction.

for a fresh temporal variable j, i.e., there must be an earlier node j that provides $\mathsf{K}^{\Uparrow}(t)$. We use provides constraints since we do not represent all construction rules explicitly with a node constraint. For example, we never introduce node constraints for multiplication rules. Then, we have four rules to solve constraints of the form $i \rhd \mathsf{K}^{\Uparrow}(m)$ for different types of messages m. First, $\mathcal{S}_{\rhd,\mathsf{K}^{\Uparrow}}$ handles the case where m is a function application such that the outermost function symbol f is neither invertible nor equal to $*$. The rule performs a case distinction, $\mathsf{K}^{\Uparrow}(m)$ is either the conclusion of the construction rule for f or it is the conclusion of Coerce. In the second case, we use the variable y to represent both e and d. Second, $\mathcal{S}_{\rhd,fr}$ handles the case where m is of sort fr with a similar case distinction. Third, $\mathcal{S}_{\rhd,inv}$ handles the case where the outermost function symbol is invertible. Here, we exploit that m must be the conclusion of the construction rule for f because of **N4**. Finally, $\mathcal{S}_{\rhd,*}$ handles products. We know that i must be a multiplication construction rule because of **N2**. Hence, the premises of i are the non-inverse factors of a ground instance of m in all models. We therefore add new provides constraints for all non-inverse factors of m that cannot be instantiated with products or inverses. Note that if $m \in \mathcal{V}_{msg}$, then no rule is applicable. These constraints can be delayed until m is further instantiated, and they are delayed indefinitely if this never happens.

$$\mathcal{S}_{\mathbf{N3},\Downarrow}\colon\quad \frac{i\colon ps \;\text{-}[]\!\!\rightarrow \mathsf{K}^{\Downarrow y}(m),\, j\colon ps' \;\text{-}[]\!\!\rightarrow \mathsf{K}^{\Downarrow y'}(m)}{i \approx j}\,\mathcal{I} \qquad \mathcal{S}_{\mathbf{N3},\Uparrow}\colon\quad \frac{i \triangleright \mathsf{K}^{\Uparrow}(m),\, j \triangleright \mathsf{K}^{\Uparrow}(m)}{i \approx j}\,\mathcal{I}$$

$$\mathcal{S}_{\mathbf{N5},1}\colon\quad \frac{i\colon ps \;\text{-}[]\!\!\rightarrow \mathsf{K}^{\Downarrow y}(m),\, j \triangleright \mathsf{K}^{\Uparrow}(m)}{i \lessdot j}\,\mathcal{I}$$

$$\mathcal{S}_{\mathbf{N5},2}\colon\quad \frac{i\colon ps \;\text{-}[]\!\!\rightarrow \mathsf{K}^{\Downarrow y}(m),\, j \triangleright \mathsf{K}^{\Uparrow}(m)}{j\colon \mathsf{K}^{\Downarrow y}(m) \;\text{-}[]\!\!\rightarrow \mathsf{K}^{\Uparrow}(m)}\,\mathcal{I} \qquad \text{if } \mathit{root}(m) \text{ not invertible and } m \notin \mathcal{V}_{\mathit{msg}}$$

$$\mathcal{S}_{\mathbf{N6}}\colon\quad \frac{i\colon \left[\mathsf{K}^{\Downarrow \mathsf{d}}(a), \mathsf{K}^{\Uparrow}(b)\right] \;\text{-}[]\!\!\rightarrow \mathsf{K}^{\Downarrow \mathsf{e}}(c\hat{\ }d)}{\bot}\,\mathcal{I} \qquad \text{if } \mathit{nifactors}(d) \subseteq \mathit{nifactors}(b),\ \mathit{vars}(c) \subseteq \mathcal{V}_{\mathit{pub}}, \text{ and } \mathit{St}(c) \cap \mathsf{FN} = \emptyset$$

Figure 3.16. Constraint solving rules that ensure **N3–6**.

The remaining two rules handle the top-down reasoning from sent messages. The rule $\mathcal{S}_{\mathbf{Prem},\mathsf{K}^{\Downarrow y}}$ introduces a chain constraint to solve a $\mathsf{K}^{\Downarrow y}(m)$ premise. The rule $\mathcal{S}_{\twoheadrightarrow}$ solves chain constraints by performing a case distinction: (a) there is an *edge* from the source of the chain to the target of the chain or (b) there is an *edge* from the source of the chain to the first premise of some deconstruction rule and a *chain* from the deconstruction rule to the target of the original chain.

The rules in Figure 3.16 enforce conditions **N3–6**. The first two rules ensure uniqueness of $\mathsf{K}^{\Downarrow y}$-facts and $\mathsf{K}^{\Uparrow}$-facts. The next two rules ensure condition **N5** by adding the required ordering constraint and ensuring that i is an instance of Coerce if the outermost function symbol is not invertible. The last rule enforces **N6**, i.e., exponentiation rules are only allowed if they cannot be directly replaced by the construction rule for exponentiation.

It is not hard to see that all constraint solving rules reduce a constraint system to a finite number of constraint systems because all case distinctions are finite. Consider for example $\mathcal{S}_{\approx}$, then there is one case for each unifier and the set of AC-unifiers of two terms is always finite. Additionally, given a constraint system Γ, there is a finite number of possible constraint solving steps $\Gamma \rightsquigarrow_P \{\Gamma_1, \dots, \Gamma_k\}$ modulo renaming of freshly introduced variables.

Example 3.32. We can now prove that $\forall i\, x. \mathsf{Secret}(x)@i \Longrightarrow \neg(\exists j. \mathsf{K}(x)@j)$ is valid for P_{MsgSimp} from Example 3.31. We prove that the guarded trace property

$$\varphi = \exists i\, x. \mathsf{Secret}(x)@i \wedge (\exists j. \mathsf{K}(x)@j \wedge \neg\bot)$$

is *not* satisfiable for P_{MsgSimp}. We start with the constraint system $\Gamma_0 = \{\varphi\}$ and obtain Γ_1 as shown in Figure 3.17 after simplifying the formula. Note that we overload notation and use edges between $\mathsf{K}^{\Uparrow}$-conclusions and premises to denote ordering constraints $i \lessdot j$ between the corresponding nodes. To obtain $\Gamma_{1.1}$ from Γ_1, we solve the two actions. Next, we solve the premise $\mathsf{K}^{\Uparrow}(x{:}\mathit{fr})$ which results in two cases. The first case is depicted in $\Gamma_{1.1.1}$ where the premise is provided by an instance of the construction rule for fresh names. Here, i and k have the same Fr-premise and we can therefore obtain $i \approx k$ using $\mathcal{S}_{\mathbf{UFresh}}$ which we solve with $\mathcal{S}_{\approx}$. Then we use $\mathcal{S}_{\mathbf{ULabel}}$ to obtain a contradiction since the two rule instances have no unifier. The second case is depicted in $\Gamma_{1.1.2}$ where $\mathsf{K}^{\Uparrow}(x{:}\mathit{fr})$ is the conclusion of a Coerce instance with premise $\mathsf{K}^{\Downarrow y}(x{:}\mathit{fr})$. We proceed by solving this premise with $\mathcal{S}_{\mathbf{Prem},\mathsf{K}^{\Downarrow y}}$ which adds a fresh instance $l\colon \mathsf{Out}(z) \;\text{-}[]\!\!\rightarrow \mathsf{K}^{\Downarrow \mathsf{d}}(z)$ of Recv and a chain constraint from l to $(k, 1)$.

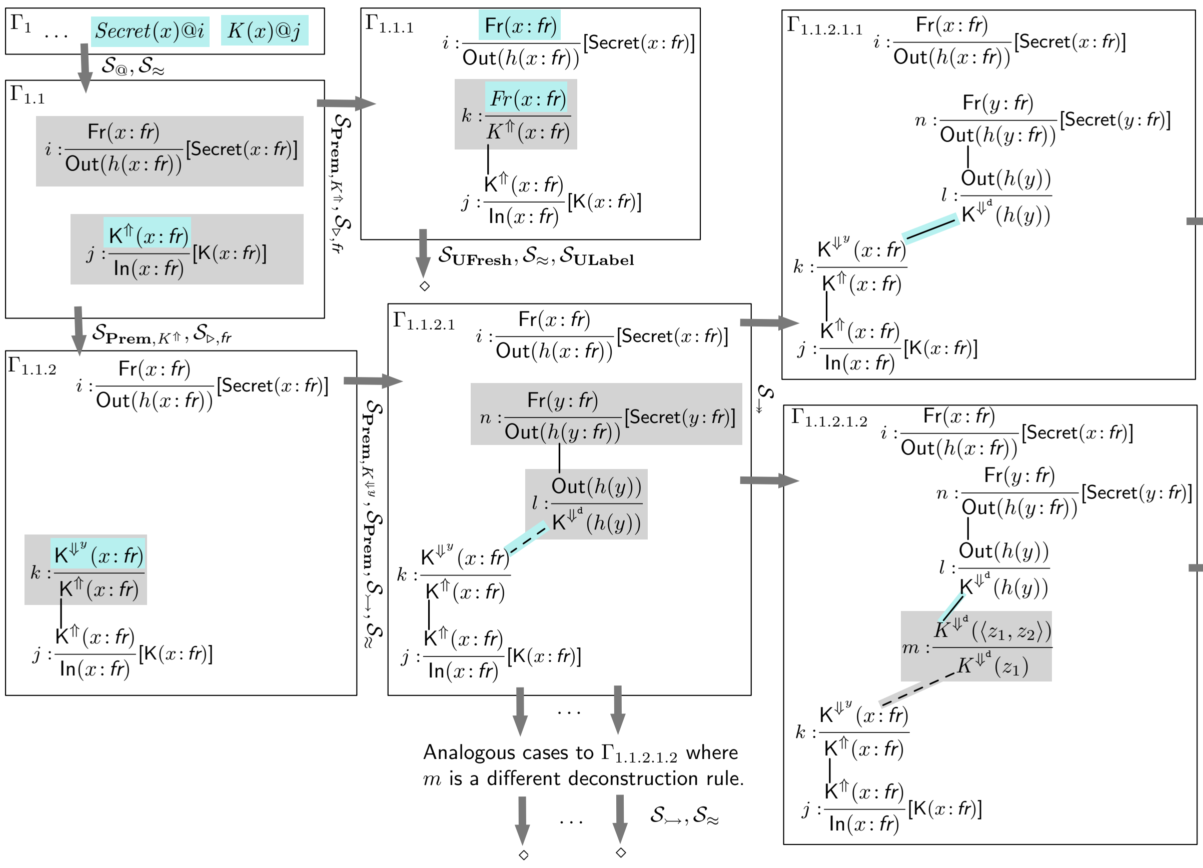

Γ_1
$\mathit{Secret}(x)@i$
$K(x)@j$
$\mathcal{S}_@, \mathcal{S}_\approx$
$\Gamma_{1.1}$
$\mathcal{S}_{\mathbf{Prem},K^\Uparrow}, \mathcal{S}_{\triangleright,fr}$
$\Gamma_{1.1.1}$
$\mathcal{S}_{\mathbf{UFresh}}, \mathcal{S}_\approx, \mathcal{S}_{\mathbf{ULabel}}$
$\Gamma_{1.1.2}$
$\mathcal{S}_{\mathbf{Prem},K^{\Downarrow y}}, \mathcal{S}_{\mathbf{Prem}}, \mathcal{S}_\rightarrowtail, \mathcal{S}_\approx$
$\Gamma_{1.1.2.1}$
$\mathcal{S}_\rightarrow$
$\Gamma_{1.1.2.1.1}$
$\Gamma_{1.1.2.1.2}$
$\mathcal{S}_\rightarrowtail, \mathcal{S}_\approx$
Analogous cases to $\Gamma_{1.1.2.1.2}$ where m is a different deconstruction rule.
$\mathcal{S}_\rightarrowtail, \mathcal{S}_\approx$

Figure 3.17. Constraint reduction that proves the validity of $\forall i\,x.\mathsf{Secret}(x)@i \Longrightarrow \neg(\exists j.\mathsf{K}(x)@j)$ for $P_{MsgSimp}$.

Solving the Out-premise of l yields the one case shown in $\Gamma_{1.1.2.1}$. Next, we solve the chain constraint. This results in the constraint system $\Gamma_{1.1.2.1.1}$ for the case where there is an edge between $(l,1)$ and $(k,1)$, constraint system $\Gamma_{1.1.2.1.2}$ for the case where the deconstruction rule for *fst* is applied to $\mathsf{K}^{\Downarrow^d}(h(y))$, and additional cases for the other deconstruction rules. In all cases, we can use $\mathcal{S}_\rightarrowtail$ and $\mathcal{S}_\approx$ to show that these constraint systems are contradictory. For $\Gamma_{1.1.2.1.1}$, there is no unifier for $h(y)$ and $x{:}fr$. For $\Gamma_{1.1.2.1.2}$, there is no unifier for $h(y)$ and $\langle z_1, z_2\rangle$. For the remaining constraint systems, there is no unifier for $h(y)$ and the message of the first premise of the added deconstruction rule. By reducing $\{\varphi\}$ to the empty set of constraint systems, we have proved that φ is *not* satisfiable for $P_{MsgSimp}$.

Properties of the Constraint Solving Relation

We now formally state and prove two important properties of $\rightsquigarrow_P$. First, the relation is sound and complete, which means that the set of P-models does not change when we solve constraints. Second, under certain well-formedness conditions, we can extract a P-model from a solved constraint system. Note that all constraint systems that occur during analysis are well-formed.

Theorem 3.33. *The constraint solving relation $\rightsquigarrow_P$ is sound and complete, i.e., for every $\Gamma \rightsquigarrow_P \{\Gamma_1, \dots, \Gamma_k\}$, $models_P(\Gamma) = \bigcup_{i=1}^k models_P(\Gamma_i)$.*

The proof of this theorem can be found in Appendix A.2. We extend the notion of redundant to the constraint solving rules from Figure 3.15 and 3.16 as follows.

Definition 3.34. *An application of a constraint solving rule is redundant if it satisfies Definition 3.29 or one of the following holds:*

- *The rule is $\mathcal{S}_{\triangleright,\mathsf{K}^{\Uparrow}}$, $\mathcal{S}_{\triangleright,fr}$, or $\mathcal{S}_{\triangleright,\mathrm{inv}}$ and there is a node constraint $i\colon ps \dashv[]\mapsto \mathsf{K}^{\Uparrow}(m)$.*
- *The rule is $\mathcal{S}_{N5,2}$ and the conclusion is already in Γ for $y = \mathtt{e}$ or $y = \mathtt{d}$.*

We now define well-formed constraint systems. Well-formedness is required for the extraction of models. For a guarded trace property φ, the constraint system $\{\varphi\}$ is well-formed and all constraint solving steps preserve well-formedness.

Definition 3.35. A constraint system is well-formed for a protocol P if the following conditions hold.

WF1. All node constraints are AC-instances of rules from $\lceil P \rceil^{\mathcal{RDH}_e}_{insts}$ or rules from ND except for the multiplication rules.

WF2. For all node constraints of the form i: $ps \dashv[]\mapsto \mathsf{K}^{\Uparrow}(m)$, there is also a provides constraint $i \triangleright \mathsf{K}^{\Uparrow}(m')$ with $m =_{AC} m'$.

WF3. All provides constraints are of the form $i \triangleright \mathsf{K}^{\Uparrow}(m)$.

WF4. For all edge constraints $(i, u) \rightarrowtail (j, v)$, there are node constraints $i\colon ri$ and $j\colon ru$ such that u and v are valid conclusion and premise indices, respectively.

WF5. For all chain constraints $i \twoheadrightarrow (j, v)$, there are node constraints $i\colon ri$ and $j\colon ru$ such that $(concs(ri))_1$ and $(prems(ru))_v$ are of the form $\mathsf{K}^{\Downarrow y}(t)$ for some y and t.

WF6. There are no fresh names in Γ.

Theorem 3.36. *Every well-formed constraint system Γ for P that is solved with respect to $\rightsquigarrow_P$ has at least one P-model.*

Proof. (Sketch) The model extraction follows the method described in Example 3.30. Note that for message deduction, there are constraints $i \triangleright \mathsf{K}^{\Uparrow}(x)$ for $x \in \mathcal{V}_{msg}$ such that there is no node constraint for i. We solve these by instantiating message variables with distinct fresh names and adding the corresponding instances of the construction rule for fresh names. A full proof for the model extraction for a slightly modified set of rules can be found in [157]. □

3.3.4 Optimizations for Constraint Solving

There are two cases where *directly* using the constraint solving rules leads to unnecessary case distinctions. First, the $\mathcal{S}_{\mathbf{Prem}}$ rule introduces one case for each variant of a protocol rule. In many proofs, these case distinctions are not required at all or can be delayed since the variants are similar enough to jointly deal with all cases. To prevent these case distinctions, we encode the variants of a rule as a pair of a single rule and a disjunction of equalities. This allows us to delay the case distinctions stemming from the different variants. Second, the constraint solving rules are not specialized for the protocol that is analyzed. This often leads to situations where the same constraint solving sequence is repeatedly used.

As an example, consider the protocol $P_{MsgSimp}$ which sends only terms of the form $h(t)$. Solving constraints $i \triangleright \mathsf{K}^{\Uparrow}(x{:}fr)$ with $\mathcal{S}_{\triangleright,fr}$ results in two cases. One where the conclusion is provided by the construction rule for fresh names and another where it is provided by COERCE. It is easy to see that the COERCE case always leads to a contradiction since it is impossible to extract a name from a hash. For $P_{MsgSimp}$, the rule $\mathcal{S}_{\triangleright,fr}$ can therefore be specialized by removing the COERCE case. To achieve this, we use constraint solving to compute derived constraint rewriting rules for a protocol P. When analyzing P, we can then use these derived constraint rewriting rules to save work. We explain these two optimizations in the following subsections.

Lazy Case Distinctions for Variants

We use the following technique to delay the case distinctions stemming from different variants of the same protocol rule. We say a pair (ru', ψ) of a rule ru' and a formula ψ is a *complete* $\mathcal{RDH}_e, AC$*-variant-formula* of a rule ru if

$$ginsts_{\mathcal{DH}_e}(ru){\downarrow_{\mathcal{RDH}_e}} =_{AC} \{ri \,|\, \exists\theta.\, ri =_{AC} ru'\theta \wedge ([\,],\theta) \vDash_{AC} \psi \wedge ri \ {\downarrow_{\mathcal{RDH}_e}}\text{-normal}\}.$$

This means that the rule ru' and the formula ψ characterize all ${\downarrow_{\mathcal{RDH}_e}}$-normal $\mathcal{DH}_e$-instances of ru. We can compute a complete variant-formula (ru', ψ) of a rule ru as follows. Let $\{t_1, \ldots, t_k\}$ denote the set of terms that occur as direct subterms of facts in ru and

$$\lceil \langle t_1, \ldots, t_k \rangle \rceil^{\mathcal{RDH}_e}_{insts} = \{\langle s^1_1, \ldots s^1_k \rangle, \ldots, \langle s^n_1, \ldots s^n_k \rangle\}.$$

Then, we obtain ru' from ru by replacing all terms t_i with distinct variables x_i and define the formula as

$$\psi = \bigvee_{j=1}^{n} \left(\exists\, \vec{y_j}.x_1 \approx s^j_1 \wedge \ldots \wedge x_k \approx s^j_k\right),$$

where $\vec{y_j} = vars(s^j_1, \ldots s^j_k)$. Afterwards, we can simplify ru' and ψ. For example, if $x_i \approx t$ is included in all disjuncts, we can instantiate x_i with t in ru' and remove the equality.

To take advantage of variant-formulas, we define a modified constraint solving rule $\mathcal{S}_{\mathbf{Prem}'}$ that uses variant-formulas instead of variants. To define this rule, we assume given an algorithm cvf that computes a complete $\mathcal{RDH}_e$,AC-variant-formula of a rule ru. The modified rule $\mathcal{S}_{\mathbf{Prem}'}$ and a similarly modified rule $\mathcal{S}_{@'}$ are then defined as follows.

$$\mathcal{S}_{\mathbf{Prem}'}: \quad \frac{i{:}\, ri}{\begin{array}{l} j{:}\, ru_1, \psi_1, (j, v_1) \rightarrowtail (i, u) \\ \mid\ \ldots \\ \mid\ j{:}\, ru_l, \psi_l, (j, v_l) \rightarrowtail (i, u) \end{array}}\,\mathcal{I} \qquad \begin{array}{l} \text{if } u \in idx(prems(ri)), \\ (prems(ri))_u \text{ not } \mathsf{K}^d \text{ or } \mathsf{Fr} \text{ fact}, \\ \{(ru_1, \psi_1, v_1), \ldots, (ru_l, \psi_l, v_l)\} = \\ \{(ru', \psi, v) \,|\, ru \in P \cup \{\textsc{Send}\} \\ \qquad\qquad \wedge (ru', \psi) = cvf(ru) \\ \qquad\qquad \wedge v \in idx(concs(ru'))\}, \\ \text{and } j \text{ freshly chosen} \end{array}$$

$$\mathcal{S}_{@'}: \quad \frac{f@i}{i{:}\, ru_1, g_1 \approx f, \psi_1 | \ldots | i{:}\, ru_l, g_l \approx f, \psi_l}\,\mathcal{I} \qquad \begin{array}{l} \text{if } \{(ru_1, g_1, \psi_i), \ldots, (ru_l, g_l, \psi_l)\} = \\ \{(ru', g, \psi) \,|\, ru \in P \cup \{\textsc{Send}\} \\ \qquad\qquad \wedge (ru', \psi) = cvf(ru) \\ \qquad\qquad \wedge g \in acts(ru')\} \end{array}$$

Note that the formula ψ is not always a guarded trace formula since it can contain quantification over variables of sorts other than *msg*. We introduced the restriction to support model extraction, but it is only required for *universally quantified* variables and can be dropped for *existentially quantified* variables. The second issue is that s_i^j can contain arbitrary function symbols. This is not a problem since the restriction is only required for switching from satisfaction with respect to $\vDash_{\mathcal{DH}_e}$ to satisfaction with respect to $\vDash_{AC}$. It is irrelevant for formulas that occur in constraint systems since $\Vdash$ already uses $\vDash_{AC}$ to define formula semantics.

Example 3.37. As an example, consider the rule $\mathsf{In}(z) \,\text{-[]}\!\!\rightarrow \mathsf{Out}(h(\mathit{fst}(z)))$. Then

$$(\mathsf{In}(x_1) \,\text{-[]}\!\!\rightarrow \mathsf{Out}(x_2), ((\exists z. x_1 \approx z \wedge x_2 \approx h(\mathit{fst}(z))) \vee (\exists u v. x_1 \approx \langle u, v\rangle \wedge x_2 \approx h(u))))$$

is a complete $\mathcal{RDH}_e$,AC-variant-formula of the original rule. Since $x_2 \approx h(t_1)$ in the first disjunct and $x_2 \approx h(t_2)$ in the second disjunct, we can simplify the rule and formula to

$$(\mathsf{In}(x_1) \,\text{-[]}\!\!\rightarrow \mathsf{Out}(h(x_2)), ((\exists z. x_1 \approx z \wedge x_2 \approx \mathit{fst}(z)) \vee (\exists u v. x_1 \approx \langle u, v\rangle \wedge x_2 \approx u))).$$

It is easy to see that using $\mathcal{S}_{\mathbf{Prem}}$ is equivalent to using $\mathcal{S}_{\mathbf{Prem}'}$ followed by $\mathcal{S}_{\vee}$, $\mathcal{S}_{\exists}$, and $\mathcal{S}_{\approx}$. Since the node constraint added by $\mathcal{S}_{\mathbf{Prem}'}$ often contains enough information to continue without solving the added disjunction, the additional control offered by $\mathcal{S}_{\mathbf{Prem}'}$ often allows to delay the use of $\mathcal{S}_{\vee}$.

Example 3.38. For P_{UM}, $\mathit{cvf}(ru) = (ru{\downarrow_{\mathcal{RDH}_e}}, \neg\bot)$ for all rules except for the responder rule. For the responder rule, we obtain (ru', ψ) for

$$ru' = \frac{\mathsf{In}(X) \quad !\mathsf{Pk}(A{:}\mathit{pub}, \mathsf{g}\hat{\ }(a{:}\mathit{fr})) \quad !\mathsf{Ltk}(B{:}\mathit{pub}, b{:}\mathit{fr})}{}[\mathsf{Accept}(\mathit{sid}, \mathit{key})]$$

$$\text{where } \mathit{sid} = \langle B, A, X, \mathcal{R}\rangle \qquad \mathit{key} = h(z, \mathsf{g}\hat{\ }(a{*}b), A, B, X)$$

and

$$\begin{aligned}\psi = \quad & (z \approx X\hat{\ }b)\\ & \vee(\exists\, u v. X \approx u\hat{\ }v \wedge z \approx u\hat{\ }(v{*}b))\\ & \vee(\exists\, u. X \approx u\hat{\ }(b^{-1}) \wedge z \approx u)\\ & \vee(\exists\, u\, v. X \approx u\hat{\ }(b^{-1}{*}v) \wedge z \approx u\hat{\ }v)\\ & \vee(\exists\, u v. X \approx u\hat{\ }(b{*}v)^{-1} \wedge z \approx u\hat{\ }(v^{-1}))\\ & \vee(\exists\, u v w. X \approx u\hat{\ }((b{*}v)^{-1}{*}w) \wedge z \approx u\hat{\ }(v^{-1}{*}w)).\end{aligned}$$

In Example 3.30, solving Γ_1 with $\mathcal{S}_{@'}$ results in one case for the responder instead of six cases, and this case can be shown to be contradictory with $\mathcal{S}_{\approx}$ without applying $\mathcal{S}_{\vee}$ to the variant-formula.

Derived Constraint Rewriting Rules

A *constraint rewriting rule* is a pair $(\Omega, \mathbf{\Delta})$ written $\Omega \hookrightarrow \mathbf{\Delta}$ where Ω is a constraint system and $\mathbf{\Delta}$ is a set of constraint systems. We say a constraint rewriting rule $\Omega \hookrightarrow \mathbf{\Delta}$ is *context-sound and context-complete for the protocol P* if for all Γ such that $\mathit{vars}(\Gamma) \cap (\mathit{vars}(\mathbf{\Delta}) \setminus \mathit{vars}(\Omega)) = \emptyset$, it holds that $\mathit{models}_P(\Omega \cup \Gamma) = \bigcup_{\Delta\in\mathbf{\Delta}} \mathit{models}_P(\Delta \cup \Gamma)$. Informally, this means that the constraint rewriting rule is sound and complete for all contexts Γ whose variables do not clash with freshly chosen variables in the rule.

A constraint rewriting rule $\Omega \hookrightarrow \mathbf{\Delta}$ can be applied to a constraint system Γ by performing the following steps.

1. Rename $\Omega \hookrightarrow \mathbf{\Delta}$ away from Γ to obtain $\Omega' \hookrightarrow \mathbf{\Delta}'$.
2. Find a substitution σ with $dom(\sigma) \subseteq vars(\Omega')$ such that $\Gamma =_{AC} (\Omega'\sigma \cup \Gamma')$ for some Γ', e.g., using matching modulo AC and the equational theory for sets.
3. Compute the result of the rule application as $\{\Delta\sigma \cup \Gamma' | \Delta \in \mathbf{\Delta}'\}$.

We call the pair $(\Gamma, \{\Delta\sigma \cup \Gamma' | \Delta \in \mathbf{\Delta}'\})$ an *application of* $\Omega \hookrightarrow \mathbf{\Delta}$ *to* Γ. Intuitively, we perform set rewriting of Γ with the rule $\Omega \hookrightarrow \mathbf{\Delta}$ and ensure that the freshly chosen variables in $\mathbf{\Delta}$ do not clash with the variables in Γ.

Lemma 3.39. *Let* $\Omega \hookrightarrow \mathbf{\Delta}$ *be an arbitrary constraint rewriting rule that is context-sound and context-complete for* P *and* $(\Gamma, \mathbf{\Theta})$ *an application of this rule to* Γ. *Then*

$$models_P(\Gamma) = \bigcup_{\Theta \in \mathbf{\Theta}} models_P(\Theta).$$

Example 3.40. The constraint rewriting rule

$$\{i \rhd \mathsf{K}^{\Uparrow}(x{:}fr)\} \hookrightarrow \{\{i \rhd \mathsf{K}^{\Uparrow}(x{:}fr), i{:}\, \mathsf{Fr}(x{:}fr) \dashv[]\!\!\mapsto \mathsf{K}^{\Uparrow}(x{:}fr)\}\}$$

is context-sound and context-complete for $P_{MsgSimp}$ and can be applied to the constraint system

$$\Gamma = \{k \rhd \mathsf{K}^{\Uparrow}(y{:}fr), j{:}\, \mathsf{Fr}(y{:}fr) \dashv[\mathsf{Secret}(y{:}fr)]\!\!\mapsto \mathsf{Out}(h(y{:}fr))\}$$

as follows. First note that the constraint rewriting rule is already renamed away from Γ. For the substitution $\sigma = \{y{:}fr/x{:}fr, k/i\}$ and $\Gamma' = \{j{:}\, \mathsf{Fr}(y{:}fr) \dashv[\mathsf{Secret}(y{:}fr)]\!\!\mapsto \mathsf{Out}(h(y{:}fr))\}$, $\Gamma =_{AC} (\{i \rhd \mathsf{K}^{\Uparrow}(x{:}fr)\}\sigma \cup \Gamma')$. The result of the rule application is therefore

$$\{k \rhd \mathsf{K}^{\Uparrow}(y{:}fr), k{:}\, \mathsf{Fr}(y{:}fr) \dashv[]\!\!\mapsto \mathsf{K}^{\Uparrow}(y{:}fr), j{:}\, \mathsf{Fr}(y{:}fr) \dashv[\mathsf{Secret}(y{:}fr)]\!\!\mapsto \mathsf{Out}(h(y{:}fr))\}.$$

By using the constraint rewriting rule for $i \rhd \mathsf{K}^{\Uparrow}(x{:}fr)$, we have replaced most of the reasoning from Figure 3.17 with one step. This shows that derived constraint rewriting rules simplify constraint solving considerably. This is similar to using lemmas in proofs.

We have now seen how to use a context-sound and context-complete constraint rewriting rule. The remaining question is how to compute such a rule for a given constraint system Ω. We know $\rightsquigarrow_P$ is sound and complete for P. But what about context-soundness and context-completeness for P? For all constraint solving rules except for $\mathcal{S}_{\approx}$, it not hard to see that $\Omega \rightsquigarrow_P \mathbf{\Delta}$ implies that $\Omega \hookrightarrow \mathbf{\Delta}$ is a context-sound and context-complete constraint rewriting rule for P. This also holds for $\rightsquigarrow_P^*$ if the constraint reduction does not use applications of $\mathcal{S}_{\approx}$. To see why $\mathcal{S}_{\approx}$ does not have this property, consider the following example.

Example 3.41. Let a be a fresh name and $x \in \mathcal{V}_{msg}$. Then $\{x \approx \mathsf{a}\} \rightsquigarrow_P \{\mathsf{a} \approx \mathsf{a}\}$, but $\{x \approx \mathsf{a}\} \hookrightarrow \{\mathsf{a} \approx \mathsf{a}\}$ is not context-sound for P. Consider the context $\{x \approx \mathsf{b}\}$ where b is a public name different from a. Then

$$\emptyset = models_P(\{x \approx \mathsf{a}, x \approx \mathsf{b}\}) \neq models_P(\{\mathsf{a} \approx \mathsf{a}, x \approx \mathsf{b}\}).$$

The problem is that the link between the variable x in the initial constraint system and its instantiation a gets lost when applying $\mathcal{S}_{\approx}$.

We obtain a constraint solving relation $\rightsquigarrow_P^{\mathrm{ctx}}$ such that $\Omega \rightsquigarrow_P^{\mathrm{ctx}*} \mathbf{\Delta}$ implies that $\Omega \hookrightarrow \mathbf{\Delta}$ is context-sound and context-complete for P in two steps.

First, we define the constraint solving rule $\mathcal{S}_{\approx'}$ which keeps track of all substitution applications by adding the corresponding equalities. To define this rule, we use $\mathit{eqs}(\sigma)$ to denote the set $\{x_1 \approx \sigma(x_1), \ldots, x_k \approx \sigma(x_k)\}$ where $\{x_1, \ldots, x_k\} = \mathit{dom}(\sigma)$.

$$\boxed{\mathcal{S}_{\approx'}\colon \quad \frac{s \approx t, \Gamma}{\Gamma\sigma_1, \mathit{eqs}(\sigma_1) \mid \ldots \mid \Gamma\sigma_l, \mathit{eqs}(\sigma_l)}\mathcal{M} \qquad \text{if } \{\sigma_1, \ldots, \sigma_l\} = \mathit{unif}_{AC}^{\mathit{vars}(\Gamma)}(s,t)}$$

The added equalities keep track of the instantiations of variables that occur in the initial constraint system Ω and prevent the described problem.

Second, we define $\rightsquigarrow_P^{\mathrm{ctx}}$ as the result of replacing $\mathcal{S}_{\approx}$ by $\mathcal{S}_{\approx'}$ in $\rightsquigarrow_P$. Note that constraint rewriting rules resulting from $\rightsquigarrow_P^{\mathrm{ctx}*}$ also maintain the well-formedness of constraint systems.

Example 3.42. For the protocol P_{UM}, we use $\rightsquigarrow_P^{\mathrm{ctx}*}$ to compute constraint rewriting rules for the initial constraint systems $\{i \rhd \mathsf{K}^{\Uparrow}(h(u))\}$, $\{i \rhd \mathsf{K}^{\Uparrow}(x{:}\mathit{fr})\}$, and $\{i \rhd \mathsf{K}^{\Uparrow}(u\hat{\ }v)\}$, where $u, v \in \mathcal{V}_{\mathit{msg}}$. For $\{i \rhd \mathsf{K}^{\Uparrow}(h(u))\}$, we first apply $\mathcal{S}_{\rhd,\mathsf{K}^{\Uparrow}}$ to $i \rhd \mathsf{K}^{\Uparrow}(h(u))$, which results in two cases: $\mathsf{K}^{\Uparrow}(h(u))$ is the conclusion of the construction rule for h or the conclusion of Coerce. In the first case, the resulting constraint system is already solved. In the second case, we proceed by solving the $\mathsf{K}^{\Downarrow y}(h(u))$ premise of Coerce. This introduces a chain constraint starting at a fresh instance of the Recv rule. By solving the Out-premise of Recv and repeatedly solving the chain constraint, we can show that this case is contradictory. This was expected since no protocol rule sends a term that contains a hash and it is therefore impossible to extract a hash from a sent message. The result of the computation is hence

$$\mathcal{R}_{\mathrm{hash}} = \{i \rhd \mathsf{K}^{\Uparrow}(h(u))\} \hookrightarrow \{\{i \rhd \mathsf{K}^{\Uparrow}(h(u)), i\colon \mathsf{K}^{\Uparrow}(u) -[]\!\!\rightarrow \mathsf{K}^{\Uparrow}(h(u))\}\}.$$

In this case, it was possible to obtain only solved or contradictory constraint systems. In general, we often stop the computation when the initial constraint and all introduced chain constraints are solved. Continuing further might lead to nontermination or constraint rewriting rules with too many cases. Note that it might not always be possible to get rid of all chain constraints, e.g., for protocols that act like decryption oracles. We discuss this issue in Section 3.5.5.

For exponentiations, we compute the rule

$$\mathcal{R}_{\mathrm{exp}} = \{i \rhd \mathsf{K}^{\Uparrow}(u\hat{\ }v)\} \hookrightarrow \{\Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5\}$$

depicted in Figure 3.18. This rule states that there are five ways to deduce an exponentiation for the adversary. First, he can learn the ephemeral public key of a protocol session. Second, he can learn the long-term public key of an agent. Third, he can add additional exponents to the ephemeral public key of a protocol session. Fourth, he can add additional exponents to the long-term public key of an agent. Finally, he can construct an exponentiation $u\hat{\ }v$ himself. Here, u cannot be instantiated with an exponentiation since the resulting term would not be $\downarrow_{\mathcal{RDH}_e}$-normal and $\mathcal{S}_{\downarrow}$ would be applicable. Repeated

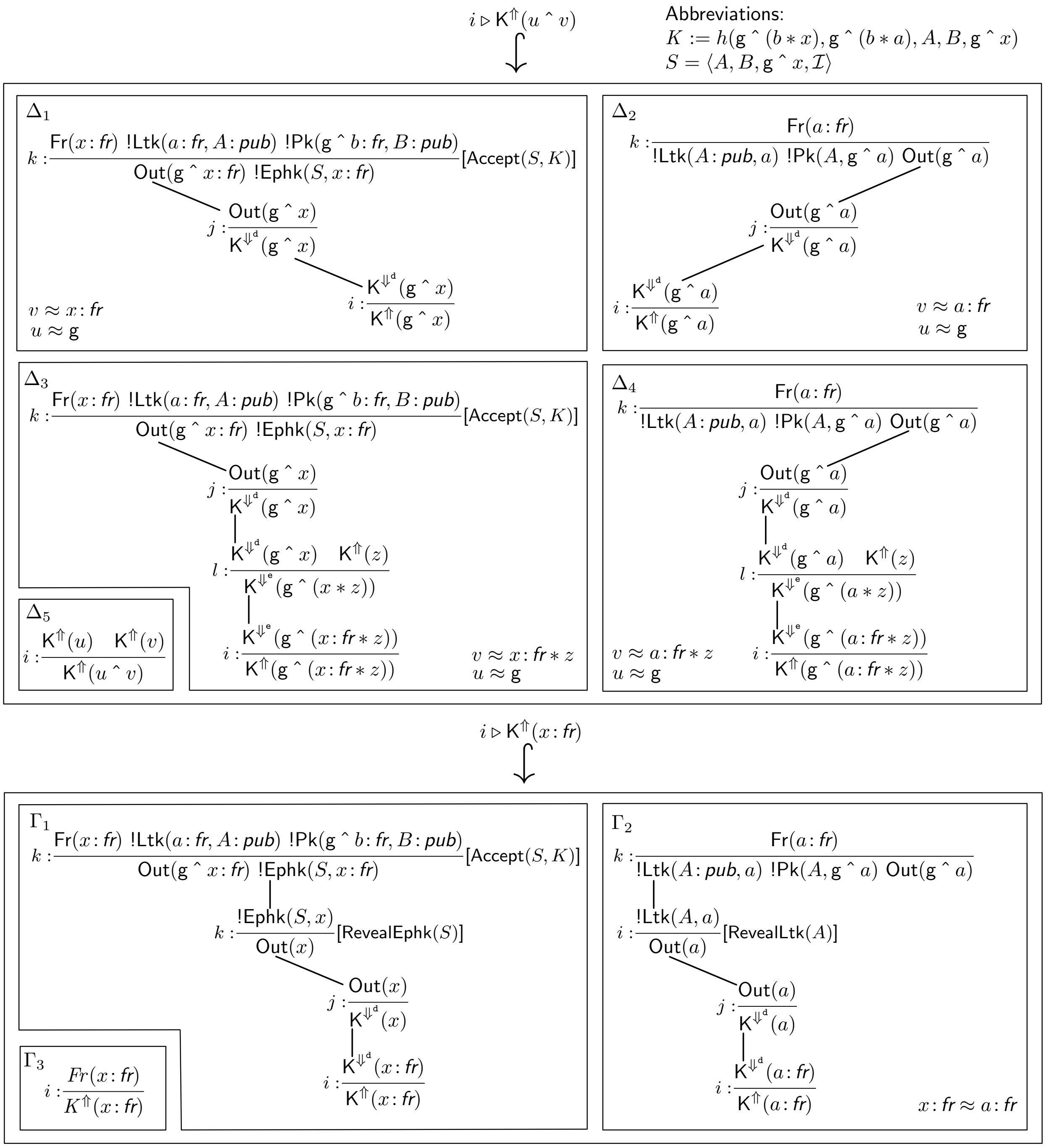


Figure 3.18. Constraint rewriting rules from Example 3.42.

exponentiation by the adversary is modeled by instantiating v with a product. In this case, the adversary knows all (non-inverse) factors of the exponent, which can exploited by $\mathcal{S}_{\triangleright,*}$ during constraint solving.

For fresh names, we compute the rule

$$\mathcal{R}_{\text{fresh}} = \{i \triangleright \mathsf{K}^{\Uparrow}(x{:}fr)\} \hookrightarrow \{\Gamma_1, \Gamma_2, \Gamma_3\}$$

depicted in Figure 3.18. This rule states that the adversary can deduce a fresh name by using ephemeral key reveal, long-term key reveal, or constructing a fresh name himself.

3.3.5 One-pass UM: Security proof

We now demonstrate how the constraint solving relation and the optimizations can be used to prove the security of UM. We want to verify that $\varphi_{\mathit{UM\text{-}sec}}$ from Figure 3.3 is valid for P_{UM} from Figure 3.2. To achieve this, we check whether $\neg\varphi_{\mathit{UM\text{-}sec}}$ is satisfiable using constraint solving. We therefore start with the constraint system $\Gamma_0 = \{\hat{\varphi}\}$ where $\hat{\varphi}$ is the result of rewriting $\neg\varphi_{\mathit{UM\text{-}sec}}$ into a guarded trace property, i.e.,

$$\begin{aligned}\hat{\varphi} = \exists i\, A\, B\, X\, \mathit{key}.&\\ &\mathsf{Accept}(\langle B, A, X, \mathcal{R}\rangle, \mathit{key})@i \wedge (\exists j.\, \mathsf{K}(\mathit{key})@j \wedge \neg\bot)\\ &\wedge(\forall k. \neg\mathsf{RevealLtk}(B)@k \vee \bot)\\ &\wedge(\forall k. \neg\mathsf{RevealLtk}(A)@k \vee (\forall l. \neg\mathsf{RevealEphk}(\langle A, B, X, \mathcal{I}\rangle)@l \vee \bot))\\ &\wedge((\exists k\, \mathit{key}'.\mathsf{Accept}(\langle A, B, X, \mathcal{I}\rangle, \mathit{key}')@k \wedge \neg\bot) \vee (\forall k. \neg\mathsf{RevealLtk}(A)@k \vee \bot))\,.\end{aligned}$$

After simplifying the formula and ignoring solved formulas, we obtain the constraint system

$$\Gamma_1 = \left\{\begin{array}{l}\mathsf{Accept}(\langle B, A, X, \mathcal{R}\rangle, \mathit{key})@i \quad \mathsf{K}(\mathit{key})@j\\ \forall k. \neg\mathsf{RevealLtk}(B)@k \vee \bot\\ (\forall k. \neg\mathsf{RevealLtk}(A)@k) \vee (\forall l. \neg\mathsf{RevealEphk}(\langle A, B, X, \mathcal{I}\rangle)@l \vee \bot)\\ (\exists k\, \mathit{key}'.\mathsf{Accept}(\langle A, B, X, \mathcal{I}\rangle, \mathit{key}')@k \wedge \neg\bot) \vee (\forall k. \neg\mathsf{RevealLtk}(A)@k \vee \bot)\end{array}\right\}.$$

In the first step, we apply $\mathcal{S}_\vee$ to the last formula, simplify the result, and obtain

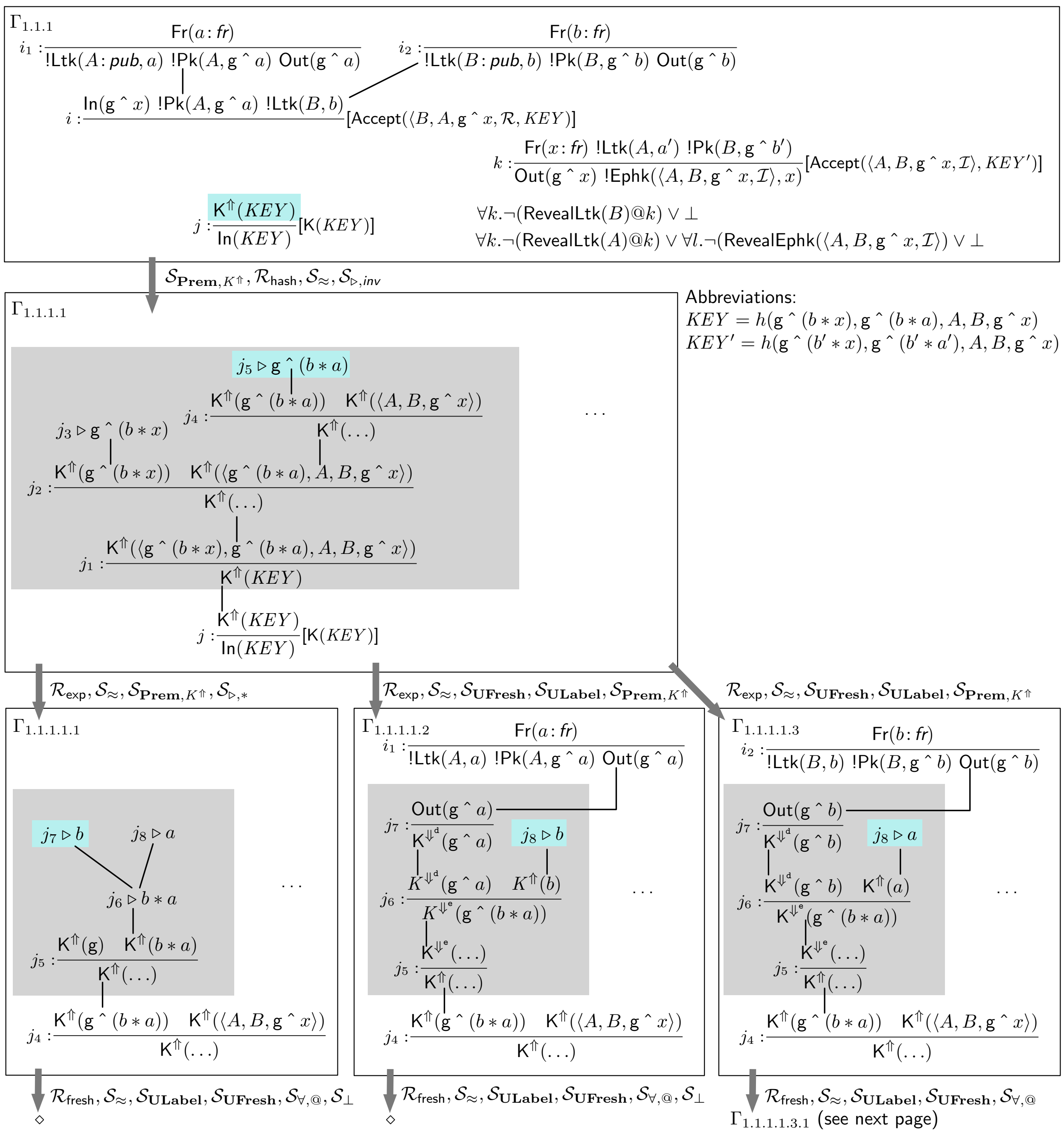
$$\begin{aligned}\Gamma_{1.1} &= \Gamma_1 \cup \{\mathsf{Accept}(\langle A, B, X, \mathcal{I}\rangle, \mathit{key}')@k\} \quad \text{and}\\ \Gamma_{1.2} &= \Gamma_1 \cup \{\forall k. \neg\mathsf{RevealLtk}(A)@k \vee \bot\}.\end{aligned}$$

The constraint system $\Gamma_{1.1}$ captures the case where the test session $\langle B,\ A,\ X,\ \mathcal{R}\rangle$ has a matching initiator session and $\Gamma_{1.2}$ captures the case where such a session might not exist. Then, a long-term key reveal for A, the peer of the test session, is forbidden. In the following, we show how to reduce $\Gamma_{1.1}$ and $\Gamma_{1.2}$ to the empty set of constraint systems.

To obtain $\Gamma_{1.1.1}$ shown in Figure 3.19, we solve both Accept-actions, the K-action, and the premises of the added responder rule. We also solve the variant formula ψ_{variants} (see Figure 3.21) which has been added with the responder rule. To achieve this, we use $\mathcal{S}_\vee$, $\mathcal{S}_\exists$, $\mathcal{S}_\approx$, and $\mathcal{S}_\downarrow$. All disjuncts except for the second one are contradictory since X has been instantiated with $\mathsf{g}\hat{\ }(x{:}\mathit{fr})$ by the initiator rule. Note that since we allow an agent to register more than one long-term key, the key KEY of the test session might differ from the key KEY' of its matching session.

In the next step, we solve the premise $\mathsf{K}^\Uparrow(\mathit{KEY})$ using $\mathcal{S}_{\mathbf{Prem},\mathsf{K}^\Uparrow}$, the previously computed constraint rewriting rule $\mathcal{R}_{\mathrm{hash}}$, and $\mathcal{S}_\approx$. Afterwards, we solve the resulting $\mathsf{K}^\Uparrow$-premises for pairs using $\mathcal{S}_{\mathbf{Prem},\mathsf{K}^\Uparrow}$ and $\mathcal{S}_{\rhd,\mathit{inv}}$ to obtain $\Gamma_{1.1.1.1}$. To save space, we use "..." to denote the unchanged constraints from the previous constraint system without new edges.

$\Gamma_{1.1.1.1}$ contains the two unsolved constraints $j_3 \rhd \mathsf{g}\hat{\ }(b{*}x)$ and $j_5 \rhd \mathsf{g}\hat{\ }(b{*}a)$. Intuitively, the adversary cannot deduce these messages since this would require either b or x *and* a. To prove this formally, we solve $j_5 \rhd \mathsf{g}\hat{\ }(b{*}a)$ using the constraint rewriting rule $\mathcal{R}_{\exp}$ and $\mathcal{S}_\approx$, which results in three cases. In the case $\Gamma_{1.1.1.1.2}$, which results from Δ_5 in $\mathcal{R}_{\exp}$, j_5 is a construction rule for exponentiation with the unsolved premise $\mathsf{K}^\Uparrow(b{*}a)$. We use $\mathcal{S}_{\mathbf{Prem},\mathsf{K}^\Uparrow}$ and $\mathcal{S}_{\rhd,*}$ to obtain constraints denoting that b and a are known to the adversary. Since the constraint system contains a formula denoting that $\mathsf{RevealLtk}(B)$ is not allowed in the trace, we obtain a contradiction by solving $j_7 \rhd b$.



$\Gamma_{1.1.1}$
i_1 : Fr(a: fr) / !Ltk(A: pub, a) !Pk(A, g ^ a) Out(g ^ a)
i_2 : Fr(b: fr) / !Ltk(B: pub, b) !Pk(B, g ^ b) Out(g ^ b)
i : In(g ^ x) !Pk(A, g ^ a) !Ltk(B, b) / [Accept(⟨B, A, g ^ x, $\mathcal{R}$, KEY)]
k : Fr(x: fr) !Ltk(A, a') !Pk(B, g ^ b') / Out(g ^ x) !Ephk(⟨A, B, g ^ x, $\mathcal{I}$⟩, x) [Accept(⟨A, B, g ^ x, $\mathcal{I}$⟩, KEY')]
j : K⇑(KEY) / In(KEY) [K(KEY)]
∀k.¬(RevealLtk(B)@k) ∨ ⊥
∀k.¬(RevealLtk(A)@k) ∨ ∀l.¬(RevealEphk(⟨A, B, g ^ x, $\mathcal{I}$⟩) ∨ ⊥
$\mathcal{S}_{\mathbf{Prem},K^{\Uparrow}}, \mathcal{R}_{\mathsf{hash}}, \mathcal{S}_{\approx}, \mathcal{S}_{\triangleright,inv}$
Abbreviations:
$KEY = h(\mathsf{g}\,\hat{}\,(b*x), \mathsf{g}\,\hat{}\,(b*a), A, B, \mathsf{g}\,\hat{}\,x)$
$KEY' = h(\mathsf{g}\,\hat{}\,(b'*x), \mathsf{g}\,\hat{}\,(b'*a'), A, B, \mathsf{g}\,\hat{}\,x)$
$\Gamma_{1.1.1.1}$
$j_5 \triangleright$ g ^ ($b*a$)
j_4 : K⇑(g ^ ($b*a$)) K⇑(⟨A, B, g ^ x⟩) / K⇑(...)
$j_3 \triangleright$ g ^ ($b*x$)
j_2 : K⇑(g ^ ($b*x$)) K⇑(⟨g ^ ($b*a$), A, B, g ^ x⟩) / K⇑(...)
j_1 : K⇑(⟨g ^ ($b*x$), g ^ ($b*a$), A, B, g ^ x⟩) / K⇑(KEY)
j : K⇑(KEY) / In(KEY) [K(KEY)]
...
$\mathcal{R}_{\mathsf{exp}}, \mathcal{S}_{\approx}, \mathcal{S}_{\mathbf{Prem},K^{\Uparrow}}, \mathcal{S}_{\triangleright,*}$
$\mathcal{R}_{\mathsf{exp}}, \mathcal{S}_{\approx}, \mathcal{S}_{\mathbf{UFresh}}, \mathcal{S}_{\mathbf{ULabel}}, \mathcal{S}_{\mathbf{Prem},K^{\Uparrow}}$
$\mathcal{R}_{\mathsf{exp}}, \mathcal{S}_{\approx}, \mathcal{S}_{\mathbf{UFresh}}, \mathcal{S}_{\mathbf{ULabel}}, \mathcal{S}_{\mathbf{Prem},K^{\Uparrow}}$
$\Gamma_{1.1.1.1.1}$
$j_7 \triangleright b$
$j_8 \triangleright a$
$j_6 \triangleright b*a$
j_5 : K⇑(g) K⇑($b*a$) / K⇑(...)
j_4 : K⇑(g ^ ($b*a$)) K⇑(⟨A, B, g ^ x⟩) / K⇑(...)
...
$\Gamma_{1.1.1.1.2}$
i_1 : Fr(a: fr) / !Ltk(A, a) !Pk(A, g ^ a) Out(g ^ a)
j_7 : Out(g ^ a) / K⇓d(g ^ a)
$j_8 \triangleright b$
j_6 : K⇓d(g ^ a) K⇑(b) / K⇓e(g ^ ($b*a$))
j_5 : K⇓e(...) / K⇑(...)
j_4 : K⇑(g ^ ($b*a$)) K⇑(⟨A, B, g ^ x⟩) / K⇑(...)
...
$\Gamma_{1.1.1.1.3}$
i_2 : Fr(b: fr) / !Ltk(B, b) !Pk(B, g ^ b) Out(g ^ b)
j_7 : Out(g ^ b) / K⇓d(g ^ b)
$j_8 \triangleright a$
j_6 : K⇓d(g ^ b) K⇑(a) / K⇓e(g ^ ($b*a$))
j_5 : K⇓e(...) / K⇑(...)
j_4 : K⇑(g ^ ($b*a$)) K⇑(⟨A, B, g ^ x⟩) / K⇑(...)
...
$\mathcal{R}_{\mathsf{fresh}}, \mathcal{S}_{\approx}, \mathcal{S}_{\mathbf{ULabel}}, \mathcal{S}_{\mathbf{UFresh}}, \mathcal{S}_{\forall,@}, \mathcal{S}_{\bot}$
⋄
$\mathcal{R}_{\mathsf{fresh}}, \mathcal{S}_{\approx}, \mathcal{S}_{\mathbf{ULabel}}, \mathcal{S}_{\mathbf{UFresh}}, \mathcal{S}_{\forall,@}, \mathcal{S}_{\bot}$
⋄
$\mathcal{R}_{\mathsf{fresh}}, \mathcal{S}_{\approx}, \mathcal{S}_{\mathbf{ULabel}}, \mathcal{S}_{\mathbf{UFresh}}, \mathcal{S}_{\forall,@}$
$\Gamma_{1.1.1.1.3.1}$ (see next page)

Figure 3.19. Constraint reduction for $\Gamma_{1.1}$.

In $\Gamma_{1.1.1.1.2}$, which results from Δ_4 in $\mathcal{R}_{\text{exp}}$, j_6 is an exponentiation rule applied to the public key $\mathsf{g}\hat{}(a{:}fr)$ of A. Again, b is known to the adversary and we can derive a contradiction in a similar way. In $\Gamma_{1.1.1.1.3}$, which also results from Δ_4 in $\mathcal{R}_{\text{exp}}$ for another AC-unifier, j_6 is an exponentiation rule applied to the public key $\mathsf{g}\hat{}(b{:}fr)$ of B. Here, a must be be known to the adversary. After solving a, we obtain the new case $\Gamma_{1.1.1.1.3.1}$ shown in Figure 3.20 where the adversary learns a by performing a long-term key reveal on A. Since this is allowed if there is no ephemeral key reveal for the matching session, applying $\mathcal{S}_{\forall,@}$ does not directly yield a contradiction. Instead, we obtain a new constraint denoting that ephemeral key reveals for the session $\langle A, B, g\hat{}(x{:}fr), \mathcal{I}\rangle$ are forbidden.

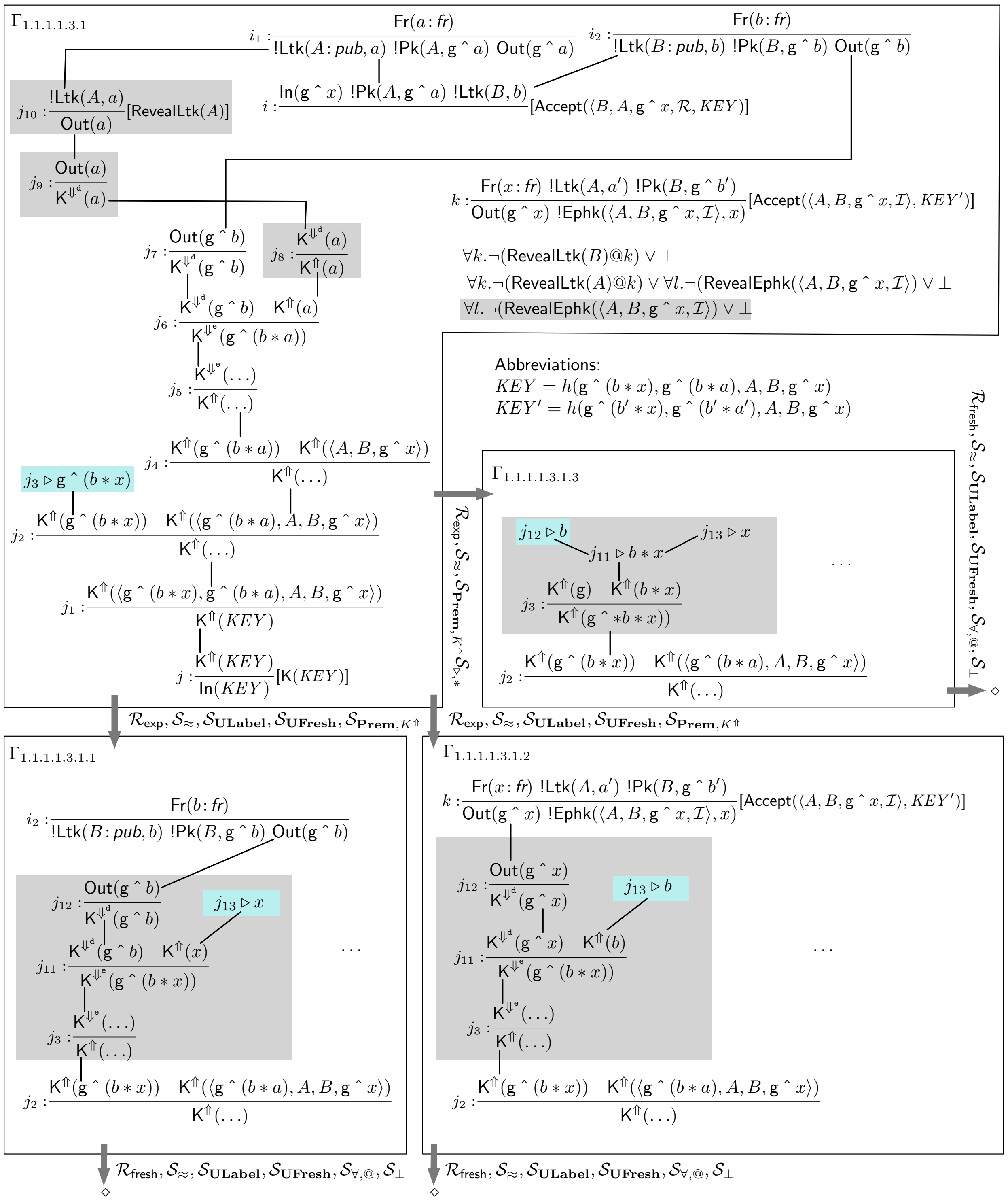


Figure 3.20. Constraint reduction for $\Gamma_{1.1}$ continued.

In $\Gamma_{1.1.1.1.3.1}$, we can now solve $j_3 \triangleright \mathsf{g}\hat{\ }(b{*}x)$ similarly to how we solved $j_5 \triangleright \mathsf{g}\hat{\ }(b{*}a)$ earlier. In all of the cases, b or x must be known. Since the adversary can learn b only after a long-term key reveal for B, which is forbidden, the corresponding cases are contradictory. The same holds for the remaining cases since the adversary can learn x only after an ephemeral key reveal for the matching session $\langle A, B, \mathsf{g}\hat{\ }x, \mathcal{I}\rangle$, which is also forbidden.

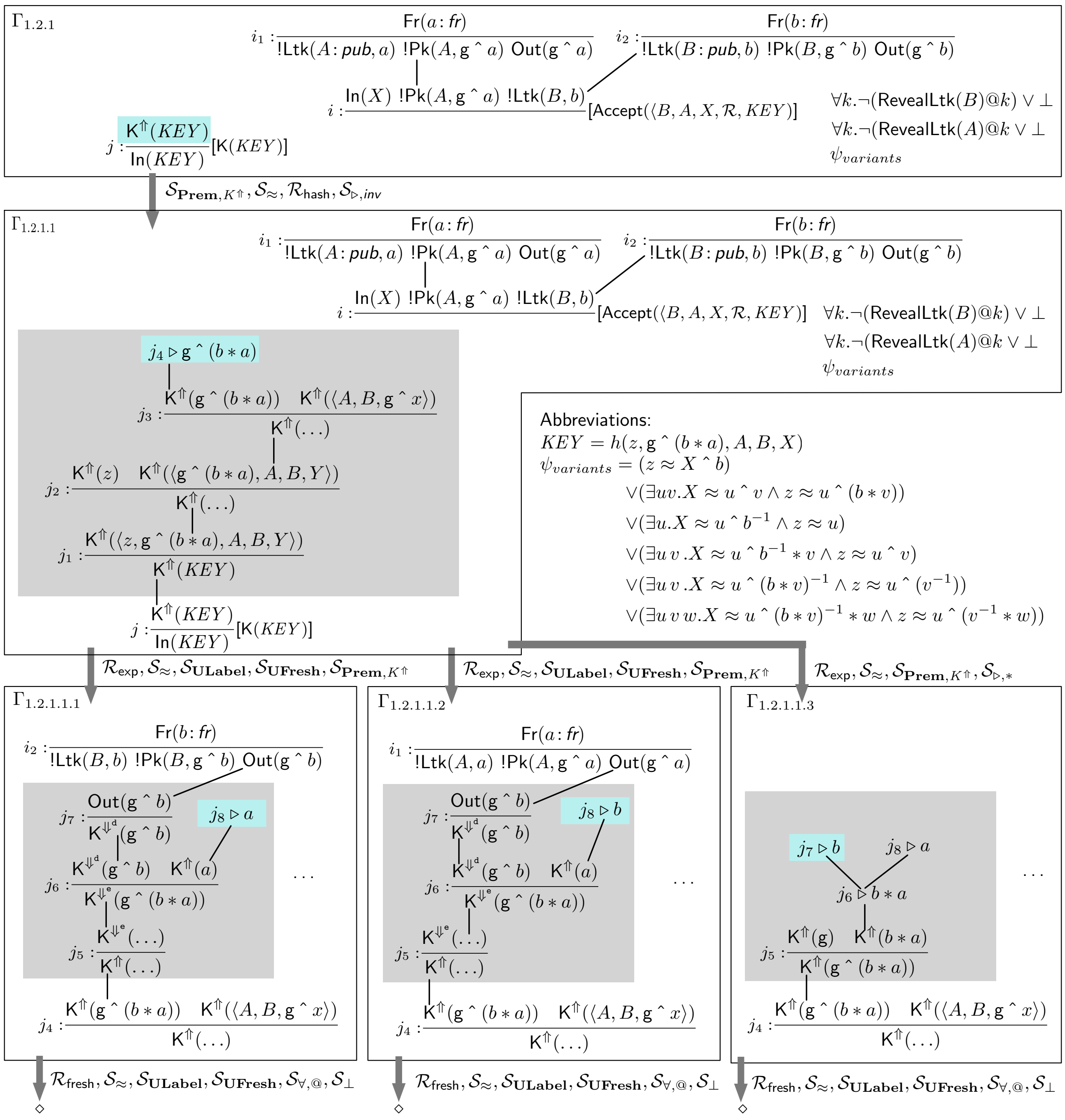

$\Gamma_{1.2.1}$
i_1 : Fr(a: fr) / !Ltk(A: pub, a) !Pk(A, g ^ a) Out(g ^ a)
i_2 : Fr(b: fr) / !Ltk(B: pub, b) !Pk(B, g ^ b) Out(g ^ b)
i : In(X) !Pk(A, g ^ a) !Ltk(B, b) [Accept(⟨$B, A, X, \mathcal{R}, KEY$⟩)]
j : K⇑(KEY) / In(KEY) [K(KEY)]
∀k.¬(RevealLtk(B)@k) ∨ ⊥
∀k.¬(RevealLtk(A)@k ∨ ⊥
$\psi_{variants}$
$\mathcal{S}_{\mathbf{Prem},K^\Uparrow}, \mathcal{S}_\approx, \mathcal{R}_{hash}, \mathcal{S}_{\triangleright,inv}$
$\Gamma_{1.2.1.1}$
$j_4 \triangleright$ g ^ ($b * a$)
j_3 : K⇑(g ^ ($b*a$)) K⇑(⟨A, B, g ^ x⟩) / K⇑(…)
j_2 : K⇑(z) K⇑(⟨g ^ ($b*a$), A, B, Y⟩) / K⇑(…)
j_1 : K⇑(⟨z, g ^ ($b*a$), A, B, Y⟩) / K⇑(KEY)
Abbreviations:
$KEY = h(z, g\,\hat{}\,(b*a), A, B, X)$
$\psi_{variants} = (z \approx X\,\hat{}\,b)$
$\vee(\exists uv.X \approx u\,\hat{}\,v \wedge z \approx u\,\hat{}\,(b*v))$
$\vee(\exists u.X \approx u\,\hat{}\,b^{-1} \wedge z \approx u)$
$\vee(\exists u\,v.X \approx u\,\hat{}\,b^{-1} * v \wedge z \approx u\,\hat{}\,v)$
$\vee(\exists u\,v.X \approx u\,\hat{}\,(b*v)^{-1} \wedge z \approx u\,\hat{}\,(v^{-1}))$
$\vee(\exists u\,v\,w.X \approx u\,\hat{}\,(b*v)^{-1} * w \wedge z \approx u\,\hat{}\,(v^{-1} * w))$
$\mathcal{R}_{exp}, \mathcal{S}_\approx, \mathcal{S}_{\mathbf{ULabel}}, \mathcal{S}_{\mathbf{UFresh}}, \mathcal{S}_{\mathbf{Prem},K^\Uparrow}$
$\mathcal{R}_{exp}, \mathcal{S}_\approx, \mathcal{S}_{\mathbf{Prem},K^\Uparrow}, \mathcal{S}_{\triangleright,*}$
$\Gamma_{1.2.1.1.1}$
i_2 : Fr(b: fr) / !Ltk(B, b) !Pk(B, g ^ b) Out(g ^ b)
j_7 : Out(g ^ b) / K⇓d(g ^ b)
$j_8 \triangleright a$
j_6 : K⇓d(g ^ b) K⇑(a) / K⇓e(g ^ ($b*a$))
j_5 : K⇓e(…) / K⇑(…)
j_4 : K⇑(g ^ ($b*a$)) K⇑(⟨A, B, g ^ x⟩) / K⇑(…)
…
$\Gamma_{1.2.1.1.2}$
i_1 : Fr(a: fr) / !Ltk(A, a) !Pk(A, g ^ a) Out(g ^ a)
j_7 : Out(g ^ b) / K⇓d(g ^ b)
$j_8 \triangleright b$
j_6 : K⇓d(g ^ b) K⇑(a) / K⇓e(g ^ ($b*a$))
j_5 : K⇓e(…) / K⇑(…)
j_4 : K⇑(g ^ ($b*a$)) K⇑(⟨A, B, g ^ x⟩) / K⇑(…)
$\Gamma_{1.2.1.1.3}$
$j_7 \triangleright b$
$j_8 \triangleright a$
$j_6 \triangleright b * a$
j_5 : K⇑(g) K⇑($b*a$) / K⇑(g ^ ($b*a$))
j_4 : K⇑(g ^ ($b*a$)) K⇑(⟨A, B, g ^ x⟩) / K⇑(…)
$\mathcal{R}_{fresh}, \mathcal{S}_\approx, \mathcal{S}_{\mathbf{ULabel}}, \mathcal{S}_{\mathbf{UFresh}}, \mathcal{S}_{\forall,@}, \mathcal{S}_\perp$
◇

Figure 3.21. Constraint reduction for $\Gamma_{1.2}$.

The constraint reduction for $\Gamma_{1.2}$ shown in Figure 3.21 is similar to the one for $\Gamma_{1.1}$. The constraint system $\Gamma_{1.2.1}$ shows the result of solving the K-action, the Accept-action for the responder, and the resulting unsolved premises of the responder rule. In contrast to $\Gamma_{1.1.1}$, we know that there is no long-term key reveal for A. We delay solving $\psi_{variants}$ since we do not know the structure of the received message X. Nevertheless, we know that KEY is a hash and the only unknown component is the first input to the hash function, the variable z constrained by $\psi_{variants}$. We therefore continue as previously until we obtain $\Gamma_{1.2.1.1}$ where $\mathsf{g}\,\hat{}\,(b{*}a)$ is known to the adversary. Solving the corresponding constraint $j_4 \triangleright \mathsf{g}\,\hat{}\,(b{*}a)$ results in three contradictory cases where either a or b is known, even though long-term key reveals for both A and B are forbidden.

Note that both optimizations helped to decrease the size of the proof considerably. Without lazy case distinctions for variants, we would have to repeat the constraint reduction sequence for $\Gamma_{1.2}$ for each of the six variants of the responder rule, even though the steps are independent of the concrete instantiation of z. The precomputed constraint rewriting rules allow us to perform the reasoning about extractable messages once and for all before starting with the actual proof. In the actual proof, there is no need to use the rules $\mathcal{S}_{\mathbf{Prem},\mathsf{K}^{\Downarrow y}}$ and $\mathcal{S}_{\twoheadrightarrow}$. Still, working out the constraint reductions manually is tedious and error-prone. That is why we implemented the TAMARIN prover.

3.4 Extension with Bilinear Pairings and AC operators

In this section, we show how to extend the method presented in the previous sections with support for bilinear pairings and an AC operator. This allows us to analyze tripartite and identity-based key exchange protocols as described in Section 2.1.3.2. The support for an AC operator, which we also allow in guarded trace properties, considerably simplifies the specification of tripartite key exchange protocols since we can directly model the multiset of participants. The structure of this section mirrors that of the previous sections since we list the modifications and additions required for the extension.

3.4.1 Security Protocol Model

First, we adapt the definition of cryptographic messages and protocols. Afterwards, we extend the set of message deduction rules.

Cryptographic Messages. To support the new cryptographic operators, we use the equational theory $\mathcal{BP} = (\Sigma_{\mathcal{BP}}, E_{\mathcal{BP}})$ instead of $\mathcal{DH}$. We define the signature as

$$\Sigma_{\mathcal{BP}} = \{\hat{e}(\_,\_), [\_]\_, \_\sharp\_\} \cup \Sigma_{\mathcal{DH}}$$

and the equations as

$$E_{\mathcal{BP}} = \left\{ \begin{array}{ll} [z]([y]x) \simeq [z*y]x & [1]x \simeq x \\ \hat{e}(x,y) \simeq \hat{e}(y,x) & \hat{e}([z]x,y) \simeq \hat{e}(x,y)\hat{\ }z \\ x\sharp(y\sharp z) \simeq (x\sharp y)\sharp z & x\sharp y \simeq y\sharp x \end{array} \right\} \cup E_{\mathcal{DH}}.$$

As for $\mathcal{DH}$, we use $\mathcal{BP}_e$ to denote the combined equational theory of $\mathcal{BP}$ and $\mathcal{ST}$ and $\Sigma_{\mathcal{BP}_e}$ to denote the combined signature. To model the setup described in Section 2.1.3.1, we can use a fixed public name P and use terms $[s]\mathsf{P}$ to model elements of the group $\mathbb{G}_1$. The bilinear map then sends two terms $[s]\mathsf{P}$ and $[t]\mathsf{P}$ to $\hat{e}([s]\mathsf{P},[t]\mathsf{P}) =_{\mathcal{BP}} \hat{e}(\mathsf{P},\mathsf{P})\hat{\ }(s*t)$. The elements of the group $\mathbb{G}_2$ are therefore modeled as terms $\hat{e}(\mathsf{P},\mathsf{P})\hat{\ }u$. We use the $\sharp$ operator to model non-empty multisets. For example, $\mathsf{A}\sharp\mathsf{B}\sharp\mathsf{C}$ models the multiset that consists of the three given agents. To encode such a message as a bitstring, a sorted list could be used.

Protocol Rules. We modify **P6** from Definition 3.4 on page ? such that $\sharp$ can be used in the premises l of a protocol rule. The other conditions remain unchanged.

Message Deduction Rules. Note that the extended signature $\Sigma_{\mathcal{BP}}$ yields additional message deduction rules for constructing multisets and performing scalar multiplication and bilinear pairings. To allow the adversary to extract elements from multisets, we extend the set of message deduction rules MD defined on page ? with the rule $\mathsf{K}(x \sharp y) \xrightarrow{[]} \mathsf{K}(x)$.

3.4.2 Formalization of the Joux Protocol

We can now formalize the signed Joux protocol from Section 2.1.3.2. The multiset rewriting rules formalizing the protocol, and the security property, are given in Figures 3.22 and 3.23.

In the first step, an agent A chooses his private ephemeral key x and two peers B and C. Then, he uses his signing key a to sign his own identity, the multiset of peers, and his public ephemeral key $[x]\mathsf{P}$. The protocol state fact $St(x, A, B\sharp C)$ denotes that a session with the given protocol parameters has executed the first step. In the second step, A waits for the signed messages from the peers, checks their signatures, extracts their ephemeral public keys Y and Z, and computes the shared key as $\hat{e}(Y, Z)\hat{\ }x$. The Accept-fact denotes that A has accepted the given key as shared with the given multiset of peers.

The property captures perfect forward secrecy of the session key under long-term key reveals. If the key was accepted by A with peers B and C at timepoint i and is known to the adversary, then there must be a long-term key reveal for one of the involved agents before i.

Note that we use pattern matching for the signature check for convenience. We can also check if $verify(s, \langle B, A\sharp C, Y\rangle, pk(b)) =_{\mathcal{BP}_e} true$ for the signature s by extending the second step with the action $\mathsf{Eq}(verify(s, \langle B, A\sharp C, Y\rangle, pk(b)), true)$ and modifying the security property to $(\forall k x y.\ \mathsf{Eq}(x, y)@k \Longrightarrow x \approx y) \Longrightarrow \varphi_{\text{SigJoux}}$. This reflects that we verify the property under the additional assumption that all equality checks performed by the protocol, encoded as Eq-actions, are successful. The same technique can be used to model inequality checks and uniqueness of events. For example, we could enforce that the participating agents are distinct or that each agent has only one registered key. For the first example, we can use InEq-actions and the assumption $\neg(\exists k x. \mathsf{InEq}(x, x)@k)$. For the second example, we can use actions of the form $\mathsf{Once}(\langle \texttt{keyreg}, A\rangle)$ and the assumption $\forall i j x.\ \mathsf{Once}(x)@i \wedge \mathsf{Once}(x)@j \Longrightarrow i \approx j$.

$$
\begin{aligned}
&\forall i\,A\,B\,C\,key.\\
&\quad \text{// If the key } key \text{ has been accepted and is known to the adversary,}\\
&\quad \mathsf{Accept}(A, B\sharp C, key)@i \wedge \mathsf{K}(key)\\
&\quad \text{// then the long-term signing key of one of the participants must have}\\
&\quad \text{// been revealed before the session key was accepted.}\\
&\quad \Longrightarrow\ (\exists l.\ \mathsf{RevealLtk}(A)@l \wedge l \prec i) \vee (\exists l.\ \mathsf{RevealLtk}(B)@l \wedge l \prec i)\\
&\qquad\quad \vee (\exists l.\ \mathsf{RevealLtk}(C)@l \wedge l \prec i)
\end{aligned}
$$

Figure 3.23. Security property for the *SIGJOUX* protocol.

3.4.3 Verification Theory

We extend parts of the verification theory as follows. First, we adapt the switch to dependency graphs modulo AC to account for the new equations in $\mathcal{BP}_e$. Then, we extend the set of normal messages deduction rules to account for the new cryptographic operators.

Key Registration: $$\frac{\mathsf{Fr}(a{:}fr)}{\mathsf{!Ltk}(A{:}pub, a)\quad \mathsf{!Pk}(A{:}pub, pk(a))\quad \mathsf{Out}(pk(a))}$$

First Step: $$\frac{\mathsf{Fr}(x{:}fr)\quad \mathsf{!Ltk}(A{:}pub, a{:}fr)}{\begin{array}{l}\mathsf{Out}(\langle [x]\mathsf{P}, sig(\langle A, (B{:}pub)\sharp(C{:}pub), [x]\mathsf{P}\rangle, a)\rangle)\\ \mathsf{St}(x, A, B\sharp C)\end{array}}$$

Second Step: $$\frac{\begin{array}{l}\mathsf{St}(x, A, B\sharp C)\\ \mathsf{In}(\langle Y, sig(\langle B, A\sharp C, Y\rangle, b)\rangle)\\ \mathsf{In}(\langle Z, sig(\langle C, A\sharp B, Z\rangle, c)\rangle)\\ \mathsf{!Pk}(B, pk(b))\quad \mathsf{!Pk}(C, pk(c))\end{array}}{}[\,\mathsf{Accept}(A, B\sharp C, \hat{e}(Y, Z)\hat{\ }x)\,]$$

Long-term key reveal: $$\frac{\mathsf{!Ltk}(A, a)}{\mathsf{Out}(a)}[\mathsf{RevealLtk}(A)]$$

Figure 3.22. Rules defining the *SIGJOUX* protocol.

Finally, we introduce new normal-form conditions to account for the new normal message deduction rules and relax the definition of guarded trace properties

Dependency Graphs modulo *ACC*

We first define the set of equations ACC as

$$ACC = \{\, x\sharp(y\sharp z) \simeq (x\sharp y)\sharp z \quad x\sharp y \simeq y\sharp x \quad \hat{e}(x, y) \simeq \hat{e}(y, x)\,\} \cup AC$$

and the rewriting system $\mathcal{RBP}$ as

$$\mathcal{RBP} = \{\, [z]([y]x) \rightarrow [z{*}y]x \quad [1]x \rightarrow x \quad \hat{e}([y]x, z) \rightarrow \hat{e}(x, z)\hat{\ }y\,\} \cup \mathcal{RDH}.$$

We define $\mathcal{RBP}_e = \mathcal{RBP} \cup \mathcal{ST}$. Then $(\Sigma_{\mathcal{BP}_e}, \mathcal{RBP}_e, ACC)$ is a decomposition of $\mathcal{BP}_e$ with the finite variant property. The arguments are similar to the ones for $\mathcal{RDH}_e$ and we have again used the AProVE termination tool [85] and the Maude Church-Rosser and Coherence Checker [73] to verify termination, confluence, and coherence.

Normal Message deduction

To define the new message deduction rules, we extend the meaning of $\mathsf{K}^{\Downarrow^{\mathsf{d}}}$-facts as follows. $\mathsf{K}^{\Downarrow^{\mathsf{d}}}(m)$ means that m is an extracted subterm or the result of applying $\hat{e}$ to at least one extracted subterm. We also extend the meaning of $\mathsf{K}^{\Downarrow^{\mathsf{e}}}$ to include that the corresponding message can be the result of changing the scalar in an extracted scalar multiplication.

The normal message deduction rules $ND_{\mathcal{BP}}$ for bilinear pairing and $\sharp$ are given in Figure 3.24. They extend the originally defined set ND. Scalar multiplication is treated similarly to exponentiation, i.e., there is a construction rule, there are deconstruction rules, and scalar multiplication rules which use the fact symbol $\mathsf{K}^{\Downarrow^{\mathsf{e}}}$ in the conclusion. For the bilinear pairing, there is a construction rule and there are bilinear pairing rules corresponding to the non-trivial variants of $\hat{e}(x, y)$. They cover all the different possibilities how to normalize the product of the scalars from the two scalar multiplications given as arguments to $\hat{e}$. The message in the conclusion of a bilinear pairing rule is always

Construction rules:

$$\frac{\mathsf{K}^{\Uparrow}(x) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Uparrow}([y]x)} \qquad \frac{\mathsf{K}^{\Uparrow}(x) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Uparrow}(\hat{e}(x,y))} \qquad \frac{\mathsf{K}^{\Uparrow}(x_1) \quad \ldots \quad \mathsf{K}^{\Uparrow}(x_k)}{\mathsf{K}^{\Uparrow}(x_1 \sharp \ldots \sharp x_k)}$$

Deconstruction rules:

$$\frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}([y]x) \quad \mathsf{K}^{\Uparrow}(y^{-1})}{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(x)} \qquad \frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}([y^{-1}]x) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(x)} \qquad \frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}([(y*z^{-1})]x) \quad \mathsf{K}^{\Uparrow}(y^{-1}*z)}{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(x)} \qquad \frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(x \sharp y)}{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(x)}$$

Scalar multiplication rules:

$$\frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}([y]x) \quad \mathsf{K}^{\Uparrow}(z)}{\mathsf{K}^{\Downarrow^{\mathsf{e}}}([y*z]x)} \qquad \frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}([y]x) \quad \mathsf{K}^{\Uparrow}(y^{-1}*z)}{\mathsf{K}^{\Downarrow^{\mathsf{e}}}([z]x)} \qquad \ldots \qquad \frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}([y_1*y_2]x) \quad \mathsf{K}^{\Uparrow}(z_1*z_2^{-1})}{\mathsf{K}^{\Downarrow^{\mathsf{e}}}([y_1*z_1*(y_2*z_2)^{-1})]x)}$$

Bilinear pairing rules:

$$\frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}([z]x) \quad \mathsf{K}^{\Uparrow}(y)}{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\hat{e}(x,y)\,\hat{}\,z)} \qquad \frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}([z_1]x) \quad \mathsf{K}^{\Downarrow^{\mathsf{d}}}([z_2]y)}{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\hat{e}(x,y)\,\hat{}\,(z_1*z_2))} \qquad \ldots \qquad \frac{\mathsf{K}^{\Downarrow^{\mathsf{d}}}([y_1*y_2^{-1}]x_1) \quad \mathsf{K}^{\Downarrow^{\mathsf{d}}}([z_1*z_2^{-1}]x_2)}{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\hat{e}(x_1,x_2)\,\hat{}\,(y_1*z_1*(y_2*z_2)^{-1}))}$$

Figure 3.24. The normal message deduction rules $ND_{\mathcal{BP}}$ for bilinear pairing. There are construction rules for $\sharp$ for all $k > 1$. There are 42 scalar multiplication rules and 28 bilinear pairing rules computed from the $\mathcal{RBP}_e$,ACC-variants of the corresponding rules.

an exponentiation and can therefore only be used by COERCE, an exponentiation rule, or a deconstruction rule for exponentiation. Note that for the first bilinear pairing rule, the first premise is a scalar multiplication and uses $\mathsf{K}^{\Downarrow^{\mathsf{d}}}$ and the second premises uses $\mathsf{K}^{\Uparrow}$ and cannot be a scalar multiplication if the instance is in normal form. For all remaining bilinear pairing rules, both premises are scalar multiplications and use $\mathsf{K}^{\Downarrow^{\mathsf{d}}}$-facts. Figure 3.25 shows the main idea why this choice does not make the rules incomplete.

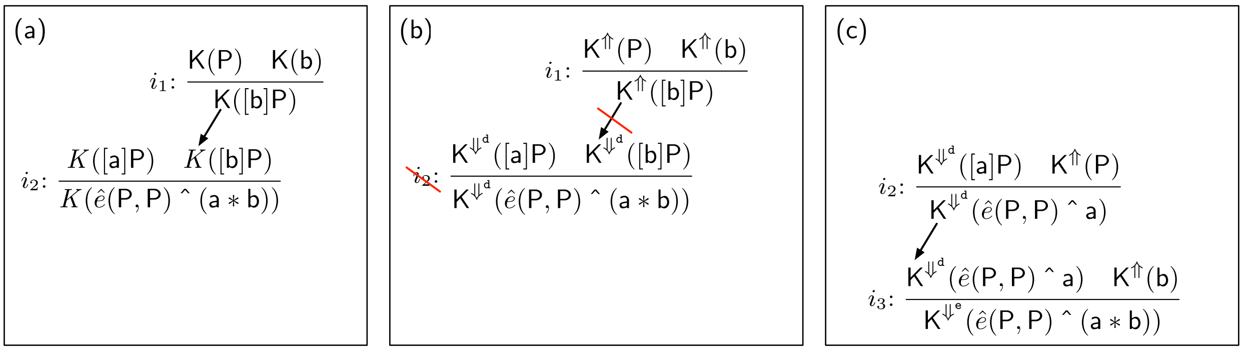

(a)
i_1: K(P) K(b) / K([b]P)
i_2: K([a]P) K([b]P) / K(ê(P, P) ^ (a ∗ b))
(b)
i_1: K⇑(P) K⇑(b) / K⇑([b]P)
i_2: K⇓d([a]P) K⇓d([b]P) / K⇓d(ê(P, P) ^ (a ∗ b))
(c)
i_2: K⇓d([a]P) K⇑(P) / K⇓d(ê(P, P) ^ a)
i_3: K⇓d(ê(P, P) ^ a) K⇑(b) / K⇓e(ê(P, P) ^ (a ∗ b))

Figure 3.25. Message deduction (a) is not possible using the normal message deduction rules because the crossed edge shown in (b) is not allowed. It can be replaced by the normal message deduction (c).

Normal Dependency Graphs

We introduce five new normal-form conditions. To state the conditions, we require the following definitions. A node i labeled with an instance of a protocol rule is the *send-node of the premise* (j, u) *in* dg if there is a node k labeled with an instance of RECV such that there is an edge $(i,v) \rightarrowtail (k,1)$ for some v and a chain $k \twoheadrightarrow (j,u)$. Intuitively, the send-node of a premise $\mathsf{K}^{\Downarrow^y}(m)$ is the protocol rule that sends the message from which m is extracted. We denote the sequence of fact symbols occurring in a multiset rewriting rule ru with $\mathit{fsyms}(ru)$. We also assume given a total order $<_{\mathit{fs}}$ on sequences of fact symbols.

N7. There is no construction rule for $\sharp$ that has a premise of the form $\mathsf{K}^{\Uparrow}(s\sharp t)$ and all conclusion facts of the form $\mathsf{K}^{\Uparrow}(s\sharp t)$ are conclusions of a construction rule for $\sharp$.

N8. The conclusion of a deconstruction rule for $\sharp$ is never of the form $\mathsf{K}^{\Downarrow^{\mathsf{d}}}(s\sharp t)$.

N9. There is no node $\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(a), \mathsf{K}^{\Uparrow}(b)\right] -[]\!\!\rightarrow \mathsf{K}^{\Downarrow^{\mathsf{e}}}([d]c)$ such that c does not contain any fresh names and $\mathit{nifactors}(d) \subseteq_{ACC} \mathit{nifactors}(b)$.

N10. There is no node i labeled with $\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}([t_1]p), \mathsf{K}^{\Downarrow^{\mathsf{d}}}([t_2]q)\right] -[]\!\!\rightarrow \mathsf{K}^{\Downarrow^{\mathsf{d}}}(\hat{e}(p,q)\hat{\ }c)$ such that there is a node j labeled with $\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\hat{e}(p,q)\hat{\ }c), \mathsf{K}^{\Uparrow}(d)\right] -[]\!\!\rightarrow \mathsf{K}^{\Downarrow^{\mathsf{d}}}(\hat{e}(p,q)\hat{\ }e)$, an edge $(i,1) \rightarrowtail (j,1)$, $\mathit{nifactors}(t_i) \subseteq_{ACC} \mathit{nifactors}(d)$ for $i = 1$ or $i = 2$, and $\hat{e}(p,\, q)$ does not contain any fresh names.

N11. There is no node $\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}([a]p), \mathsf{K}^{\Downarrow^{\mathsf{d}}}([b]q)\right] -[]\!\!\rightarrow \mathsf{K}^{\Downarrow^{\mathsf{d}}}(\hat{e}(p,q)\hat{\ }(a*b))$ such that the send-nodes of the first and second premise are labeled with ru_1 and ru_2 and $\mathit{fsyms}(ru_2) <_{\mathit{fs}} \mathit{fsyms}(ru_1)$.

The condition **N7** is similar to condition **N2** for multiplication, but deals with $\sharp$ instead. The condition **N8** ensures that the deconstruction rule for $\sharp$ never extracts a multiset. Together with **N7**, this enforces that multisets are completely deconstructed and then constructed from scratch. The condition **N9** directly corresponds to condition **N6** for exponentiation and forbids unnecessary uses of scalar multiplication rules.

The condition **N10** prevents deductions where an exponentiation rule is applied to the result of a bilinear pairing rule such that the deduction can be replaced by a simpler one. Figure 3.26 (a) shows one case prevented by this condition and (b) shows a simpler alternative to deduce the same message that is allowed. The condition **N11** prevents redundant cases resulting from the commutativity of $\hat{e}$ where two dependency graphs only differ in the order of premises of a bilinear pairing rule. This is especially problematic for the second bilinear pairing rule in Figure 3.24 which is symmetric and used very often. We therefore enforce that the send-node of the second premise cannot be smaller than the send-node of the first premise. Since we want to evaluate this condition on constraint systems, which are symbolic, we choose a very simple partial order on rule instances that considers only the fact symbols. Figure 3.26 (c) shows a violation of **N11**, swapping the premises of i_5 yields a node that is allowed and has the same conclusion (modulo ACC).

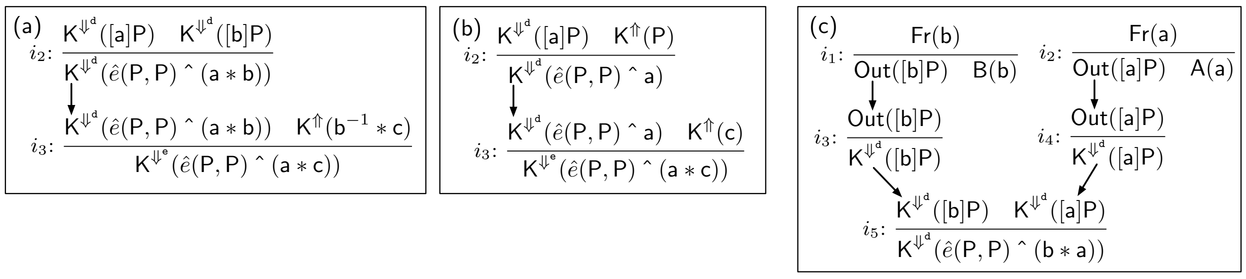

(a)
i2: K⇓d([a]P) K⇓d([b]P) / K⇓d(ê(P,P) ^ (a ∗ b))
i3: K⇓d(ê(P,P) ^ (a ∗ b)) K⇑(b⁻¹ ∗ c) / K⇓e(ê(P,P) ^ (a ∗ c))
(b)
i2: K⇓d([a]P) K⇑(P) / K⇓d(ê(P,P) ^ a)
i3: K⇓d(ê(P,P) ^ a) K⇑(c) / K⇓e(ê(P,P) ^ (a ∗ c))
(c)
i1: Fr(b) / Out([b]P) B(b)
i2: Fr(a) / Out([a]P) A(a)
i3: Out([b]P) / K⇓d([b]P)
i4: Out([a]P) / K⇓d([a]P)
i5: K⇓d([b]P) K⇓d([a]P) / K⇓d(ê(P,P) ^ (b ∗ a))

Figure 3.26. Message deduction (a) is forbidden by Condition **N10**, (b) can be used instead. (c) is forbidden by Condition **N11** if $[\mathsf{Fr}, \mathsf{Out}, B] >_{\mathit{fs}} [\mathsf{Fr}, \mathsf{Out}, A]$.

$\mathcal{S}_{\rhd,\sharp}$:	$\dfrac{i \rhd \mathsf{K}^{\Uparrow}(m)}{j \rhd \mathsf{K}^{\Uparrow}(t),\, j \prec i}\,\mathcal{I}$	if $root(m) = \sharp$, j freshly chosen, and $t \in elems(m) \setminus \mathcal{V}_{msg}$
$\mathcal{S}_{\twoheadrightarrow,\sharp}$:	$\dfrac{i \twoheadrightarrow (k,w), i\colon ru, \Gamma}{\begin{array}{l} i\colon ru, (i,1) \rightarrowtail (j,1), j\colon ru_1, j \twoheadrightarrow (k,w), \Gamma \\ \mid \ldots \\ \mid i\colon ru, (i,1) \rightarrowtail (j,1), j\colon ru_l, j \twoheadrightarrow (k,w), \Gamma \end{array}}\,\mathcal{M}$	if $(concs(ri))_1 = \mathsf{K}^{\Downarrow^{\mathsf{d}}}(t)$ and $t = a\sharp b$ for some a and b, $elems(t) \cap \mathcal{V}_{msg} \subseteq known^{\prec}_{\Gamma}(i)$, $\{ru_1, \ldots, ru_l\} =$ $\{\mathsf{K}^{\Downarrow^{\mathsf{d}}}(t) \mathrel{-[]\!\!\rightarrow} \mathsf{K}^{\Downarrow^{\mathsf{d}}}(m)$ $\mid m \in elems(t) \setminus \mathcal{V}_{msg}\}$, and j freshly chosen

Figure 3.27. Constraint solving rules for message deduction with respect to $\sharp$.

Guarded Trace Properties

We relax the definition of guarded trace property to allow for subterms t with $root(t) = \sharp$ in addition to variables, public names, and irreducible function symbols from $\Sigma_{\mathcal{ST}}$. This enables the usage of $\sharp$ in security properties as already demonstrated for the Joux protocol.

3.4.4 Constraint Solving

In this section, we first present five new constraint solving rules required for $\sharp$ and bilinear pairing. Then, we discuss constraint solving for the signed Joux protocol.

Constraint solving rules for message deduction

Before showing the new constraint solving rules, we present the required definitions. The *messages known before i in Γ* are defined as

$$known^{\prec}_{\Gamma}(i) = \{m \mid \exists j. j \prec_{\Gamma} i \wedge j \rhd \mathsf{K}^{\Uparrow}(m)\}.$$

The *set of elements of a term t* is defined as

$$elems(t) = \begin{cases} elems(a) \cup elems(b) & \text{if } t = a\sharp b \\ elems(t) & \text{otherwise} \end{cases}.$$

The new constraint solving rules for $\sharp$ are shown in Figure 3.27. The rule $\mathcal{S}_{\rhd,\sharp}$ is analogous to $\mathcal{S}_{\rhd,*}$ and directly introduces provides constraint for the premises of the construction rule for $\sharp$ instead of introducing a node constraint for the rule itself. The rule $\mathcal{S}_{\twoheadrightarrow,\sharp}$ solves chains that start at conclusions $\mathsf{K}^{\Downarrow^{\mathsf{d}}}(a\sharp b)$. We therefore modify $\mathcal{S}_{\twoheadrightarrow}$ by adding the additional side condition that $(concs(ri))_1 \neq \mathsf{K}^{\Downarrow^{\mathsf{d}}}(a\sharp b)$ for all terms a and b. The new rule $\mathcal{S}_{\twoheadrightarrow,\sharp}$ handles this case by adding one case for every element of $a\sharp b$ that is not a message variable. The rule is only applicable if all elements of $a\sharp b$ that are message variables are known before. The rule exploits three normal form conditions. Condition **N5** allows us to ignore all cases where a message that is already $\mathsf{K}^{\Uparrow}$-known is extracted. Condition **N7** allows us to ignore the Coerce case since $\mathsf{K}^{\Uparrow}(a\sharp b)$ is never the conclusion of Coerce. Finally, condition **N8** allows us to ignore all cases where a term of the form $a\sharp b$ is extracted. The constraint solving

$$\mathcal{S}_{\mathbf{N9}}\colon\ \frac{i\colon \big[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(a), \mathsf{K}^{\Uparrow}(b)\big] \mathrel{-\!\![]\!\!\rightarrow} \mathsf{K}^{\Downarrow^{\mathsf{e}}}([d]c)}{\bot}\,\mathcal{I}$$

if $vars(c) \subseteq \mathcal{V}_{pub}$, $St(c) \cap \mathsf{FN} = \emptyset$, and $nifactors(d) \subseteq nifactors(b)$

$$\mathcal{S}_{\mathbf{N10}}\colon\ \frac{\begin{array}{l}(j,1) \rightarrowtail (i,1),\\ i\colon \big[\mathsf{K}^{\Downarrow^{\mathsf{d}}}([t_1]p), \mathsf{K}^{\Downarrow^{\mathsf{d}}}([t_2]q)\big] \mathrel{-\!\![]\!\!\rightarrow} \mathsf{K}^{\Downarrow^{\mathsf{e}}}(\hat{e}(p,q)\hat{\ }c),\\ j\colon \big[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\hat{e}(p,q)\hat{\ }c), \mathsf{K}^{\Uparrow}(d)\big] \mathrel{-\!\![]\!\!\rightarrow} \mathsf{K}^{\Downarrow^{\mathsf{e}}}(\hat{e}(p,q)\hat{\ }e)\end{array}}{\bot}\,\mathcal{I}$$

if $vars(p,q) \subseteq \mathcal{V}_{pub}$, $St(p,q) \cap \mathsf{FN} = \emptyset$, and $nifactors(t_i) \subseteq nifactors(d)$ for $i = 1$ or $i = 2$.

$$\mathcal{S}_{\mathbf{N11}}\colon\ \frac{\begin{array}{l} j\colon \big[\mathsf{K}^{\Downarrow^{\mathsf{d}}}([a]p), \mathsf{K}^{\Downarrow^{\mathsf{d}}}([b]q)\big] \mathrel{-\!\![]\!\!\rightarrow} \mathsf{K}^{\Downarrow^{\mathsf{e}}}(\hat{e}(p,q)\hat{\ }(a{*}b)),\\ k_1\colon ri_1, k_2\colon ri_2,\\ i_1\colon ru_1, (i_1,u_1) \rightarrowtail (k_1,1), k_1 \twoheadrightarrow (j,1),\\ i_2\colon ru_2, (i_2,u_2) \rightarrowtail (k_2,1), k_2 \twoheadrightarrow (j,2)\end{array}}{\bot}\,\mathcal{I}$$

if ri_1 and ri_2 are instances of RECV and $fsyms(ru_2) >_{fs} fsyms(ru_2)$

Figure 3.28. Constraint solving rules that ensure **N9–11**.

rules ensuring the new normal form conditions are shown in Figure 3.28. The other normal form conditions are maintained as invariants. The resulting constraint solving relation is sound and complete, which we prove in Appendix A.2, and we can still obtain P-models from solved constraint systems.

Analyzing the Joux protocol

The key computation in our Joux model applies $\hat{e}$ to two message variables and exponentiates the result. Since the rewriting system $\mathcal{RBP}_e$ is also more complicated than $\mathcal{RDH}_e$, it is not surprising that the rules for Joux have more variants than the rules for UM. More precisely, the responder rule of the UM protocol has 6 variants while the second step of the Joux protocol has 160 variants. Nevertheless, the derived constraint rewriting rule for exponentiation given in Figure 3.29 is comparable to the one for UM. The rule states that there are five ways to deduce an exponentiation for the adversary captured by the cases Δ_i.

1. He can construct the exponentiation himself if he knows the base u, which cannot be an exponentiation itself, and all non-inverse factors of the exponent v.
2. He can apply the bilinear map $\hat{e}$ to a scalar multiplication $[x{:}fr]\mathsf{P}$, extracted from the message sent in the first protocol step, and an arbitrary message q that is not a scalar multiplication. If q is a scalar multiplication $[a]b$, then the result $\hat{e}(\mathsf{P},[a]b)\hat{\ }(x{:}fr)$ of applying $\hat{e}$ is not $\downarrow_{\mathcal{RBP}_e}$-normal.
3. He can extract the scalar multiplications $[x{:}fr]\mathsf{P}$ and $[y{:}fr]\mathsf{P}$ from two protocol sends and apply the bilinear map $\hat{e}$ to both.
4. He can perform the same steps as in 2 and then use an exponentiation rule to multiply the exponent of the message deduced in 2 with z.
5. He can perform the same steps as in 3 and then use an exponentiation rule to multiply the exponent of the message deduced in 3 with z.

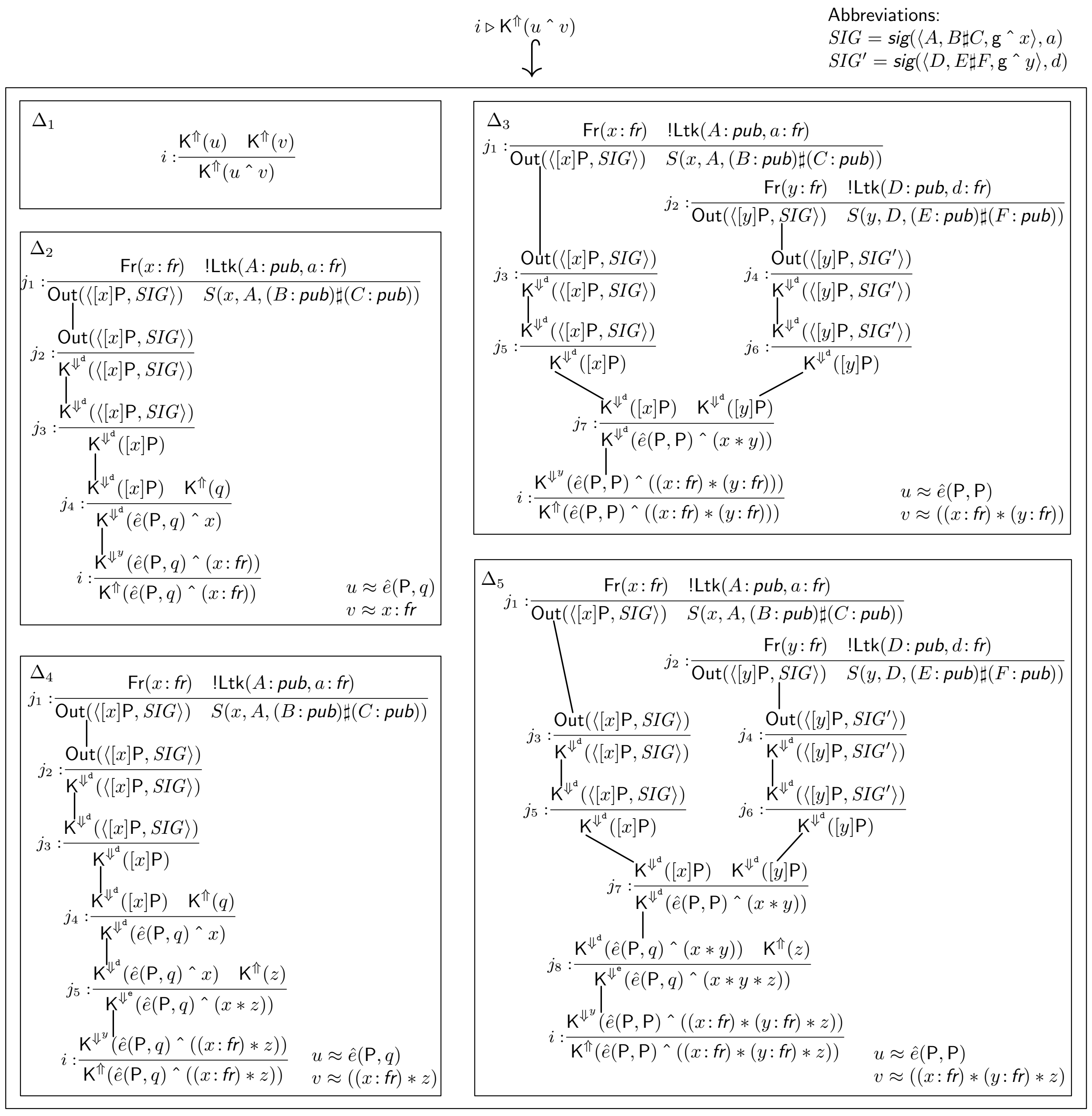

$i \triangleright \mathsf{K}^{\Uparrow}(u \,\hat{}\, v)$
Abbreviations:
$SIG = sig(\langle A, B\sharp C, g \,\hat{}\, x\rangle, a)$
$SIG' = sig(\langle D, E\sharp F, g \,\hat{}\, y\rangle, d)$
Δ_1
Δ_2
$u \approx \hat{e}(\mathsf{P}, q)$
$v \approx x{:}fr$
Δ_3
$u \approx \hat{e}(\mathsf{P}, \mathsf{P})$
$v \approx ((x{:}fr) * (y{:}fr))$
Δ_4
$u \approx \hat{e}(\mathsf{P}, q)$
$v \approx ((x{:}fr) * z)$
Δ_5
$u \approx \hat{e}(\mathsf{P}, \mathsf{P})$
$v \approx ((x{:}fr) * (y{:}fr) * z)$

Figure 3.29. Constraint rewriting rules for *SIGJOUX*.

We will present the results and runtimes for the automated verification with the Tamarin prover in the next section. The automated analysis roughly proceeds as follows. First, the signatures are used to deduce that the session key of the test session has the form

$$\hat{e}(\mathsf{P},\mathsf{P})\,\hat{}\,((x{:}fr)*(y{:}fr)*(z{:}fr)).$$

Then, the above constraint rewriting rule is used to reason about all possible ways to deduce this exponentiation. To finish the proof, all resulting cases are shown to be contradictory. For example, in the first case corresponding to Δ_1, the adversary performs an exponentiation with base $\hat{e}(\mathsf{P},\mathsf{P})$ and exponent $(x{:}fr)*(y{:}fr)*(z{:}fr)$. Hence, he must know x, y, and z, which is shown to be impossible.

3.5 Implementation and Case Studies

In this section, we first describe our implementation of the constraint solving algorithm in a tool called the TAMARIN prover. Then, we describe two additional case studies in detail: the security of NAXOS with respect to the eCK model and the security of the RYY protocol with respect to a model of wPFS with session key reveals and long-term key reveals. Finally, we present an evaluation of the TAMARIN prover on numerous protocols.

3.5.1 The TAMARIN prover

The TAMARIN prover implements a constraint solving algorithm based on $\rightsquigarrow_P$ to analyze security protocols. Figure 3.30 depicts TAMARIN's workflow for proving the validity of security properties. The octagons denote computations and the rectangles denote inputs, outputs, and intermediate values of the algorithm. We use edges to denote the arguments and results of computations, but leave some arguments implicit, e.g., the equational theory is used by all computations.

TAMARIN takes the following inputs:

1. An equational theory $\mathcal{E}$ that is specified by giving (a) function symbols and subterm-convergent equations for $\mathcal{ST}$ and (b) boolean flags to denote whether the (built-in) function symbols and equations for Diffie-Hellman exponentiation, bilinear pairing, and multiset union should be included
2. A protocol P specified as a set of multiset rewriting rules.
3. A sequence $\vec{\psi}$ of axioms specified as guarded trace properties.
4. A sequence $\vec{\varphi}$ of security properties that the user wants to prove specified as guarded trace properties.

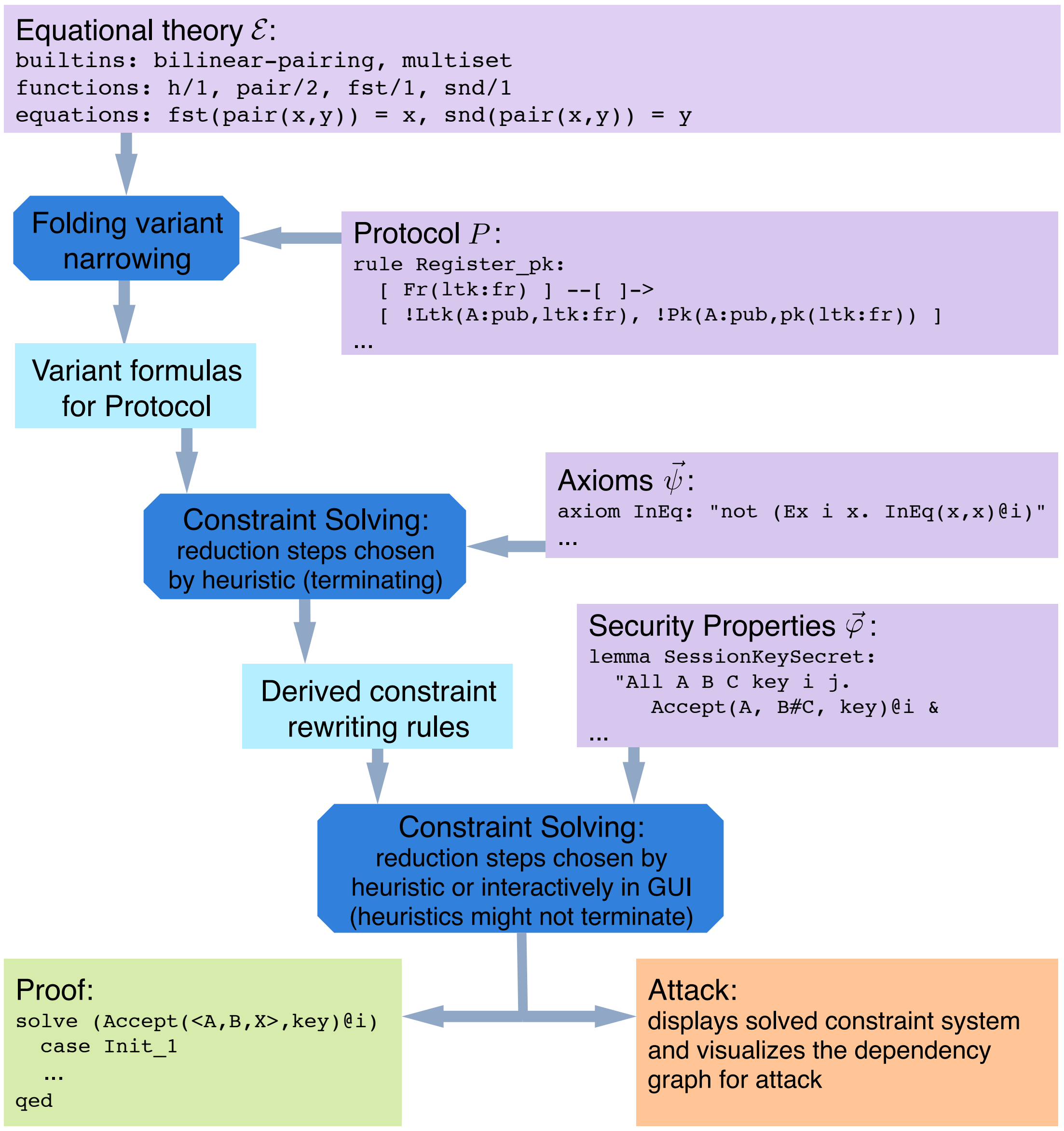
Then TAMARIN checks whether

$$\mathit{trace}(\mathit{execs}_{\mathcal{E}}(P \cup \mathit{MD}_{\mathcal{E}})) \vDash_{\mathcal{E}} \left(\bigwedge_{i=1}^{|\vec{\psi}|} \psi_i \right) \Longrightarrow \left(\bigwedge_{j=1}^{|\vec{\varphi}|} \varphi_j \right)$$

holds, where $\mathit{MD}_{\mathcal{E}}$ denotes the message deduction rules for $\mathcal{E}$, $\mathit{execs}_{\mathcal{E}}(P \cup \mathit{MD}_{\mathcal{E}})$ denotes the executions for $P \cup \mathit{MD}_{\mathcal{E}}$ defined via multiset rewriting modulo $\mathcal{E}$, and $\vDash_{\mathcal{E}}$ denotes the semantics of trace formulas modulo $\mathcal{E}$. If TAMARIN terminates, it either returns a proof or a counterexample.

To achieve this, TAMARIN performs the following steps:

1. It computes a finite variant decomposition $\mathcal{R}$,$\mathcal{AX}$ from the given specification of $\mathcal{E}$. Then, it uses folding variant narrowing to compute the complete $\mathcal{R}$,$\mathcal{AX}$-variant-formulas for the protocol rules in P. It also computes the normal message deduction rules $\mathit{ND}_{\mathcal{E}}$ from $\mathcal{E}$. The normal message deduction rules for Diffie-Hellman and bilinear pairing are usually precomputed and loaded from disk.
2. It uses constraint solving with $\rightsquigarrow_P^{\text{ctx}}$ to precompute constraint rewriting rules for protocol facts, the provides constraint $i \rhd (x{:}\mathit{fr})$, and provides constraints $i \rhd g(x_1, \ldots, x_k)$ for all function symbols g. Here, the axioms $\vec{\psi}$ are already taken into account and the previously computed variant-formulas for P are used. This precomputation of constraint


Equational theory $\mathcal{E}$:
Folding variant narrowing
Protocol P:
Variant formulas for Protocol
Axioms $\vec{\psi}$:
Constraint Solving: reduction steps chosen by heuristic (terminating)
Security Properties $\vec{\varphi}$:
Derived constraint rewriting rules
Constraint Solving: reduction steps chosen by heuristic or interactively in GUI (heuristics might not terminate)
Proof:
Attack: displays solved constraint system and visualizes the dependency graph for attack

```
builtins: bilinear-pairing, multiset
functions: h/1, pair/2, fst/1, snd/1
equations: fst(pair(x,y)) = x, snd(pair(x,y)) = y
```

```
rule Register_pk:
  [ Fr(ltk:fr) ] --[ ]->
  [ !Ltk(A:pub,ltk:fr), !Pk(A:pub,pk(ltk:fr)) ]
...
```

```
axiom InEq: "not (Ex i x. InEq(x,x)@i)"
...
```

```
lemma SessionKeySecret:
  "All A B C key i j.
     Accept(A, B#C, key)@i &
...
```

```
solve (Accept(<A,B,X>,key)@i)
  case Init_1
  ...
qed
```

Figure 3.30. Tamarin workflow for proving validity of security properties.

rewriting rules is carefully designed to prevent nontermination or an explosion of cases, while still yielding useful constraint rewriting rules.

3. For each φ_j, it uses constraint solving with $\rightsquigarrow_P$ and the precomputed constraint rewriting rules to search for a P-model of the constraint system

$$\Gamma_0 := \left\{ \left(\bigwedge_{i=1}^{|\vec{\psi}|} \psi_i \right) \wedge gf(\neg\varphi_j) \right\},$$

where gf denotes a function that rewrites a given formula into a guarded trace formula. If the search fails, the tree of constraint solving steps constitutes a proof of φ_j. To search for P-models, it uses the approach described in Section 3.3.2 extended with some additional bookkeeping. In the following, we call applications of both the constraint solving rules from $\rightsquigarrow_P$ and the precomputed constraint rewriting rules constraint solving steps. If the state of the algorithm is the set of constraint systems $\{\Gamma_1, \ldots, \Gamma_k\}$ where none of the constraint systems is solved, then it must make two choices to determine the next step.

a) It must choose the constraint system Γ_i which will be reduced next. This choice is mostly relevant if φ_j is *not* valid since it is possible to stop when the *first* solved constraint system is encountered. To obtain a proof of φ_j, the algorithm must reduce *all* constraint system to the empty set of constraint systems.

b) Given a constraint system Γ_i, the algorithm must choose one of the applicable constraint solving steps. Here, the main goal is to make progress and to add constraints that eventually lead to a contradiction. Another goal is to prevent duplicated work by delaying case distinctions if possible .

Tamarin provides an automated mode where this choice is performed by a heuristic and an interactive mode where a GUI (depicted in Figure 3.31) presents the possible choices to the user. For both modes, trivial constraint solving steps that directly lead to contradictions or simplify constraint systems without leading to multiple cases are applied eagerly.

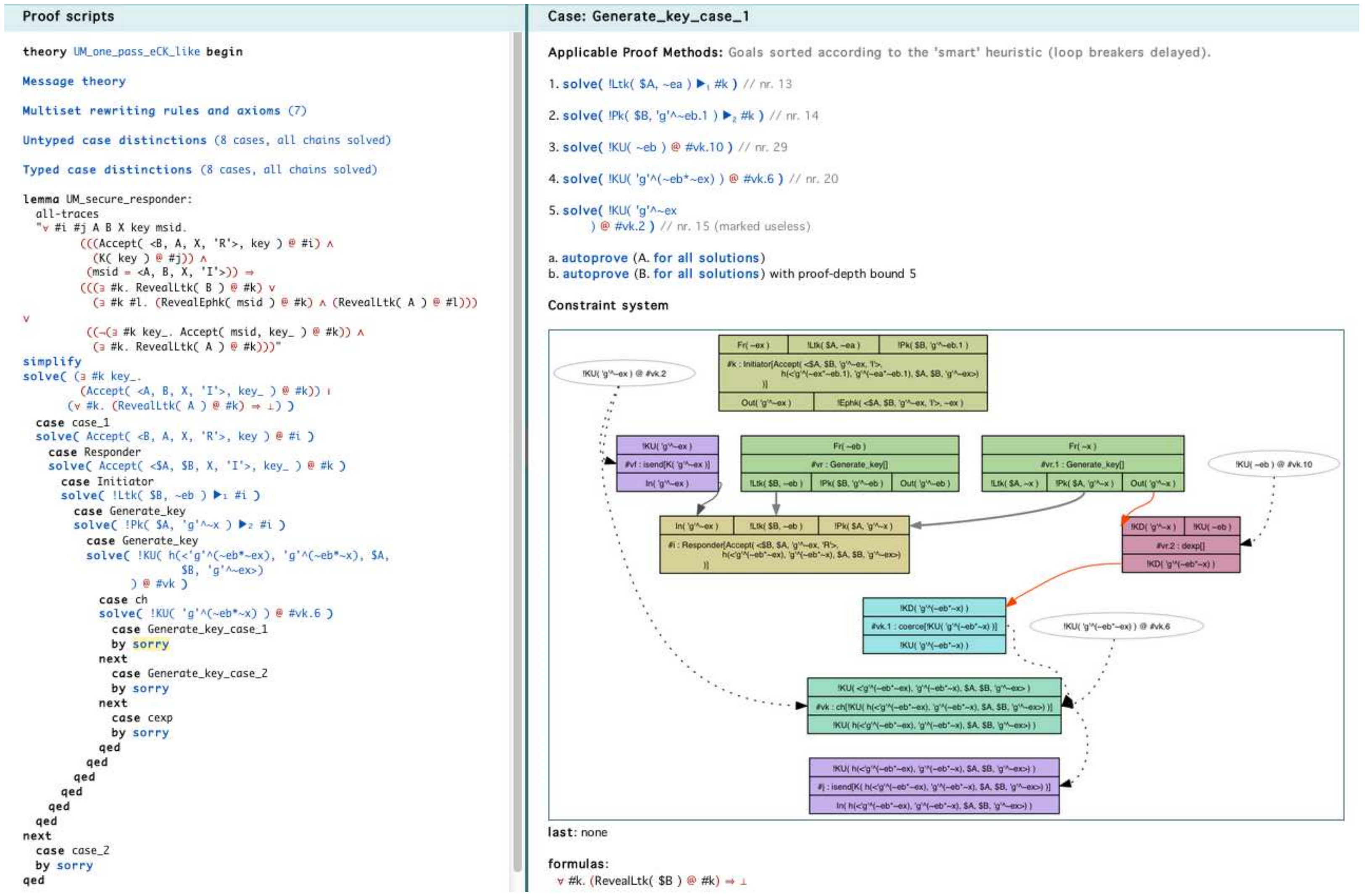
For choosing between non-trivial steps, our heuristic takes both previously described goals into account. For example, provides constraints $i \triangleright \mathsf{K}^{\Uparrow}(m)$ are delayed if the message m is earlier sent by the protocol since solving such constraints rarely leads to contradictions. On the other hand, provides constraints $i \triangleright \mathsf{K}^{\Uparrow}(k)$ for secret keys k are usually solved early on since they often lead to contradictions.

Even though we will demonstrate in Section 3.5.4 that the heuristics usually performs well and we could, ignoring efficiency, use iterative deepening to explore *all* constraint reductions up to a given size, the GUI is still very useful. First, we can perform automated proof search in the GUI and explore the resulting constraint reduction tree, including attacks, if there are any. Second, if the automated proof search takes too long or does not terminate, we can use the GUI to investigate the reason or to override the heuristics.

For more details about Tamarin, we refer to its documentation, the included case studies, and Meier's thesis [124], which describes the computation of constraint rewriting rules and the heuristics employed by Tamarin in more detail. Note that the initial version of the GUI was developed by Staub and is described in his Bachelor thesis [163]. In Section 3.5.5, we will discuss limitations of the method. In this context, we will describe two extensions developed by Meier [124] that allow Tamarin to handle protocols with loops and protocols that perform blind forwarding of messages received in encryptions.

3.5.2 Security of NAXOS in the eCK Model

The NAXOS protocol [111] depicted in Figure 3.32 has been designed to be secure in the eCK [111] model. Before starting the NAXOS protocol, the agents $\mathcal{A}$ and $\mathcal{B}$ know their own private long-term key a and b and the other agent's long-term public key $\hat{\mathcal{B}} = g^b$ and $\hat{\mathcal{A}} = g^a$, respectively. Then, they generate ephemeral private keys x and y and combine them with their long-term private keys to compute the ephemeral public keys $g^{h_1(x,a)}$ and $g^{h_1(y,b)}$. After exchanging these ephemeral public keys, both agents can compute the shared secret $h_2(g^{a h_1(y,b)}, g^{b h_1(x,a)}, g^{h_1(y,b) h_1(x,a)}, \mathcal{A}, \mathcal{B})$. The basic idea for the security is that if the adversary was passive and the key is computed as expected, then *either* x and a



Proof scripts

```
theory UM_one_pass_eCK_like begin

Message theory

Multiset rewriting rules and axioms (7)

Untyped case distinctions (8 cases, all chains solved)

Typed case distinctions (8 cases, all chains solved)

lemma UM_secure_responder:
  all-traces
  "∀ #i #j A B X key msid.
        (((Accept( <B, A, X, 'R'>, key ) @ #i) ∧
          (K( key ) @ #j)) ∧
         (msid = <A, B, X, 'I'>)) ⇒
        (((∃ #k. RevealLtk( B ) @ #k) ∨
          (∃ #k #l. (RevealEphk( msid ) @ #k) ∧ (RevealLtk( A ) @ #l)))
∨
         ((¬(∃ #k key_. Accept( msid, key_ ) @ #k)) ∧
          (∃ #k. RevealLtk( A ) @ #k)))"
simplify
solve( (∃ #k key_.
        (Accept( <A, B, X, 'I'>, key_ ) @ #k)) ∥
       (∀ #k. (RevealLtk( A ) @ #k) ⇒ ⊥) )
  case case_1
  solve( Accept( <B, A, X, 'R'>, key ) @ #i )
    case Responder
    solve( Accept( <$A, $B, X, 'I'>, key_ ) @ #k )
      case Initiator
      solve( !Ltk( $B, ~eb ) ▶₁ #i )
        case Generate_key
        solve( !Pk( $A, 'g'^~x ) ▶₂ #i )
          case Generate_key
          solve( !KU( h(<'g'^(~eb*~ex), 'g'^(~eb*~x), $A,
                         $B, 'g'^~ex>)
                 ) @ #vk )
            case ch
            solve( !KU( 'g'^(~eb*~x) ) @ #vk.6 )
              case Generate_key_case_1
              by sorry
            next
              case Generate_key_case_2
              by sorry
            next
              case cexp
              by sorry
            qed
          qed
        qed
      qed
    qed
  qed
next
  case case_2
  by sorry
qed
```

Case: Generate_key_case_1

Applicable Proof Methods: Goals sorted according to the 'smart' heuristic (loop breakers delayed).

1. solve(!Ltk($A, ~ea) ▶₁ #k) // nr. 13
2. solve(!Pk($B, 'g'^~eb.1) ▶₂ #k) // nr. 14
3. solve(!KU(~eb) @ #vk.10) // nr. 29
4. solve(!KU('g'^(~eb*~ex)) @ #vk.6) // nr. 20
5. solve(!KU('g'^~ex
) @ #vk.2) // nr. 15 (marked useless)

a. autoprove (A. for all solutions)
b. autoprove (B. for all solutions) with proof-depth bound 5

Constraint system

Fr(~ex) | !Ltk($A, ~ea) | !Pk($B, 'g'^~eb.1)
#k : Initiator[Accept(<$A, $B, 'g'^~ex, 'I'>, h(<'g'^(~ex*~eb.1), 'g'^(~ea*~eb.1), $A, $B, 'g'^~ex>))]
Out('g'^~ex) | !Ephk(<$A, $B, 'g'^~ex, 'I'>, ~ex)
!KU('g'^~ex) @ #vk.2
!KU('g'^~ex)
#vf : isend[K('g'^~ex)]
In('g'^~ex)
Fr(~eb)
#vr : Generate_key[]
!Ltk($B, ~eb) | !Pk($B, 'g'^~eb) | Out('g'^~eb)
Fr(~x)
#vr.1 : Generate_key[]
!Ltk($A, ~x) | !Pk($A, 'g'^~x) | Out('g'^~x)
!KU(~eb) @ #vk.10
In('g'^~ex) | !Ltk($B, ~eb) | !Pk($A, 'g'^~x)
#i : Responder[Accept(<$B, $A, 'g'^~ex, 'R'>, h(<'g'^(~eb*~ex), 'g'^(~eb*~x), $A, $B, 'g'^~ex>))]
!KD('g'^~x) | !KU(~eb)
#vr.2 : dexp[]
!KD('g'^(~eb*~x))
!KD('g'^(~eb*~x))
#vk.1 : coerce[!KU('g'^(~eb*~x))]
!KU('g'^(~eb*~x))
!KU('g'^(~eb*~ex)) @ #vk.6
!KU(<'g'^(~eb*~ex), 'g'^(~eb*~x), $A, $B, 'g'^~ex>)
#vk : ch[!KU(h(<'g'^(~eb*~ex), 'g'^(~eb*~x), $A, $B, 'g'^~ex>))]
!KU(h(<'g'^(~eb*~ex), 'g'^(~eb*~x), $A, $B, 'g'^~ex>))
!KU(h(<'g'^(~eb*~ex), 'g'^(~eb*~x), $A, $B, 'g'^~ex>))
#j : isend[K(h(<'g'^(~eb*~ex), 'g'^(~eb*~x), $A, $B, 'g'^~ex>))]
In(h(<'g'^(~eb*~ex), 'g'^(~eb*~x), $A, $B, 'g'^~ex>))

last: none

formulas:
∀ #k. (RevealLtk($B) @ #k) ⇒ ⊥

Figure 3.31. TAMARIN visualizes a proof attempt of the security of P_{UM}. The visited constraint systems and applied constraint solving steps are shown in the tree on the left which can also be used for navigation. The currently considered constraint system which corresponds to $\Gamma_{1.1.1.1}$ in Figure 3.19 is shown on the right. On top, the applicable constraint solving steps are displayed. In this case, we can choose method 3 to show that the current constraint system is contradictory.

or y and b are required to compute the key. If the adversary was active and $\mathcal{A}$ receives a message Y that was not created by the (honest) intended partner, then *either* b *or* x and a must be known to compute $\hat{\mathcal{B}}^{h_1(x,a)} = g^{bh_1(x,a)}$.

We formalize NAXOS security in the eCK model with the protocol rules and guarded trace property given in Figures 3.33 and 3.34. Intuitively, the adversary queries are modeled by the rules and the winning condition, including clean, is formalized by the property.

$$\text{Key Generation:}\quad \frac{\mathsf{Fr}(a{:}\mathit{fr})}{!\mathsf{Ltk}(A{:}\mathit{pub}, a)\quad !\mathsf{Pk}(A{:}\mathit{pub}, \mathsf{g}\hat{\ }a)\quad \mathsf{Out}(\mathsf{g}\hat{\ }a)}$$

$$\text{Initiator 1:}\quad \frac{\mathsf{Fr}(x{:}\mathit{fr})\quad !\mathsf{Ltk}(A, a{:}\mathit{fr})}{\mathsf{Out}(\mathsf{g}\hat{\ }h_1(x,a))\quad \mathsf{Init}(x, A{:}\mathit{pub}, B{:}\mathit{pub}, a)\quad !\mathsf{Ephk}(x)}$$

$$\text{Initiator 2:}\quad \frac{\begin{array}{l}\mathsf{Init}(x{:}\mathit{fr}, A{:}\mathit{pub}, B{:}\mathit{pub}, a{:}\mathit{fr})\\ \mathsf{In}(Y)\quad !\mathsf{Pk}(B, \bar{B})\end{array}}{!\mathsf{Sessk}(x, \mathit{key})}\begin{bmatrix}\mathsf{Accept}(x, \mathit{key})\\ \mathsf{Sid}(x, \langle A, B, \mathsf{g}\hat{\ }h_1(x,a), Y, \mathcal{I}\rangle)\end{bmatrix}$$

$$\text{where}\quad \begin{array}{l}\bar{B} = \mathsf{g}\hat{\ }(b{:}\mathit{fr})\\ \mathit{key} = h_2(Y\hat{\ }a, \bar{B}\hat{\ }h_1(x,a), Y\hat{\ }h_1(x,a), A, B)\end{array}$$

$$\text{Responder 1:}\quad \frac{\begin{array}{l}\mathsf{Fr}(y{:}\mathit{fr})\quad !\mathsf{Ltk}(B{:}\mathit{pub}, b{:}\mathit{fr})\\ \mathsf{In}(X)\quad !\mathsf{Pk}(A{:}\mathit{pub}, \bar{A})\end{array}}{\begin{array}{l}\mathsf{Out}(\mathsf{g}\hat{\ }h_1(y,b))\quad !\mathsf{Ephk}(y)\\ !\mathsf{Sessk}(y, \mathit{key})\end{array}}\begin{bmatrix}\mathsf{Accept}(y, \mathit{key})\\ \mathsf{Sid}(y, \langle B, A, X, \mathsf{g}\hat{\ }h_1(y,b), \mathcal{R}\rangle)\end{bmatrix}$$

$$\text{where}\quad \begin{array}{l}\bar{A} = \mathsf{g}\hat{\ }(a{:}\mathit{fr})\\ \mathit{key} = h_2(\bar{A}\hat{\ }h_1(y,b), X\hat{\ }b, X\hat{\ }h_1(y,b), A, B)\end{array}$$

$$\text{Long-term key reveal:}\quad \frac{!\mathsf{Ltk}(A, a)}{\mathsf{Out}(a)}[\,\mathsf{RevealLtk}(A)\,]$$

$$\text{Session key reveal:}\quad \frac{!\mathsf{Sessk}(s, k)}{\mathsf{Out}(k)}[\,\mathsf{RevealSessk}(s)\,]$$

$$\text{Ephemeral key reveal:}\quad \frac{!\mathsf{Ephk}(x)}{\mathsf{Out}(x)}[\,\mathsf{RevealEphk}(x)\,]$$

Figure 3.33. Rules defining the NAXOS protocol in the eCK model.

To model the hash functions, we use free function symbols h_1 and h_2. We model key generation with a key generation rule like in the UM example. Note that some case studies presented in the next section also allow for key registration by the adversary. The first step of an initiator A generates a fresh ephemeral private key x, which is also used to identify the session, and looks up A's long-term private key a. Then, the ephemeral public key is sent and the session's state is stored using an Init-fact. Additionally, a fact $!\mathsf{Ephk}(x)$ is created to allow the adversary to reveal the ephemeral private key x of the session identified by x. The second initiator step consumes an Init-fact to obtain the session's ephemeral private key x, the actor's identity A, the peer's identity B, and the actor's long-term private key a.

$\mathcal{A}$		$\mathcal{B}$
long-term key pair: $\big(a, \hat{\mathcal{A}} = g^a\big)$		long-term key pair: $\big(b, \hat{\mathcal{B}} = g^b\big)$
choose random exponent x		
$X := g^{h_1(x,a)}$	$\xrightarrow{X}$	choose random exponent y
	$\xleftarrow{Y}$	$Y := g^{h_1(y,b)}$
compute $h_2\big(Y^a, \hat{\mathcal{B}}^{h_1(x,a)}, Y^{h_1(x,a)}, \mathcal{A}, \mathcal{B}\big)$		compute $h_2\big(\hat{\mathcal{A}}^{h_1(y,b)}, X^b, X^{h_1(y,b)}, \mathcal{A}, \mathcal{B}\big)$

Figure 3.32. The NAXOS protocol.

Additionally, it receives the message Y and looks up B's long-term public key $\bar{B}$. Using these, the session computes the session key, creates a !Sessk-fact for session key reveals, denotes that it accepted a key using an Accept-action, and associates the session identifier x with the given session string using an Sid-action. The responder rule is defined analogously. The remaining three rules model long-term key reveal, session key reveal, and ephemeral key reveal.

```
$\forall\, key\ test\ A\ B\ X\ Y\ R.$
  // If there is a test session $test$ whose session key is known to the adversary,
  $\mathsf{Accept}(test, key) \wedge \mathsf{K}(key) \wedge \mathsf{Sid}(test, \langle A, B, X, Y, R\rangle)$
  // then one of the following must have happened:
  $\Longrightarrow$ (// 1. The adversary revealed $test$'s session key.
        $\mathsf{RevealSessk}(test)$

      // 2. The adversary revealed the long-term key of $test$'s actor and
      //    $test$'s ephemeral key.
      $\vee\ (\mathsf{RevealLtk}(A) \wedge \mathsf{RevealEphk}(test))$

      // 3. There is a matching session $match$ and the adversary revealed
      $\vee\ (\exists\, match\ R'.\ \mathsf{Sid}(match, \langle B, A, Y, X, R'\rangle) \wedge \neg(R \approx R')$
         $\wedge$ (// (a) $match$'s session key, or
              $\mathsf{RevealSessk}(match)$

            // (b) the long-term key of $test$'s peer and $match$'s ephemeral key.
            $\vee\ \mathsf{RevealLtk}(B) \wedge \mathsf{RevealEphk}(match)))$

      // 4. There is no matching session and the adversary revealed
      $\vee\ (\neg(\exists\, match\ R'.\ \mathsf{Sid}(match, \langle B, A, Y, X, R'\rangle) \wedge \neg(R \approx R'))$
            // the long-term key of $test$'s peer.
            $\wedge\ \mathsf{RevealLtk}(B)))$
```

Figure 3.34. eCK security property for the NAXOS protocol.

The security property states that whenever the adversary knows the key of a session $test$, then the session is not clean. The definition of clean is a direct translation of the definition in Section 2.1.2.4. Note that in contrast to the computational definition, our symbolic model captures computability of the session key instead of indistinguishability.

3.5.3 wPFS Security of RYY

The formalization of RYY given in Figures 3.35 and 3.36 follows the description in Section 2.1.3.3. We use the free function symbol H to model the hash function that hashes into $\mathbb{G}_1$ and the free function symbol h to model the hash function used as the key derivation function. The rules in the first line model a key generation center that first creates a fresh master secret key msk and then allows owners of identities to obtain their long-term private keys. Since the public key for an identity A is just $H(A)$, there is no need for public key distribution. The first step of the initiator sends the ephemeral public key $\mathsf{g}\hat{}x$. The second step of the initiator receives the ephemeral public key of the responder and computes the session key by combining the ephemeral DH key and the static identity-based key. We also include the exchanged messages and the identities of the participants in the input to the hash function. We use Accept-actions and Sid-actions to denote the accepted key and the session string. The responder rule is defined analogously. The remaining rules handle long-term key reveal, session key reveal, and master key reveal.

Key Generation Center: $$\frac{\mathsf{Fr}(msk{:}fr)}{!\mathsf{Msk}(msk)} \qquad \frac{!\mathsf{Msk}(msk)}{!\mathsf{Ltk}(ID{:}\,pub, [msk]H(ID))}$$

Initiator step 1: $$\frac{\mathsf{Fr}(x{:}fr)}{\mathsf{Out}(\mathsf{g}\hat{}x) \quad \mathsf{Init}(x, A{:}pub, B{:}pub)}$$

Initiator step 2: $$\frac{\mathsf{Init}(x, A{:}pub, B{:}pub) \quad \mathsf{In}(Y) \quad !\mathsf{Ltk}(A, skA)}{!\mathsf{Sessk}(x, key)}\left[\begin{array}{l}\mathsf{Accept}(x, key)\\ \mathsf{Sid}(x, \langle A, B, \mathsf{g}\hat{}x, Y, \mathcal{A}\rangle)\end{array}\right]$$
$$\text{where} \quad key = h(Y\hat{}x, \hat{e}(H(B), skA), A, B, \mathsf{g}\hat{}x, Y)$$

Responder step 1: $$\frac{\mathsf{Fr}(y{:}fr) \quad \mathsf{In}(X) \quad !\mathsf{Ltk}(B{:}pub, skB)}{\mathsf{Out}(\mathsf{g}\hat{}y) \quad !\mathsf{Sessk}(y, key)}\left[\begin{array}{l}\mathsf{Accept}(y, key)\\ \mathsf{Sid}(y, \langle B, A, X, \mathsf{g}\hat{}y, \mathcal{B}\rangle)\end{array}\right]$$
$$\text{where} \quad key = h(X\hat{}y, \hat{e}(H(A), skB), A, B, X, \mathsf{g}\hat{}y)$$

Long-Term key reveal: $$\frac{!\mathsf{Ltk}(A, a)}{\mathsf{Out}(a)}[\,\mathsf{RevealLtk}(A)\,]$$

Session key reveal: $$\frac{!\mathsf{Ltk}(s, k)}{\mathsf{Out}(k)}[\,\mathsf{RevealSessk}(s)\,]$$

Master key reveal: $$\frac{!\mathsf{Ltk}(A, a)}{\mathsf{Out}(a)}[\,\mathsf{RevealMsk}()\,]$$

Figure 3.35. Rules defining the RYY protocol.

The adversary model formalized by the security property allows the adversary to reveal session keys of unrelated sessions, to reveal long-term keys of the participants (or the master-key) after the test session has accepted the key *if* he was passive, and to reveal long-term keys of other agents. As we will demonstrate in the next section, Tamarin finds an attack if we add ephemeral key reveals to this model.

```
$\forall i\ key\ test\ A\ B\ X\ Y\ R.$
    // If there is a test session test whose session key is known to the adversary,
    $\mathsf{Accept}(test, key)@i \wedge \mathsf{K}(key) \wedge \mathsf{Sid}(test, \langle A, B, X, Y, R\rangle)$

    // then one of the following must have happened.
    $\Longrightarrow$ // 1. The adversary revealed test's session key.
        $\mathsf{RevealSessk}(test)$

        // 2. There is a matching session match and the adversary revealed
        $\vee\ (\exists\ match\ R'. \mathsf{Sid}(match, \langle B, A, Y, X, R'\rangle) \wedge \neg(R \approx R')$
            $\wedge$ ( // (a) match's session key, or
                $\mathsf{RevealSessk}(match)$
                // (b) the long-term key of test's peer before i, or
                $\vee\ (\exists m. \mathsf{RevealLtk}(B)@m \wedge m \prec i)$
                // (c) the long-term key of test's actor before i, or
                $\vee\ (\exists m. \mathsf{RevealLtk}(A)@m \wedge m \prec i)$
                // (d) the master key before i.
                $\vee\ (\exists m. \mathsf{RevealMsk}()@m \wedge m \prec i)))$

        // 3. There is no matching session and the adversary revealed
        $\vee\ (\neg(\exists l\ match\ R'. \mathsf{Sid}(match, \langle B, A, Y, X, R'\rangle)@l \wedge \neg(R \approx R'))$
            $\wedge$ // the long-term key of test's peer, of test's actor, or the master key.
              $(\mathsf{RevealLtk}(B) \vee \mathsf{RevealLtk}(A) \vee \mathsf{RevealMsk}()\ ))$
```

Figure 3.36. Security property for the RYY protocol

3.5.4 Experimental Results

We applied Tamarin to three type of case studies. First, we analyzed a number of two-pass AKE protocols with respect to different adversary models. The results, obtained on a laptop with a 2.9 Ghz Intel Core i7 processor, are listed in Table 3.1. For each protocol, we formalized its intended and related adversary models and analyzed them using Tamarin. We analyzed NAXOS with respect to the formalization in Section 3.5.2. For a modified adversary model that accounts for Perfect Forward Secrecy (PFS), Tamarin discovered an attack. We modeled NIST's KAS1 and KAS2 protocols [102] and the related DH2 protocol by Chatterjee et al. [43] which uses inverses. For these protocols, our analysis confirms the security proofs and the informal statements made in [43]. We also analyzed the SIGDH protocol from Section 2.1.2 and the KEA+ protocol [112] and obtained the expected results. To verify Key Independence (KI) for KEA+ and STS, we model that the adversary can reveal certain keys. For STS, we additionally allow the adversary to register arbitrary public keys for corrupted agents. In this setting, we find the UKS attack reported in [27]. We model and successfully verify both fixes from [27]. The first fix is to require a Proof-of-Possession of the private key for key registration and the second fix is to include the identities of the participants in the signatures. We also analyzed the TS1, TS2, and TS3 protocols from [95] and their updated versions [96].

	Protocol	Adversary Model	Result	Time [s]
1.	NAXOS	eCK	proof	4.1
2.	NAXOS	eCK^{PFS}	attack	3.4
3.	KAS1	KI+KCI	proof	0.7
4.	KAS2	weakened eCK	proof	5.5
5.	KAS2	eCK	attack	3.6
6.	DH2	weakened eCK	proof	21.7
7.	SIGDH	PFS	proof	0.6
8.	SIGDH	eCK	attack	9.4
9.	KEA+	KI+KCI	proof	0.7
10.	KEA+	KI+KCI+wPFS	attack	1.0
11.	STS-MAC	KI, reg-PK	UKS-attack	4.1
12.	STS-MAC-fix1	KI, reg-PK (with PoP)	proof	9.2
13.	STS-MAC-fix2	KI, reg-PK	proof	2
14.	TS1-2004	KI	UKS-attack	0.3
15.	TS1-2008	KI	proof	0.3
16.	TS2-2004	KI+wPFS	attack	0.5
17.	TS2-2008	KI+wPFS	proof	1.2
18.	TS3-2004/2008	KI+wPFS	non-termination	-

Table 3.1. Verification Results for two-pass Diffie-Hellman protocols.

Second, we analyzed the one-pass and three-pass versions of the UM protocol with respect to different adversary models. We also analyzed these and other protocols in combined models where multiple protocols are executed jointly sharing static key material. The results are shown in Table 3.2. We analyzed our formalization of the one-pass UM protocol [137] from Figures 3.2 and 3.3 which models a weakened version of the eCK model. Since we obtained the description of the one-pass UM protocol from [42], we then modeled one-pass UM in their adversary model, which is based on the CK-model [37], and TAMARIN automatically discovered the attack depicted in Figure 3.37.

In the attack, the test session is a responder session with actor B and peer A that receives g^1. After a long-term key reveal for A, the adversary can compute the session key $h(g^b, g^{ba}, A, B)$ of the test session. There is also a second responder session with peer A and actor B that receives g^1. In our model, this is not an attack since a long-term key reveal for the test session's peer is only allowed if there is a matching session for the test session. Since two responder sessions are never matching sessions in our model, this is not the case for this execution.

In the model from [42], the adversary is only allowed to reveal the long-term key of the test session's peer if there is a matching session, like in our model. Unlike our model, their model does not include the role in the session identifier and the second responder session *is* a matching session of the test session in their model. They define the session identifier of the test session as (A, B, g^1), the session identifier of the second session as (B, A, g^1), and the two are matching in their model since the actor of the first session is the peer of the second session, the actor of the second session is the peer of the first session, and the messages are equal. We checked the security proof in [42] and discovered that a case corresponding to our attack is missing. The proof assumes that the matching session of a responder is always an initiator session. We then modified our formalization to include the role in the matching session definition and successfully verified this fix with TAMARIN. Following the analysis in [42] and [129] we also modeled the three-pass UM protocol and analyzed the one-pass and the three-pass versions of UM in a joint model where agents use the same long-term keys for both protocols. Our results confirm the attack and the fix from [129]. Finally, we verify three-pass variants of the KEA+ and NAXOS protocols from [129] in a joint eCK$^{\mathrm{PFS}}$ [110] model with partially matching sessions [129] where the adversary can register arbitrary public keys for corrupted agents.

	Protocol	Adversary Model	Result	Time [s]
19.	UM-one-pass	weakened eCK	proof	0.4
20.	UM-one-pass	CK-like	attack*	3.4
21.	UM-one-pass	CK-like^a	proof	2.6
22.	UM-C+UM-one-pass	CK-like^a	attack	27.4
23.	UM-C^a+UM-one-pass^b	CK-like^a	proof	41
24.	NAXOS-C+DHKEA	eCK$^{\mathrm{PFS}c}$,reg-PK	proof	107

a added role to session identifier
b added protocol identifier to key string
c with partially matching sessions

Table 3.2. Verification Results for one-pass and three-pass Diffie-Hellman protocols. We use * to denote attacks that contradict result proved in the respective papers.

Third, we analyzed tripartite group key exchange protocols and identity-based AKE protocols that employ bilinear pairings. The results are listed in Table 3.3. For the signed Joux protocol [97], we confirmed that it provides PFS. If we add ephemeral key reveals, then TAMARIN discovers an attack. For this protocol, it takes 76 seconds to compute the 160 variants of the second protocol step, which is a substantial amount of the total analysis time. For the TAK1 protocol [5], our adversary model distinguishes between two cases. If the adversary was passive with respect to the test session *test*, then he can reveal *either* the long-term keys of *test*'s actor and *test*'s peers *or* the ephemeral keys of *test* and the matching sessions, but not both. If the adversary was active, then revealing long-term keys of *test*'s actor or *test*'s peers is forbidden. Next, we verify the security of the RYY protocol [155] with respect to wPFS and confirm an attack after adding ephemeral key reveals. For the Scott protocol [159], we obtain a similar result. Finally, we model the Chen-Kudla protocol [44], which uses point addition. We do not support this operation and the required equalities like $[c]([a]P+[b]Q)=([c\,a]P+[c\,b]Q)$ in our model and therefore approximate the point addition with the associative and commutative operator $\sharp$. Since the Chen-Kudla protocol still works if point addition is replaced by sorted concatenation, our model includes the required equalities to execute the protocol. We verify the security of the Chen-Kudla protocol in an eCK-like model where, in addition to the usual restrictions, the adversary is not allowed to reveal the ephemeral keys of the test session *and* the matching session, even if the adversary performs no long-term key reveals. If we remove this restriction, then TAMARIN discovers an attack.

	Protocol	Adversary Model	Result	Time [s]
25.	SIGJOUX (tripartite)	PFS	proof	90.7
26.	SIGJOUX (tripartite)	PFS,eph-reveal	attack	99.8
27.	TAK1 (tripartite)	weakened eCK-like	proof	56.8
28.	TAK1 (tripartite)	eCK-like	attack	77.2
29.	RYY (ID-based)	wPFS	proof	8.3
30.	RYY (ID-based)	wPFS,eph-reveal	attack	7.9
31.	Scott (ID-based)	wPFS	proof	19.3
32.	Scott (ID-based)	wPFS+eph-reveal	attack	26.2
33.	Chen-Kudla^d (ID-based)	eCK-like	proof	61
34.	Chen-Kudla^d (ID-based)	eCK	attack	45.3
	d we use sorted concatenation instead of point addition			

Table 3.3. Verification Results for protocols that use bilinear pairing.

3.5.5 Limitations and Extensions

There are two main causes for non-termination. First, non-termination occurs when the protocol contains loops and the search has to explore the loops to analyze the desired property. For example, it is impossible to prove that $\varphi = \forall i\, x.\, \mathsf{St}(x)@i \Longrightarrow x \approx 1$ holds for

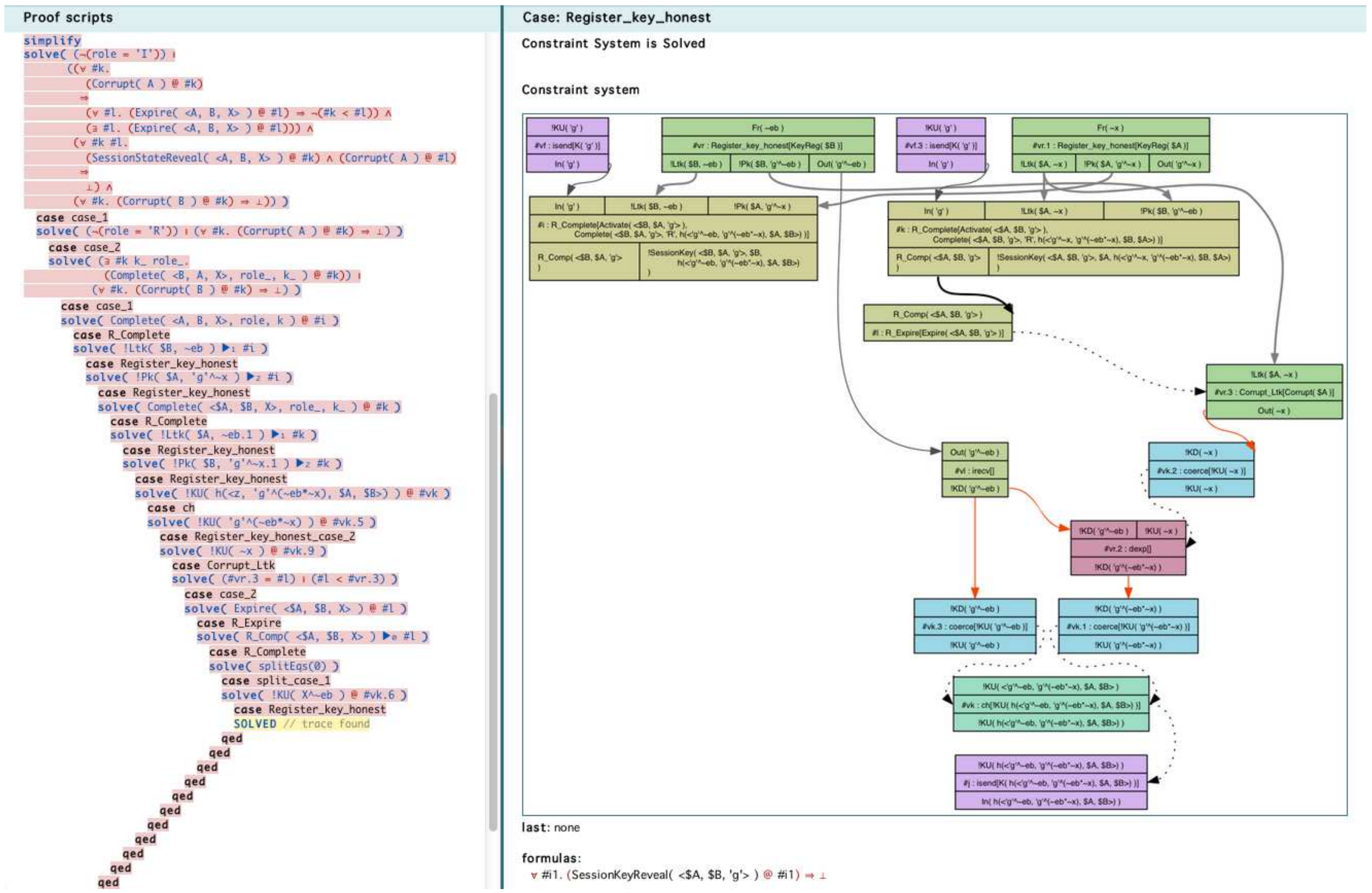
$$P_{Loop} = \left\{ \frac{}{\mathsf{St}(1)}[\,\mathsf{St}(1)\,] \qquad \frac{\mathsf{St}(x)}{\mathsf{St}(x)}[\,\mathsf{St}(x)\,] \right\}$$

using our constraint solving algorithm. To solve this problem, Meier [124] extends the approach with induction. First, he checks if φ is valid for the empty trace, which is always possible. Then, he encodes the induction hypothesis "φ is valid for all prefixes of tr" in a formula φ_{IH}. Since this formula is also a guarded trace property, constraint solving can then be used to check if $\varphi_{IH} \Longrightarrow \varphi$ is valid. This is equivalent to checking if $\varphi_{IH} \wedge \neg\varphi$ is satisfiable, i.e., is there a trace that violates φ such that all prefixes satisfy φ.

Second, we have seen that enforcing normal message deductions and using a search strategy based on the chain property is critical to achieve termination in many cases. Since we apply these techniques only to message deduction rules, protocol rules that provide very general message deduction steps are still problematic. For example, the described constraint solving algorithm does not handle protocols that provide decryption rules such as

$$\frac{\mathsf{In}(enc(x, pk(k))) \quad !\mathsf{Ltk}(A, k)}{\mathsf{Out}(x)}$$

where x is a message variable well. The problem is that if we do not know anything about the possible instantiations for x, then *any* message can be extracted from the sent message. To deal with such protocols, Meier [124] proposes type assertions which are invariants similar to "the variable x in the protocol step i is either instantiated with a nonce or already known to the adversary before i". Given this invariant, it suffices to consider the case where the extracted message is a nonce since the adversary is not allowed to extract a message that is already known in the other cases. To prove such invariants, induction is used.



Proof scripts

```
simplify
solve( (¬(role = 'I')) ∥
       ((∀ #k.
          (Corrupt( A ) @ #k)
         ⇒
          (∀ #l. (Expire( <A, B, X> ) @ #l) ⇒ ¬(#k < #l)) ∧
          (∃ #l. (Expire( <A, B, X> ) @ #l))) ∧
        (∀ #k #l.
          (SessionStateReveal( <A, B, X> ) @ #k) ∧ (Corrupt( A ) @ #l)
         ⇒
          ⊥) ∧
        (∀ #k. (Corrupt( B ) @ #k) ⇒ ⊥)) )
  case case_1
  solve( (¬(role = 'R')) ∥ (∀ #k. (Corrupt( A ) @ #k) ⇒ ⊥) )
    case case_2
    solve( (∃ #k k_ role_.
             (Complete( <B, A, X>, role_, k_ ) @ #k)) ∥
           (∀ #k. (Corrupt( B ) @ #k) ⇒ ⊥) )
      case case_1
      solve( Complete( <A, B, X>, role, k ) @ #i )
        case R_Complete
        solve( !Ltk( $B, ~eb ) ▶₁ #i )
          case Register_key_honest
          solve( !Pk( $A, 'g'^~x ) ▶₂ #i )
            case Register_key_honest
            solve( Complete( <$A, $B, X>, role_, k_ ) @ #k )
              case R_Complete
              solve( !Ltk( $A, ~eb.1 ) ▶₁ #k )
                case Register_key_honest
                solve( !Pk( $B, 'g'^~x.1 ) ▶₂ #k )
                  case Register_key_honest
                  solve( !KU( h(<z, 'g'^(~eb*~x), $A, $B>) ) @ #vk )
                    case ch
                    solve( !KU( 'g'^(~eb*~x) ) @ #vk.5 )
                      case Register_key_honest_case_2
                      solve( !KU( ~x ) @ #vk.9 )
                        case Corrupt_Ltk
                        solve( (#vr.3 = #l) ∥ (#l < #vr.3) )
                          case case_2
                          solve( Expire( <$A, $B, X> ) @ #l )
                            case R_Expire
                            solve( R_Comp( <$A, $B, X> ) ▶₀ #l )
                              case R_Complete
                              solve( splitEqs(0) )
                                case split_case_1
                                solve( !KU( X^~eb ) @ #vk.6 )
                                  case Register_key_honest
                                  SOLVED // trace found
                                qed
                              qed
                            qed
                          qed
                        qed
                      qed
                    qed
                  qed
                qed
              qed
            qed
```

Case: Register_key_honest

Constraint System is Solved

Constraint system

!KU('g')
#vf : isend[K('g')]
In('g')
Fr(~eb)
#vr : Register_key_honest[KeyReg($B)]
!Ltk($B, ~eb)
!Pk($B, 'g'^~eb)
Out('g'^~eb)
!KU('g')
#vf.3 : isend[K('g')]
In('g')
Fr(~x)
#vr.1 : Register_key_honest[KeyReg($A)]
!Ltk($A, ~x)
!Pk($A, 'g'^~x)
Out('g'^~x)
In('g')
!Ltk($B, ~eb)
!Pk($A, 'g'^~x)
#i : R_Complete[Activate(<$B, $A, 'g'>), Complete(<$B, $A, 'g'>, 'R', h(<'g'^~eb, 'g'^(~eb*~x), $A, $B>))]
R_Comp(<$B, $A, 'g'>)
!SessionKey(<$B, $A, 'g'>, $B, h(<'g'^~eb, 'g'^(~eb*~x), $A, $B>))
In('g')
!Ltk($A, ~x)
!Pk($B, 'g'^~eb)
#k : R_Complete[Activate(<$A, $B, 'g'>), Complete(<$A, $B, 'g'>, 'R', h(<'g'^~x, 'g'^(~eb*~x), $B, $A>))]
R_Comp(<$A, $B, 'g'>)
!SessionKey(<$A, $B, 'g'>, $A, h(<'g'^~x, 'g'^(~eb*~x), $B, $A>))
R_Comp(<$A, $B, 'g'>)
#l : R_Expire[Expire(<$A, $B, 'g'>)]
!Ltk($A, ~x)
#vr.3 : Corrupt_Ltk[Corrupt($A)]
Out(~x)
Out('g'^~eb)
#vl : irecv[]
!KD('g'^~eb)
!KD(~x)
#vk.2 : coerce[!KU(~x)]
!KU(~x)
!KD('g'^~eb)
!KU(~x)
#vr.2 : dexp[]
!KD('g'^(~eb*~x))
!KD('g'^~eb)
#vk.3 : coerce[!KU('g'^~eb)]
!KU('g'^~eb)
!KD('g'^(~eb*~x))
#vk.1 : coerce[!KU('g'^(~eb*~x))]
!KU('g'^(~eb*~x))
!KU(<'g'^~eb, 'g'^(~eb*~x), $A, $B>)
#vk : ch[!KU(h(<'g'^~eb, 'g'^(~eb*~x), $A, $B>))]
!KU(h(<'g'^~eb, 'g'^(~eb*~x), $A, $B>))
!KU(h(<'g'^~eb, 'g'^(~eb*~x), $A, $B>))
#j : isend[K(h(<'g'^~eb, 'g'^(~eb*~x), $A, $B>))]
In(h(<'g'^~eb, 'g'^(~eb*~x), $A, $B>))

last: none

formulas:
∀ #i1. (SessionKeyReveal(<$A, $B, 'g'>) @ #i1) ⇒ ⊥

Figure 3.37. TAMARIN visualizes the attack on the UM-one-pass protocol in the CK-like model from [42] after automatically finding it. The test session is displayed on the left and its matching session, which is also a responder session, is shown on the right. On the far right below the matching session, a long-term key reveal for its actor is displayed. In the bottom half, the message deductions for computing the session key are displayed. Note that A's long-term key is $\sim x$ and B's long-term key is $\sim eb$ where $\sim$ is used to denote the sort annotation $:fr$.

Meier [124] and Künnemann and Steel [107] have successfully applied the two extensions described above to verify protocols with loops and unbounded mutable state such as TESLA [145], the envelope protocol [63], and the Yubikey security token [170].

Chapter 4

Impossibility Results for Key Establishment

In Chapter 2, we presented the Diffie-Hellman protocol, which was the first protocol to “enable private communication without *ever* relying on private channels“ (Katz and Lindell [106]). The main insight in the development of the protocol was the usage of DH groups where exponentiation is efficient, repeated exponentiation commutes, and taking the discrete logarithm is hard. In this chapter, we investigate if it is possible to achieve the above goal also with operations that are simpler than exponentiation in a DH group. More precisely, we model cryptographic operators and their algebraic properties abstractly by equational theories, as we did in Chapter 3. Then, we consider the question if a given equational theory, modeling for example XOR, symmetric encryption, or hashing, allows the establishment of a secret using only public (but authentic) channels.

In the following, we first state some preliminaries and define our formal model of secret establishment. Then, we consider symmetric encryption and subterm-convergent theories. Next, we consider monoidal theories such as XOR and abelian groups. Finally, we consider the disjoint combination of equational theories and summarize our results.

4.1 Preliminaries

In this chapter, we use unsorted term algebras and a single set of names $\mathcal{N}$. We also assume that all considered equational theories are consistent, i.e., for two different names n and n', it never holds that $n =_{\mathcal{E}} n'$. For a sequence $a = [a_1, ..., a_k]$, we use $\sigma_{[a]}$ to denote the substitution $\bigcup_{i=1}^{k} \{a_i/x_i\}$. Note that all equational theories are stable under replacement of names by terms, i.e., if ρ is a replacement of names by terms and $s =_{\mathcal{E}} t$, then $s^\rho =_{\mathcal{E}} t^\rho$.

4.2 Traces and Deducibility

In this section, we define formally what it means to establish a shared secret. We first define messages, frames, and deducibility. Then, we define the set of derivation traces that denotes the possible executions where agents construct messages and exchange messages over a public channel. Afterwards, we define the notion of shared secret for such a trace and relate derivation traces and protocols.

4.2.1 Messages, Frames, and Deducibility

As is common practice in symbolic protocol analysis, we abstract away from concrete implementations where messages are encoded and manipulated as bitstrings. Given an equational theory $\mathcal{E} = (\Sigma, E)$, we define the set of messages as $\mathcal{M}_\Sigma = \mathcal{T}_\Sigma(\mathcal{N})$. We use the set of names $\mathcal{N}$ to model atomic messages such as nonces and agent names. The function symbols in Σ model cryptographic operations on abstract messages, where the operations' semantics is given by the set of equations E.

We next define a notion of deducibility on a set of messages. Consider, for example, the case where an adversary has overheard communication between honest agents and has seen the messages m_1, ..., m_k. Given these messages, we are interested in the set of messages that the adversary can deduce by applying cryptographic operations. Here, we have to account for nonces, keys, and other randomly chosen messages, which the adversary cannot construct. We therefore define the notion of a *frame*, as it has been introduced in the applied pi calculus [3]. A *frame* is a pair $(\tilde{n}, \sigma)$ written as $\nu\tilde{n}.\sigma$ such that:

- $\tilde{n}$ is a finite set of *restricted* names. Intuitively, this is a set of fresh names and models unguessable messages created by honest agents. Although it might be possible for the adversary to deduce restricted names, the adversary cannot create them directly.
- σ is a substitution $\{m_1/x_1, ..., m_k/x_k\}$. This allows the adversary to use the observed messages $m_1, ..., m_k$ when constructing new ones.

$$\textsc{Const}\ \frac{n \in \mathcal{N} \setminus \tilde{n}}{\nu\tilde{n}.\sigma \vdash_{\mathcal{E}} n} \qquad\qquad \textsc{Know}\ \frac{x \in dom(\sigma)}{\nu\tilde{n}.\sigma \vdash_{\mathcal{E}} x\sigma}$$

$$\textsc{Apply}\ \frac{\nu\tilde{n}.\sigma \vdash_{\mathcal{E}} m_1 \quad ... \quad \nu\tilde{n}.\sigma \vdash_{\mathcal{E}} m_k \quad f \in \Sigma^k}{\nu\tilde{n}.\sigma \vdash_{\mathcal{E}} f(m_1, ..., m_k)} \qquad\qquad \textsc{Equal}\ \frac{\nu\tilde{n}.\sigma \vdash_{\mathcal{E}} m \quad m =_{\mathcal{E}} m'}{\nu\tilde{n}.\sigma \vdash_{\mathcal{E}} m'}$$

Figure 4.1. The deducibility relation $\vdash_{\mathcal{E}}$ is inductively defined by these rules.

Based on the notion of a frame, we define the deducibility relation $\vdash_{\mathcal{E}}$ between frames and messages for an equational theory $\mathcal{E}$. The corresponding rules are presented in Figure 4.1 and model that the adversary can take any of the following actions.

- Const: The adversary can deduce any name, except the restricted ones in the set $\tilde{n}$.
- Know: The adversary can deduce all message in the range of σ.
- Apply: The adversary can apply function symbols in Σ to deducible messages.
- Equal: The adversary can deduce messages that are equivalent to deducible messages modulo the equational theory $\mathcal{E}$.

The relation $\vdash_{\mathcal{E}}$ can be equivalently defined as follows.

Lemma 4.1. *For all finite sets of names $\tilde{n}$, substitutions σ, and terms m, $\nu\tilde{n}.\sigma \vdash_{\mathcal{E}} m$ if and only if there is a term $\zeta \in \mathcal{T}_\Sigma(dom(\sigma) \cup (\mathcal{N} \setminus \tilde{n}))$ such that $\zeta\sigma =_{\mathcal{E}} m$. We call such a term ζ a recipe for m.*

For theories that contain a nullary function symbol, we can always find a recipe that does not contain unrelated names.

Lemma 4.2. *If $\mathcal{E}$ is a theory with a nullary function symbol c, $names(range(\sigma) \cup \{m\}) \subseteq \tilde{n}$, then $\nu\tilde{n}.\sigma \vdash_{\mathcal{E}} m$ if and only if there is recipe $\zeta \in \mathcal{T}_\Sigma(dom(\sigma))$.*

Proof. We can just take an arbitrary recipe and replace all names $n \in \mathit{names}(\zeta)$with c. The result is still a recipe since all names in m and $\mathit{range}(\sigma)$ are restricted and cannot occur in ζ. Since $\mathcal{E}$ is stable under replacement of names by terms, we have $\zeta^\rho\sigma =_\mathcal{E} (\zeta\sigma)^\rho =_\mathcal{E} m^\rho = m$ for a replacement ρ that maps all $n \in \mathit{names}(\zeta)$ to c. □

Note that the notion of frames and the deducibility relation $\vdash_\mathcal{E}$ can also be used for terms that are not ground, which we exploit in Section 4.3.2.

4.2.2 Derivation Traces

In the following, let $\mathcal{A}$ be the set of agents and let $\mathcal{E} = (\Sigma, E)$ describe the cryptographic operators under consideration and their relevant properties. An event either denotes that an agent sends a message or learns a message. Events are therefore associated with the corresponding agent's identity. A learn event is additionally tagged with the rule R that describes *how* the agent learned the message.

$$\mathit{Event} ::= \mathit{Send}(\mathcal{A}, \mathcal{M}_\Sigma) | \mathit{Learn}_R(\mathcal{A}, \mathcal{M}_\Sigma)$$

The steps taken to construct and communicate messages are modeled by traces, where a trace is a list of events. The set of valid traces $\mathit{TR}_\mathcal{E}$ is inductively defined by the rules in Figure 4.2. We use $\mathit{restricted}(\mathit{tr}) = \{n | \exists A.\, \mathit{Learn}_\mathit{Fresh}(A, n)\}$ to denote the restricted names in the trace and Learn_R to denote *Learn*-events with arbitrary rule-tags. The rules model the following actions.

- Send: An agent A sends a previously learned message m on the public channel.
- Recv: An agent A receives a message m that has been previously sent by B.
- Fresh: An agent A creates a fresh name n. Note that the minimal name n_min under the ordering $\succ$ used for ordered completion cannot be used here or in the Public rule.
- Public: An agent A uses a public value p.
- App: An agent A applies a k-ary function symbol f to the previously learned messages $m_1, \ldots, m_k$. Here we use the fact that every message has a unique normal form modulo $\mathcal{E}$ with respect to $\to_{(\succ, O_\mathcal{E})}$.

4.2.3 Shared Secrets and Deducibility

Clearly, we must restrict the initial knowledge of agents to prove impossibility results for secret establishment. Some restrictions are necessary to prevent initial knowledge distributions that allow the creation of a shared secret, but require the previous existence of secret channels, e.g., shared secret keys distributed by a third party. We enforce this restriction by requiring that every derivation starts with the empty trace, which corresponds to the empty initial knowledge for the involved agents. However, in our model, we do not distinguish between the setup phase and the execution phase. Therefore, any prefix of a derivation trace can be interpreted as a setup phase where agents establish private and public knowledge in the presence of the attacker. This covers precisely the initial knowledge distributions that do not require secret channels and include all messages involved in establishing the knowledge.

$$\textsc{Empty}\ \frac{}{[\,] \in TR_{\mathcal{E}}} \qquad \textsc{Send}\ \frac{tr \in TR_{\mathcal{E}} \quad Learn_R(A,m) \in tr}{tr \cdot Send(A,m) \in TR_{\mathcal{E}}}$$

$$\textsc{Recv}\ \frac{tr \in TR_{\mathcal{E}} \quad Send(B,m) \in tr}{tr \cdot Learn_{Recv}(A,m) \in TR_{\mathcal{E}}}$$

$$\textsc{Fresh}\ \frac{tr \in TR_{\mathcal{E}} \quad n \in \mathcal{N} \setminus (names(tr) \cup \{n_{min}\})}{tr \cdot Learn_{Fresh}(A,n) \in TR_{\mathcal{E}}}$$

$$\textsc{Public}\ \frac{tr \in TR_{\mathcal{E}} \quad p \in \mathcal{N} \setminus (restricted(tr) \cup \{n_{min}\})}{tr \cdot Learn_{Public}(A,p) \in TR_{\mathcal{E}}}$$

$$\textsc{App}\ \frac{\begin{array}{c} tr \in TR_{\mathcal{E}} \quad f \in \Sigma^k \\ Learn_{R_1}(A,m_1) \in tr \quad \ldots \quad Learn_{R_k}(A,m_k) \in tr \end{array}}{tr \cdot Learn_{App(f(m_1,\ldots,m_k))}(A, f(m_1,\ldots,m_k)\downarrow_{\mathcal{E}}^{\succ}) \in TR_{\mathcal{E}}}$$

Figure 4.2. The set $TR_{\mathcal{E}}$ is inductively defined by these rules.

We are interested in impossibility results. An impossibility result for a class of adversaries implies impossibility for any larger class. As a consequence, we reason about weak adversaries, namely passive adversaries who are restricted to eavesdropping communication on the public channel. This models an adversary who is not involved in the derivation process. For a given trace tr, we therefore define the corresponding frame ϕ_{tr} as follows. Let $sent(tr)$ denote the list of messages $[m_1, \ldots, m_k]$ that have been sent in tr, then ϕ_{tr} is defined as $\phi_{tr} = \nu\, restricted(tr).\sigma_{[sent(tr)]}$.

Note that we combine the deduction rules for honest agents with the rules for exchanging messages in the definition of $TR_{\mathcal{E}}$. However, the deduction rules formalize deduction capabilities that are equivalent to the relation $\vdash_{\mathcal{E}}$. The only difference is that the messages in derivation traces are always in normal form and we therefore do not need an Equal rule.

We now define what it means to share a secret.

Definition 4.3. *A term s is a shared secret between A and B in a trace tr if $A \neq B$, $Learn_{R_1}(A,s) \in tr$, $Learn_{R_2}(B,s) \in tr$, and $\phi_{tr} \nvdash_{\mathcal{E}} s$.*

To simplify subsequent proofs, we first show that we can restrict ourselves to a pair of agents.

Lemma 4.4. *If there is a trace that establishes a secret between the distinct agents A and B involving an arbitrary number of agents, then there is also a trace where only A and B participate.*

Proof. We translate the trace with an arbitrary number of agents to a trace with only A and B. The translation maps all events executed by an agent $C \notin \{A,\ B\}$ to the corresponding event executed by A. It is easy to check that this translation yields a valid trace. $\square$

In the following, we therefore fix $\mathcal{A} = \{\mathbb{A}, \mathbb{B}\}$, where $\mathbb{A}$ and $\mathbb{B}$ are distinct agents. To define the notion of minimal traces, we require the following definition.

Definition 4.5. *We say that an agent A* constructs *a message m in* tr*, if A first learns m in a $Learn_R$-event for $R \neq Recv$. We say A* freely constructs *m in tr if no reduction of the message occurs in the event where A first learns m, i.e., $R = Fresh$, $R = Public$, or $R = App(m)$.*

We now introduce the notion of minimal trace.

Definition 4.6. *A trace $tr \in TR_{\mathcal{E}}$ is a* minimal trace *if each of the following conditions hold.*

1. *There is at most one shared secret s. This shared secret is learned by $\mathbb{B}$ in the last event, which is a $Learn_{App}$-event.*
2. *There are no $Learn_{Public}$-events in the trace, i.e., all names are restricted.*
3. *There are no duplicate events and every message except for s is constructed only once.*
4. *The secret s is freely constructed at most once.*

We denote the subset of minimal traces by $TR_{\mathcal{E}}^{min}$.

We can now prove that the conditions on minimal traces do not prohibit secret establishment if it is otherwise possible.

Lemma 4.7. *If there is a trace $tr \in TR_{\mathcal{E}}$ that establishes a shared secret, then there is also a minimal trace $\hat{tr} \in TR_{\mathcal{E}}^{min}$ that establishes a shared secret.*

Proof. We first prove by induction that a trace $tr \in TR_{\mathcal{E}}$ without a shared secret can be transformed into a minimal trace without changing $names(tr)$, the deducible messages for the adversary, and the messages known by $\mathbb{A}$ and $\mathbb{B}$. The empty trace $[]$ is already minimal. Assume that the trace tr can be transformed into the minimal trace $\hat{tr}$. If tr is extended with a *Send*-event by the SEND rule, then we extend $\hat{tr}$ with the same *Send*-event, provided it is not a duplicate event. If tr is extended with an event $Learn_{Recv}(A, m)$ by the RECV rule, then we extend $\hat{tr}$ with the same event, provided m is not already known to A. If tr is extended by the FRESH rule, then we extend $\hat{tr}$ with the same event, which is allowed since $names(tr) = names(\hat{tr})$. If tr is extended with an event $Learn_{Public}(A, p)$ by PUBLIC, then p is deducible by the adversary and we distinguish three cases. If p is already known to A, we do not extend $\hat{tr}$. If p is not known to A, but to some other agent B, we extend $\hat{tr}$ with the events $Send(B,\ p)$ (if required) and $Learn_{Recv}(A,\ p)$. If p is not known to any agent, then $p \notin names(\hat{tr})$ and we extend $\hat{tr}$ with $Learn_{Fresh}(A, p)$ and $Send(A, p)$. Finally, if tr is extended with an event $ev = Learn_{App(f(m_1,\dots,m_k))}(A, m)$ by APP, we again distinguish three cases. If m is already known to A, then we do not extend $\hat{tr}$. If m is only known to another agent B, we extend $\hat{tr}$ with the required *Send* and $Learn_{Recv}$ events. This is possible because we only consider traces $tr \cdot ev$ which contain no shared secrets. Since m is shared knowledge, it cannot be secret. Finally, if m is not known to any agent in $\hat{tr}$, then we extend $\hat{tr}$ with ev.

To transform a trace $tr \in TR_{\mathcal{E}}$ with shared secrets into a minimal trace that establishes exactly one shared secret, we first take the shortest prefix $trp \cdot ev$ of tr that establishes a shared secret. Since an agent cannot establish a secret with a *Send* or $Learn_R$ event for $R \in \{Recv, Public, Fresh\}$, the last event ev must be a $Learn_{App}$-event, where an agent C learns some secret s. Without loss of generality, we assume $C = \mathbb{B}$. We can transform trp into a minimal trace $\widehat{trp}$ using the previously described transformation since it does not contain a shared secret. Then $\widehat{trp} \cdot ev \in TR_{\mathcal{E}}^{min}$ since ev can be added to $\widehat{trp}$ without violating properties 1.–4. Properties 1.–3. are obvious. To see that 4. holds, assume that s is freely constructed by both $\mathbb{A}$ and $\mathbb{B}$. Then $s = f(m_1, \ldots, m_k)$ for some $f \in \Sigma^k$ and messages m_i and there are $Learn_{App(s)}(C, s)$ events for $C = \mathbb{A}$ and $C = \mathbb{B}$. Then all m_i are shared knowledge and since s is the only secret, all m_i are deducible by the adversary. Hence, $s = f(m_1, \ldots, m_k)$ is also deducible by applying f. □

4.2.4 Relating Protocols and Derivation Traces

Above we have reduced the question whether it is possible to establish a shared secret using an equational theory $\mathcal{E}$ to the question of whether there is a derivation trace $tr \in TR_{\mathcal{E}}$ that establishes a shared secret. This is closely related to the question of whether there is a *protocol* that establishes a shared secret in a given symbolic protocol model.

The existence of a protocol implies that there is a successful protocol execution where the involved agents establish a shared secret. For all reasonable protocol models that use the same notion of deducibility as we do, a successful protocol execution directly yields a corresponding derivation trace that establishes a shared secret. An impossibility result for some equational theory $\mathcal{E}$ in our model thus directly implies the corresponding impossibility result for protocols in such a symbolic model. A concrete example of a protocol model where this relationship holds is the applied pi calculus with equational theory $\mathcal{E}$ where agents are only allowed to communicate over public channels.

4.3 Symmetric Encryption and Subterm-Convergent Theories

In this section, we prove the folk theorem that it is impossible to establish a shared secret using only symmetric encryption and public channels. We then present a necessary and sufficient condition for impossibility for the more general case of subterm-convergent theories. This condition can be used to automatically decide wether it is possible to create a shared secret for a given subterm-convergent theory. We have implemented a decision procedure that checks this condition and illustrate its application to the theory of symmetric encryption. Afterwards, we show how our procedure finds a derivation trace that establishes a shared secret for the theory of public-key encryption.

4.3.1 Symmetric Encryption

We use the equational theory $Sym = (\Sigma_{Sym}, E_{Sym})$ to model symmetric encryption, pairing, a hash function, decryption, and projections on pairs.

$$\Sigma_{Sym} = \{enc(\_,\_), \langle\_,\_\rangle, h(\_), dec(\_,\_), \pi_1(\_), \pi_2(\_)\}$$
$$E_{Sym} = \{dec(enc(m,k),k) \simeq m, \pi_1(\langle m_1, m_2\rangle) \simeq m_1, \pi_2(\langle m_1, m_2\rangle) \simeq m_2\}$$

Since the rewriting system R_{Sym} obtained from E_{Sym} by orienting the equations from left to right is subterm-convergent, we directly use $\rightarrow_{R_{Sym}}$ to normalize terms in the App rule and do not require ordered completion. In the following, we say an equational theory is subterm-convergent if the equations can be oriented such that the corresponding rewrite system is subterm-convergent and we denote the normal form with respect to this rewrite system with $t{\downarrow_{\mathcal{E}}}$. The following lemma holds for all subterm-convergent theories and will be used in Section 4.3.2 as well.

Lemma 4.8. *Let $\mathcal{E} = (\Sigma, E)$ be a subterm-convergent theory and $tr \in TR_{\mathcal{E}}$. For every event $Learn_R(A, m) \in tr$ and $m' \in St(m)$ such that $names(m') \neq \emptyset$, m' has been freely constructed by some agent C. More precisely, if m' is a name, then $Learn_{Fresh}(C, m') \in tr$ or $Learn_{Public}(C, m') \in tr$. Otherwise, there exist $m_1, ..., m_k$ and an $f \in \Sigma^k$, such that $m' = f(m_1, ..., m_k)$ and $Learn_{App(m')}(C, m') \in tr$.*

Proof. The proof is straightforward using rule induction on $TR_{\mathcal{E}}$. The rules Empty and Send are trivial since they do not add any *Learn*-events. The Fresh and Public rules are also trivial since they only add (atomic) names. The Recv rule adds an event $Learn_{Recv}(A, m)$, but there must be a corresponding $Learn_R(B,\ m)$ by the sender B. We can therefore apply the induction hypothesis. The App-rule adds an event $Learn_{App(f(m_1,...,m_k))}(A, m)$, where $m = f(m_1, ..., m_k){\downarrow_{\mathcal{E}}}$. We must show that the lemma's statement holds for all subterms of m that contain names. Since the equational theory is subterm-convergent, we have either $m = g$ for some $g \in \mathcal{T}_\Sigma(\emptyset)$, $m = f(m_1, ..., m_k)$, or m is a proper subterm of $f(m_1, ..., m_k)$. In the first case, the statement follows trivially, since no term in $St(g)$ contains names. In the second case, a subterm of m is either m itself and the statement trivially holds or a subterm of some m_i and the statement holds by the induction hypothesis since there is an event $Learn_R(A, m_i)$ in the trace. The same reasoning applies to the final case since m and all its subterms are subterms of some m_i. □

Theorem 4.9. *There is no derivation trace for Sym that establishes a shared secret. Namely, if $tr \in TR_{Sym}$, $Learn_{R_1}(\mathbb{A}, s) \in tr$, and $Learn_{R_2}(\mathbb{B}, s) \in tr$, then $\phi_{tr} \vdash_{Sym} s$.*

Proof. We prove the theorem by contradiction. Assume that there is a trace in TR_{Sym} that establishes a shared secret. Then, there is also a trace $tr \in TR^{min}_{Sym}$ that establishes a shared secret s and the last event of tr is of the form $ev = Learn_{App(f(m_1,...,m_l))}(\mathbb{B}, s)$ for $s = f(m_1, ..., m_l){\downarrow_{Sym}}$. Thus $Learn_R(\mathbb{A}, s) \in tr$ and we show that $\phi_{tr} \vdash_{Sym} s$ to obtain a contradiction. We distinguish two cases.

1. If $s = f(m_1, ..., m_l)$ then Lemma 4.8 can be applied to $Learn_R(\mathbb{A}, s)$ and the prefix of the trace up to this event. Thus there are two free construction events for s which contradicts minimality of tr.
2. If $f(m_1, ..., m_l)$ is not in normal form, then we must consider two more cases.
   a) Assume that $f(m_1, ..., m_l) = \pi_i(\langle t_1, t_2\rangle)$ for some messages t_i. Then $s = t_i$ and $\langle t_1, t_2\rangle$ has been freely constructed by $\mathbb{A}$ by Lemma 4.8. Since the pair is known to both $\mathbb{A}$ and $\mathbb{B}$ and s is the only secret in tr, the pair is deducible by the adversary and therefore s is also deducible.
   b) Assume that $f(m_1, ..., m_l) = enc(m, k)$ for some messages m and k. Then $s = m$ and $enc(m, k)$ has been freely constructed by $\mathbb{A}$ by Lemma 4.8. Then k and $enc(m, k)$ are known to both $\mathbb{A}$ and $\mathbb{B}$ and are therefore deducible by the adversary. Hence, s is also deducible. □

Note that the only parts in the proof that are specific to *Sym* and do not hold for all subterm-convergent theories are the cases 2.a) and 2.b) for the different rules in R_{Sym}. Here, a rule $l \rightarrow r$ is applied to the result of the function application in the App-rule and we prove impossibility by showing that there is no way for $\mathbb{A}$ and $\mathbb{B}$ to build an instance of l such that the corresponding instance of r is not deducible by the adversary. We will show in the next section that this proof method works with arbitrary subterm-convergent theories.

4.3.2 Subterm-Convergent Theories

We now generalize the previous proof method to a necessary and sufficient condition for the possibility of secret establishment for a subterm-convergent theory $\mathcal{E} = (\Sigma, E)$. Since the condition can be automatically checked, we thereby obtain a decision procedure. The main idea is that if there is a derivation trace where a secret is established, then there is also a minimal trace with exactly one reduction step that establishes a secret, i.e., all constructions except for one are free. We can decide if such a minimal trace exists by considering all equations in E individually and enumerating all possible ways to jointly construct a reducible term. Finally, we check if this construction leads to a shared secret.

In the rest of this section, we assume that $\mathcal{E}$ is a subterm-convergent equational theory. Note that we can assume without loss of generality that the right-hand sides of the rules in the corresponding rewrite system are normalized. To prove our main result of this section, we require the following lemma about deducibility in subterm-convergent theories.

Lemma 4.10. *For all substitutions σ with $range(\sigma) \subseteq \mathcal{T}_\Sigma(\mathcal{V})$, $s \in \mathcal{T}_\Sigma(\mathcal{V})$, and τ an arbitrary substitution, if $\nu\emptyset.\sigma \vdash_{\mathcal{E}} s$, then $\nu\,\tilde{n}.\,(\sigma \circ \tau) \vdash_{\mathcal{E}} s\tau$ for $\tilde{n} = names(range(\tau))$.*

Proof. Let σ, s, τ, and $\tilde{n}$ as defined above. If $\nu\emptyset.\sigma \vdash_{\mathcal{E}} s$ then there is a recipe ζ such that $names(\zeta) \cap \tilde{n} = \emptyset$ and $\zeta\sigma =_{\mathcal{E}} s$. Thus $\zeta(\sigma \circ \tau) = (\zeta\sigma)\tau =_{\mathcal{E}} s\tau$ and hence $\nu\,\tilde{n}.\,(\sigma \circ \tau) \vdash_{\mathcal{E}} s\tau$. $\square$

We also require the notion of reduction events.

Definition 4.11. *A reduction event is an event of the form $Learn_{App(f(m_1,\ldots,m_k))}(C, m)$ where $f(m_1, \ldots, m_k)$ is not $\downarrow_{\mathcal{E}}$-normal, i.e., $f(m_1, \ldots, m_k) \neq f(m_1, \ldots, m_k)\downarrow_{\mathcal{E}} = m$.*

We now show that we can restrict ourselves to a subset of the minimal traces in the case of subterm-convergent theories.

Lemma 4.12. *If there is a trace $tr \in TR_{\mathcal{E}}$ that establishes a shared secret between $\mathbb{A}$ and $\mathbb{B}$, then there is also a minimal trace $\hat{tr} \in TR_{\mathcal{E}}^{min}$ that establishes a shared secret s between $\mathbb{A}$ and $\mathbb{B}$ such that s is freely constructed by $\mathbb{A}$ and the last event is the only reduction event in $\hat{tr}$.*

Proof. Assume that there is a trace that establishes a shared secret. Then there is a minimal trace $tr \in TR_{\mathcal{E}}^{min}$ that establishes the shared secret s, where the last event is of the form $Learn_{App(f(m_1,\ldots,m_k))}(\mathbb{B}, s)$. This event must be a reduction event. Otherwise, $s = f(m_1, \ldots, m_k)$ and Lemma 4.8 can be applied to $Learn_R(\mathbb{A}, s)$. Then there must be two events $Learn_{App(s)}(C, s)$, for $C = \mathbb{A}$ and $C = \mathbb{B}$, which contradicts the minimality of tr. Note also that $\mathbb{A}$ must have freely constructed s by Lemma 4.8, since it is new to $\mathbb{B}$.

We now show that all other reduction events in tr can be replaced by non-reduction-events. A reduction event has the form $\mathit{Learn}_{\mathit{App}(f(m_1,\ldots,m_k))}(C,\, m)$ such that $m \in \mathcal{T}_\Sigma(\emptyset)$ and $\downarrow_{\mathcal{E}}$-normal or m is a subterm of some m_i. In the first case, the reduction-events can be replaced by $\mathit{Learn}_{\mathit{App}}$-events to construct m where subterms that are already known by the other agent are transmitted. Since all these subterms do not contain any names, they are deducible by the adversary. In the second case, the other agent must have freely constructed m by Lemma 4.8. Hence, m is constructed twice which contradicts minimality of tr since m is not a shared secret. □

Lemma 4.12 implies that we can consider the equations in E separately, since we only have to consider traces with a single reduction step. To enumerate all different ways of building a reducible term that establishes a shared secret, we introduce labelings of terms. A labeling of a term t is a function from $St(t)$ to $\{\mathbb{A}, \mathbb{B}\}$ and captures which agent has constructed which subterm. For such a labeling l_t of a term t, the minsent function returns the minimal set of exchanged terms that corresponds to l_t. It captures that if $\mathbb{A}$ uses a term created by $\mathbb{B}$ or vice versa, it must have been sent.

Definition 4.13. *We define the* minimal set of sent terms of as term t with labeling l_t *as*

$$\mathit{minsent}(t, l_t) = \begin{cases} (\cup_{1\leq i\leq k}\, \mathit{minsent}(t_i, l_t)) \cup \{t_i \mid 1 \leq i \leq k \wedge l_t(t) \neq l_t(t_i)\} & \text{if } t = f(t_1, \ldots, t_k) \\ \emptyset & \text{otherwise} \end{cases}.$$

Using the notion of labeling and the function *minsent*, we obtain the following condition to decide impossibility.

Theorem 4.14. *There is a derivation trace $tr \in TR_{\mathcal{E}}$ that establishes a shared secret if and only if there is an equation $t \simeq s$ in E where s is a proper subterm of t and a labeling l_t of t such that*

1. $l_t(t) = \mathbb{B}$,

2. $l_t(c) = \mathbb{A}$ for all $c \in St(t) \cap \Sigma^0$,

3. $l_t(s) = \mathbb{A}$, and

4. $\nu\, \emptyset.\, \sigma_{[\mathit{minsent}(t,l_t)]} \not\vdash_{\mathcal{E}} s$.

Proof. ⇒: Assume there is a trace that establishes a shared secret. Then there is also a minimal trace $tr \in TR_{\mathcal{E}}^{min}$ where $\mathbb{B}$ learns the secret s in the last event, which is the only reduction event by Lemma 4.12. Consider the term $t = f(m_1, \ldots, m_k)$ where s is extracted using the rule $t' \to s'$ in the last step. Then s' is a proper subterm of t' and there is a substitution τ such that $t'\tau = t$ and $s'\tau = s$.

We can extract a labeling l_t from the trace by labeling t with $\mathbb{B}$, all constants in t with $\mathbb{A}$ and all proper subterms of t that are not constants with the agent who freely constructed the term. Note that every subterm of t that is not a constant must have been freely constructed by exactly one of the agents because of Lemma 4.8 and minimality of tr. The labeling l_t can be translated to a labeling $l_{t'}$ of t' by defining $l_{t'}(u) := l_t(u\tau)$ for $u \in St(t')$. Then $l_{t'}$ obviously has properties 1.–2. It has property 3. because $s'\tau = s$ and $\mathbb{A}$ freely constructs s in tr. We know that $\mathrm{minsent}(t', l_{t'})\tau \subseteq \mathrm{minsent}(t, l_t) \subseteq \mathit{sent}(tr)$ and $\phi_{tr} \not\vdash_{\mathcal{E}} s$. Thus we have $\nu\, \mathit{restricted}(\mathrm{tr}).\, (\sigma_{[\mathit{minsent}(t',l_{t'})]} \cdot \tau) \not\vdash_{\mathcal{E}} s'\tau$. By Lemma 4.12, we obtain $\nu\, \emptyset.\, \sigma_{[\mathit{minsent}(t',l_{t'})]} \not\vdash_{\mathcal{E}} s'$.

$\Leftarrow$: Given a term $t \in \mathcal{T}_\Sigma(\mathcal{V})$, we show that for all substitutions τ that instantiate the variables in t with distinct names, and for all labelings l_t of t, we can obtain a trace $tr \in TR_\mathcal{E}$ such that: (1) all names are restricted, (2) $sent(tr) = (minsent(t, l_t)\tau){\downarrow}$, and (3) for all $u \in St(t)$, the agent $l_t(u)$ learns $u\tau$ in tr. We prove this by induction over the term t. First, if $t = x$ for a variable x and τ and l_t as stated above, then $tr = [Learn_{Fresh}(l_t(x), x\tau)]$ is in $TR_\mathcal{E}$ and satisfies (1)–(3). Next, if $t = f(t_1, ..., t_k)$ for $f \in \Sigma^k$ and terms t_i, τ and l_t as stated above, then there are traces tr_i for t_i, τ, and l_t with the expected properties by the induction hypothesis. Let $\hat{tr}$ denote the concatenation of the tr_i, where duplicate events are removed, keeping only the first occurrence of an event. Then $\hat{tr} \in TR_\mathcal{E}$ and we can extend $\hat{tr}$ with events $Send(l_t(t_i), (t_i\tau){\downarrow_\mathcal{E}})$ and $Learn_{Recv}(l_t(t), (t_i\tau){\downarrow_\mathcal{E}})$ for all i where $l_t(t) \neq l_t(t_i)$. Then, we add the event $Learn_{App(m')}(l_t(t), m)$ with $m' = f((t_1\tau){\downarrow_\mathcal{E}}, ..., (t_k\tau){\downarrow_\mathcal{E}})$ and $m = f(t_1\tau, ..., t_k\tau){\downarrow_\mathcal{E}}$ to obtain the trace $tr \in TR_\mathcal{E}$ with the desired properties. If there is an equation $t \simeq s$ where s is a proper subterm of t in E and 1.–4. hold for the given labeling, then this trace establishes the shared secret $s\tau$ between $\mathbb{A}$ and $\mathbb{B}$. $\square$

Based on this theorem, we define a decision procedure FIND-DERIVATION-TRACES that checks the theorem's conditions for a given theory. Our procedure takes a subterm-convergent theory $\mathcal{E} = (\Sigma_\mathcal{E}, E_\mathcal{E})$ as input and either returns IMPOSSIBLE if there is no labeling that allows secret establishment for a rule in $E_\mathcal{E}$ or a list of derivation traces if there is a labeling of a rule in $E_\mathcal{E}$ that satisfies the conditions of Theorem 4.14.

```
FIND-DERIVATION-TRACES($\mathcal{E}$)
1 traces ← [ ]
2 for ($t \simeq s$) in $E_\mathcal{E}$
3   for $l_t$ in LABELINGS($t$)
4     if ¬ (MINSENT($t, l_t$) $\vdash_\mathcal{E} s$) ∧ $s \in St(t)$ then
5       traces ← traces·LABELTOTRACE($t, l_t, s$)
6 if (traces = [ ]) then return IMPOSSIBLE
7 else return traces
```

Figure 4.3. The pseudocode for the decision procedure FIND-DERIVATION-TRACES.

The procedure uses four subroutines. The subroutine LABELINGS returns all labelings for a rule that satisfy conditions (1)–(3). MINSENT returns the minimal set of sent terms for a labeling. The subroutine for $\vdash_E$ implements the procedure described in [2] to check deducibility. Finally, LABELTOTRACE converts a labeling to a trace. Note that by Theorem 4.14 we can consider all rules individually. However, we must check deducibility for the whole equational theory $\mathcal{E}$.

We have implemented our decision procedure in Haskell. Although the number of labelings grows exponentially in the size of the equations, the procedure returns the result immediately for the examples we considered. Typically, the rules are small and most labels are already predetermined by the conditions of Theorem 4.14. We can further optimize LABELINGS by taking into account that properties (3) and (4) imply $l_t(u) = \mathbb{A}$ for all $u \in St(t)$ that have s as immediate subterm. This is because s would otherwise be in $minsent(t, l_t)$ and therefore be trivially deducible. We have applied our implementation of the decision procedure to the theory *Sym* from the previous section. We have thereby obtained an automated confirmation of the pen-and-paper proof of Theorem 4.9. We use *Sym* and a theory that models public-key encryption below to illustrate our decision procedure.

Example 4.15. The procedure considers the three following reduction rules individually.

1. $\pi_1(\langle x, y\rangle) \to x$: The only possible labelings are $\pi_1(\langle x^{\mathbb{A}}, y^{C}\rangle^{\mathbb{A}})^{\mathbb{B}}$ for $C = \mathbb{A}$ and $C = \mathbb{B}$. For both labelings, $\langle x, y\rangle$ is in the minimal set of sent terms and x is therefore deducible.
2. $\pi_2(\langle x, y\rangle) \to y$: Analogous to previous rule.
3. $\mathit{dec}(\mathit{enc}(m, k), k) \to m$: The only possible labelings are $\mathit{dec}(\mathit{enc}(m^{\mathbb{A}}, k^{C})^{\mathbb{A}}, k^{C})^{\mathbb{B}}$ for $C = \mathbb{A}$ and $C = \mathbb{B}$. For both choices of C, k is used by both $\mathbb{A}$ and $\mathbb{B}$ and thus must have been sent. Similarly, $\mathit{enc}(m, k)$ must have been sent in both case. Therefore, m is deducible by the intruder.

Thus $\textsc{MinSent}(t, l_t) \vdash_{\mathit{Sym}} s$ for all labelings and the procedure returns $\textsc{Impossible}$.

Example 4.16. Consider the theory $\mathit{ASym} = (\Sigma_{\mathit{Sym}}, E_{\mathit{Sym}})$ with

$$\Sigma_{\mathit{ASym}} = \{\mathit{aenc}(\_, \_), \mathit{pk}(\_), \mathit{adec}(\_, \_)\} \quad \text{and}$$
$$E_{\mathit{ASym}} = \{\mathit{adec}(\mathit{aenc}(m, \mathit{pk}(k))) \simeq m\}.$$

For this theory, our procedure returns the labeling $\mathit{adec}(\mathit{aenc}(m^{\mathbb{B}}, \mathrm{pk}(k^{\mathbb{A}})^{\mathbb{A}})^{\mathbb{B}}, k^{\mathbb{A}})^{\mathbb{A}}$ and the following derivation trace that establishes the secret s.

$$\begin{aligned}[\ & \mathit{Learn}_{\mathit{Fresh}}(\mathbb{A}, k), \mathit{Learn}_{\mathit{App}(\mathit{pk}(k))}(\mathbb{A}, \mathit{pk}(k)), \mathit{Send}(\mathbb{A}, \mathit{pk}(k)), \mathit{Learn}_{\mathit{Recv}}(\mathbb{B}, \mathit{pk}(k)),\\ & \mathit{Learn}_{\mathit{Fresh}}(\mathbb{B}, s), \mathit{Learn}_{\mathit{App}(\mathit{aenc}(s, \mathit{pk}(k)))}(\mathbb{B}, \mathit{aenc}(s, \mathit{pk}(k))), \mathit{Send}(\mathbb{B}, \mathit{aenc}(s, \mathit{pk}(k))),\\ & \mathit{Learn}_{\mathit{Recv}}(\mathbb{A}, \mathit{aenc}(s, \mathit{pk}(k))), \mathit{Learn}_{\mathit{App}(\mathit{adec}(\mathit{aenc}(s, \mathit{pk}(k)), k))}(\mathbb{A}, s)\ \ ]\end{aligned}$$

Note that Theorem 4.14 uses the notion of secret establishment introduced in Definition 4.3. Therefore, the existence of a trace that establishes a shared secret according to Theorem 4.14 does not guarantee that secret establishment is possible in the presence of active adversaries.

We also use our implementation to obtain new impossibility results. For example, our implementation returns $\textsc{Impossible}$ for the theory describing pairing, signatures, and symmetric cryptography and thereby proves the following theorem.

Theorem 4.17. *Secret establishment is impossible for the combined equational theory* $(\Sigma_{\mathit{Sym}} \cup \Sigma_{\mathit{Sig}}, E_{\mathit{Sym}} \cup E_{\mathit{Sig}})$*, where the theory for signatures is defined as*

$$\Sigma_{\mathit{Sig}} = \{\mathit{sig}(\_, \_), \mathit{extr}(\_), \mathit{verify}(\_, \_), \mathit{pk}(\_)\}$$
$$E_{\mathit{Sig}} = \{\mathit{extr}(\mathit{sig}(m, k)) \simeq m, \mathit{verify}(\mathit{sig}(m, k), \mathit{pk}(k))) \simeq m\}.$$

We will later prove a combination result that allows us to consider the two theories separately.

4.4 Monoidal Theories

In this section we will investigate impossibility of secret establishment for monoidal theories. We will exploit that each monoidal theory $\mathcal{E}$ has an associated semiring $\boldsymbol{S}_{\mathcal{E}}$ and deducibility can be reduced to equation-solving in this algebraic structure. We mostly follow the presentation of [56] and [135] for basic facts about monoidal theories and deducibility in such theories.

Definition 4.18. *An equational theory is monoidal if it has an equational presentation $\mathcal{E} = (\Sigma, E)$ such that the following holds.*

1. *Σ consists of the binary function symbol $+$, the constant $\mathsf{0}$, and an arbitrary number of unary function symbols.*
2. *The function symbol $+$ is associative and commutative and the constant $\mathsf{0}$ is a unit for $+$. Formally, this means the equations $(x + y) + z \simeq x + (y + z)$ (A), $x + y \simeq y + z$ (C), and $x + \mathsf{0} = x$ (U) are in E.*
3. *All unary function symbols h are homomorphisms with respect to $+$ and $\mathsf{0}$, i.e., the equations $h(x + y) \simeq h(x) + h(y)$ and $h(\mathsf{0}) \simeq \mathsf{0}$ are in E.*

If for all terms t, there is a term s such that $t + s =_{\mathcal{E}} 0$, then the theory is a *group theory*.

Example 4.19. The following theories are monoidal:

- *ACU*: The signature Σ is $\{\mathsf{0}, +\}$ and E consists of A, C, and U. This theory can be used to model multisets with the empty multiset and multiset union.
- *ACUI*: The signature Σ is $\{\mathsf{0}, +\}$ and E consists of A, C, U, and $x + x \simeq x$ (I). This theory can be be used to model sets with the empty set and set union, which is idempotent.
- *ACUN*: The signature Σ is $\{\mathsf{0}, +\}$ and E consists of A, C, U, and $x + x \simeq 0$ (N). This theory can be be used to model the XOR operation, which is nilpotent.
- *AG*: The signature Σ is $\{\mathsf{0}, +, \text{-}\}$ and E consists of A, C, U, and $x + (\text{-}x) \simeq 0$. This theory can be be used to model abelian groups. The unary homomorphism - models the inverse operation and the equations characterizing that - is homomorphic are consequences of the other equations.
- *ACUh*, *ACUIh*, *ACUNh*, *AGh*: These theories can be obtained by extending the corresponding above theories with a homomorphism h and the required equations $h(\mathsf{0}) = \mathsf{0}$ and $h(x + y) \simeq h(x) + h(y)$.

Except for *ACU*, *ACUh*, *ACUI*, and *ACUIh*, all of these are group theories.

Definition 4.20. *A* semiring $\boldsymbol{S}$ *is a tuple $(S, +, 0, \cdot, 1)$ such that S is a set, $+ : S \times S \rightarrow S$, $0 \in S$, $\cdot : S \times S \rightarrow S$, $1 \in S$, and the following holds:*

1. *$(S, +, 0)$ is a commutative monoid, i.e., $+$ is associative and commutative and 0 is a unit for $+$.*
2. *$(S, \cdot, 1)$ is a monoid, i.e., $\cdot$ is associative and 1 is a unit for $\cdot$.*
3. *For all $a, b, c \in S$, $(a + b) \cdot c = a \cdot c + b \cdot c$ and $a \cdot (b + c) = a \cdot b + a \cdot c$.*

For each monoidal theory $\mathcal{E}$, there is an associated semiring $\boldsymbol{S}_{\mathcal{E}}$.

Definition 4.21. *Given a monoidal theory $\mathcal{E} = (\Sigma, E)$, the semiring $\boldsymbol{S}_{\mathcal{E}}$ is defined as $(S_{\mathcal{E}}, +, 0, \cdot, 1)$ where $S_{\mathcal{E}} = \mathcal{T}_{\Sigma}(\{\mathbf{1}\}) / =_{\mathcal{E}}$ for a variable $\mathbf{1}$, and $+$, 0, $\cdot$, and 1 are defined as follows.*

1. *$[s]_{\mathcal{E}} + [t]_{\mathcal{E}} = [s + t]_{\mathcal{E}}$, i.e., given $a, b \in \boldsymbol{S}_{\mathcal{E}}$, first choose $s, t \in \mathcal{T}_{\Sigma}(\{\mathbf{1}\})$ such that $a = [s]_{\mathcal{E}}$ and $b = [t]_{\mathcal{E}}$. Then take the equivalence class of the term $s + t$.*
2. *$0 = [\mathsf{0}]_{\mathcal{E}}$, i.e., the equivalence class of the constant $\mathsf{0}$.*
3. *$1 = [\mathbf{1}]_{\mathcal{E}}$, i.e., the equivalence class of the variable $\mathbf{1}$.*

4. $[s]_{\mathcal{E}} \cdot [t]_{\mathcal{E}} = [s\{t/\mathbf{1}\}]_{\mathcal{E}}$, i.e., the equivalence class of the term that results from replacing every occurrence of $\mathbf{1}$ in s with t.

If $\mathcal{E}$ is a group theory, then $\boldsymbol{S}_{\mathcal{E}}$ is a ring because $(S_{\mathcal{E}}, +, 0)$ is a abelian group. If $\mathcal{E}$ has commuting homomorphisms, then $(S_{\mathcal{E}}, \cdot, 1)$ is a commutative monoid. We use ψ_b to denote the function that maps a term $t \in \mathcal{T}_{\Sigma}(\{b\})$ to $[t^{[b \mapsto \mathbf{1}]}]_{\mathcal{E}} \in \boldsymbol{S}_{\mathcal{E}}$, i.e., ψ_b first replaces all occurrences of b in t by $\mathbf{1}$ and then takes the equivalence class with respect to $\mathcal{E}$.

Example 4.22. The semirings for ACU, $ACUI$, $ACUN$, and AG are isomorphic to the natural numbers $\mathbb{N}$, the semiring $(\{\emptyset, \mathbf{A}\}, \cup, \emptyset, \cap, \mathbf{A})$ where $\mathbf{A}$ is a nonempty set, the ring $\mathbb{Z}/2\mathbb{Z}$ (which is a finite field), and the integers $\mathbb{Z}$. Note that the semiring $(\{\emptyset, \mathbf{A}\}, \cup, \emptyset, \cap, \mathbf{A})$ is isomorphic to the boolean semiring $\mathbb{B} = (\{T, F\}, \vee, F, \wedge, T)$. For $ACUh$, $ACUIh$, $ACUNh$, and AGh, the corresponding semirings are the polynomials $\mathbb{N}[h]$, the polynomials $\mathbb{B}[h]$, the polynomials $(\mathbb{Z}/2\mathbb{Z})[h]$, and the polynomials $\mathbb{Z}[h]$.

For a fixed base $B = [b_1, ..., b_k]$ of names or variables b_i, we define the function ψ_B that maps a term $t \in \mathcal{T}_{\Sigma}(\{b_1, ..., b_n\})$ to a vector $v \in (\boldsymbol{S}_{\varepsilon})^n$ as follows. Write the term t as a sum $t_1 + ... + t_k$ such that $t_i \in \mathcal{T}_{\Sigma}(\{b_i\})$. This is always possible and the t_i are unique modulo $\mathcal{E}$. Then, $\psi_B(t) = (\psi_{b_1}(t_1), ..., \psi_{b_k}(t_k))$. It is not hard to verify that ψ_B is an isomorphism between $\mathcal{T}_{\Sigma}(B)/{=_{\mathcal{E}}}$ and $(\boldsymbol{S}_{\varepsilon})^n$. We use $\psi_B^{-1}(a)$ to denote t with $\psi_B(t) = a$, which is unique modulo $\mathcal{E}$. We now define the required operations to obtain a left $\boldsymbol{S}_{\mathcal{E}}$-module, an algebraic structure which is similar to a vector-space, but over a semiring instead of a field.

Theorem 4.23. *The structure $\big((\boldsymbol{S}_{\varepsilon})^n, +, \vec{0}, \cdot\big)$ is a left $\boldsymbol{S}_{\mathcal{E}}$-module for $+ : (\boldsymbol{S}_{\mathcal{E}})^n \times (\boldsymbol{S}_{\mathcal{E}})^n \rightarrow (\boldsymbol{S}_{\mathcal{E}})^n$ defined as component-wise addition, $\cdot : \boldsymbol{S}_{\mathcal{E}} \times (\boldsymbol{S}_{\mathcal{E}})^n \rightarrow (\boldsymbol{S}_{\mathcal{E}})^n$ defined as $s \cdot (t_1, ..., t_k) = (s \cdot t_1, ..., s \cdot t_k)$, and $\vec{0} = (0, ..., 0)$. For $a, b \in \boldsymbol{S}_{\mathcal{E}}$ and $\vec{c}, \vec{e} \in (\boldsymbol{S}_{\mathcal{E}})^n$, these operations satisfy the following equations required for left $\boldsymbol{S}_{\mathcal{E}}$-modules:*

1. $(a \cdot b) \cdot \vec{c} = a \cdot (b \cdot \vec{c})$
2. $(a + b) \cdot \vec{c} = a \cdot \vec{c} + b \cdot \vec{c} \;\; a$
3. $a \cdot (\vec{c} + \vec{e}) = a \cdot \vec{c} + a \cdot \vec{e}$
4. $a \cdot \vec{0} = \vec{0}$
5. $1 \cdot \vec{c} = \vec{c}$
6. $0 \cdot \vec{c} = \vec{0}$

We will now give an algebraic characterization of deducibility for monoidal theories.

Theorem 4.24. *Let $\mathcal{E} = (\Sigma, E)$ be a monoidal theory, $\tilde{n} = \{n_1, ..., n_k\} \subseteq \mathcal{N}$, $m \in \mathcal{T}_{\Sigma}(\tilde{n})$, and $\nu\, \tilde{n}.\sigma$ a frame such that names(range(σ)) $\subseteq \tilde{n}$. Let $B = [n_1, ..., n_k]$, $\vec{d} = \psi_B(m)$, and $\vec{c_i} = \psi_B(x_i\sigma)$ for dom(σ) $= \{x_1, ..., x_l\}$. Then $\nu\, \tilde{n}.\sigma \vdash_{\mathcal{E}} m$ if and only if there are $a_i \in \boldsymbol{S}_{\mathcal{E}}$ for $1 \leq i \leq l$ such that $a_1 \cdot \vec{c_1} + ... + a_l \cdot \vec{c_l} = \vec{d}$.*

Proof. $\Rightarrow$: Let $\nu\, \tilde{n}.\sigma \vdash_{\mathcal{E}} m$, then there is a recipe $\zeta \in \mathcal{T}_{\Sigma}(dom(\sigma))$ such that $\zeta\sigma =_{\mathcal{E}} m$ by Lemma 4.1 and Lemma 4.2. We can write $\zeta = \zeta_1 + ... + \zeta_l$ with $\zeta_i \in \mathcal{T}_{\Sigma}(\{x_i\})$. For $a_i = \psi_{x_i}(\zeta)$, we have $a_1 \cdot \vec{c_1} + ... + a_l \cdot \vec{c_l} = \vec{d}$.

$$\begin{aligned} a_1 \cdot \vec{c_1} + ... + a_l \cdot \vec{c_l} &= \psi_{x_1}(\zeta_1) \cdot \psi_B(x_i\sigma) + ... + \psi_{x_1}(\zeta_1) \cdot \psi_B(x_i\sigma) \\ &= \psi_B(\zeta_1\sigma) + ... + \psi_B(\zeta_l\sigma) \\ &= \psi_B(\zeta_1\sigma + ... + \zeta_l\sigma) \\ &= \psi_B(\zeta\sigma) \\ &= \psi_B(m) \\ &= \vec{d} \end{aligned}$$

($\Leftarrow$): Let $a_1 \cdot \vec{c_1} + \ldots + a_l \cdot \vec{c_l} = d$, then for $\zeta_i = \psi_{x_i}^{-1}(a_i)$ and $\zeta = \zeta_1 + \ldots + \zeta_l \in \mathcal{T}_\Sigma(dom(\sigma))$, we have $\zeta\sigma =_{\mathcal{E}} m$.

$$\begin{aligned}
\zeta\sigma &=_{\mathcal{E}} \zeta_1(\sigma|_{\{x_1\}}) + \ldots + \zeta_l(\sigma|_{\{x_l\}})\\
&=_{\mathcal{E}} \psi_{x_1}^{-1}(a_1)(\sigma|_{\{x_1\}}) + \ldots + \psi_{x_l}^{-1}(a_l)(\sigma|_{\{x_{1l}\}})\\
&=_{\mathcal{E}} \psi_B^{-1}(a_1 \cdot \vec{c_1}) + \ldots + \psi_B^{-1}(a_l \cdot \vec{c_l})\\
&=_{\mathcal{E}} \psi_B^{-1}(a_1 \cdot \vec{c_1} + \ldots + a_l \cdot \vec{c_l})\\
&=_{\mathcal{E}} \psi_B^{-1}(\vec{d}\,)\\
&=_{\mathcal{E}} m
\end{aligned}$$

□

It is not hard to see that the side conditions $m \in \mathcal{T}_\Sigma(\tilde{n})$ and $names(range(\sigma)) \subseteq \tilde{n}$ are always satisfied for $\nu\,\tilde{n}.\sigma \vdash_{\mathcal{E}} m$ if $\nu\,\tilde{n}.\sigma = \phi_{tr}$ for some minimal trace tr and m has been learned in tr.

We will first prove impossibility for group theories, then consider the remaining theories from Example 4.19 separately. We will provide an example protocol for *ACU* and show impossibility for *ACUI* and *ACUIh*.

4.4.1 Impossibility for Group Theories

We now consider group theories. Let $\mathcal{E} = (\Sigma, E)$ be a monoidal theory such that for every term t, there is a term s with $t + s =_{\mathcal{E}} 0$. For example, $s = t$ is such a term for the theory *ACUN* and $s = -t$ is such a term for the theory *AG*. For such theories $\mathcal{E}$, the associated semiring $\boldsymbol{S}_{\mathcal{E}}$ is a ring and $(S_{\mathcal{E}}, +, 0)$ is an abelian group. For each $a \in \boldsymbol{S}_{\mathcal{E}}$, we denote the unique element b with $a + b = 0$ with $-a$. We also use $-\vec{v}$ to denote the inverse of a vector $\vec{v}$.

Theorem 4.25. *There is no derivation trace using the equational theory $\mathcal{E}$ that establishes a shared secret. If $tr \in TR_{\mathcal{E}}$, $Learn(\mathbb{A}, m) \in tr$, and $Learn(\mathbb{B}, m) \in tr$, then $\phi_{tr} \vdash_{\mathcal{E}} m$.*

Proof. Let $m_1, \ldots, m_k$ be the sequence of transmitted messages, $N_{\mathbb{A}} = \{na_1, \ldots, na_{|N_{\mathbb{A}}|}\}$ and $N_{\mathbb{B}} = \{nb_1, \ldots, nb_{|N_{\mathbb{B}}|}\}$ the set of names created by $\mathbb{A}$ and $\mathbb{B}$, and m a shared secret. We assume that this sequence is of minimal length, i.e., it is not possible to establish a shared secret with fewer exchanged messages. We also assume that $\mathbb{A}$ sends the last message.

We fix the base $B = [na_1, \ldots, na_{|N_{\mathbb{A}}|}, nb_1, \ldots, nb_{|N_{\mathbb{B}}|}]$ and let $\vec{e} = \psi_B(m)$, and $\vec{c_i} = \psi_B(m_i)$. Since m is known by $\mathbb{A}$ and $\mathbb{B}$, there are $a_i, b_i \in \boldsymbol{S}_{\mathcal{E}}$ for $1 \leq i \leq k$ and $\vec{u}, \vec{v} \in (\boldsymbol{S}_{\mathcal{E}})^{|B|}$ such that $(\vec{u})_j$ is 0 for all $j > |N_{\mathbb{A}}|$, $(\vec{v})_j$ is 0 for all $j \leq |N_{\mathbb{A}}|$ and

$$\begin{aligned}
a_1 \cdot \vec{c_1} + \ldots + a_k \cdot \vec{c_k} + \vec{u} &= \vec{e} \qquad (1)\\
b_1 \cdot \vec{c_1} + \ldots + b_k \cdot \vec{c_k} + \vec{v} &= \vec{e} \qquad (2)\ .
\end{aligned}$$

Since $\vec{e}$ is a (shared) secret, $\vec{e} - b_k \cdot \vec{c_k}$ must also be a shared secret: It is deducible by $\mathbb{A}$ since we can replace a_k by $a_k - b_k$ in (1), it is deducible by $\mathbb{B}$ since we can replace b_k by 0 in (2), and $\vec{e} - b_k \cdot \vec{c_k}$ is secret since $\vec{e}$ is secret. If the adversary can deduce $\vec{e} - b_k \cdot \vec{c_k}$, then he can also deduce $\vec{e}$ by adding $b_k \cdot \vec{c_k}$ which is deducible. But this contradicts our assumption that it is impossible to establish a shared secret with fewer than k exchanged messages. $\mathbb{A}$ can learn $\vec{e} - b_k \cdot \vec{c_k}$ without c_k since this message is sent by $\mathbb{A}$ himself. $\mathbb{B}$ does not use c_k to learn $\vec{e} - b_k \cdot \vec{c_k}$ since $\vec{e} - b_k \cdot \vec{c_k} = b_1 \cdot \vec{c_1} + \ldots + b_{k-1} \cdot \vec{c}_{k-1} + \vec{v}$. □

4.4.2 Protocol for *ACU*

In this section, we present a protocol proposed by Rabi and Sherman [148] which uses an associative hash function. We will show that the protocol is also secure in the symbolic model with the equational theory ACU, i.e., with an associative and commutative function. As mentioned before, the semiring $\boldsymbol{S}_{ACU}$ is isomorphic to $\mathbb{N}$. For a base $B=[n_1,\ldots,n_k]$, the $\boldsymbol{S}_{ACU}$-module $((\boldsymbol{S}_{ACU})^k, +, 0, \cdot)$ is therefore isomorphic to $(B^\sharp, \cup^\sharp, \emptyset^\sharp, \cdot)$, where $B^\sharp$ denotes the set of multisets over B and $\cdot$ is defined as the n-fold multiset union.

Example 4.26. For the base $B=[n_1, n_2, n_3]$, the term $n_1+n_2+n_1$ corresponds to the vector $(2,1,0)\in\mathbb{N}^3$ and the operations in the associated $\mathbb{N}$-module are vector addition and scalar multiplication. Alternatively, we can interpret each vector (a_1,a_2,a_3) as the multiset where n_i occurs a_i times, e.g., $(2,1,0)$ corresponds to $\{n_1,n_1,n_2\}^\sharp$. Then, the vector addition corresponds to multiset union and the scalar multiplication $s\cdot m$ corresponds to the s-fold multiset union $m\cup^\sharp\ldots\cup^\sharp m$.

In the following, we use multiset notation for ACU and the associated semiring and module.

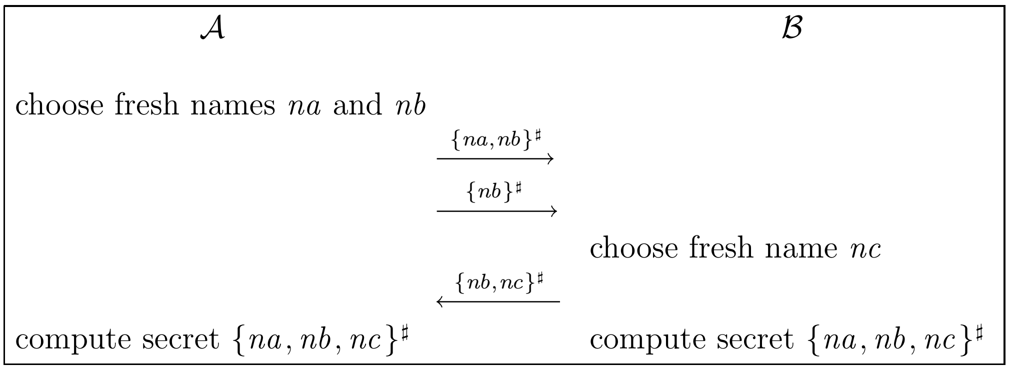

$\mathcal{A}$
$\mathcal{B}$
choose fresh names na and nb
$\{na,nb\}^\sharp$
$\{nb\}^\sharp$
choose fresh name nc
$\{nb,nc\}^\sharp$
compute secret $\{na,nb,nc\}^\sharp$
compute secret $\{na,nb,nc\}^\sharp$

Figure 4.4. Protocol from [148] for ACU.

It is not hard to see that

$$\nu\,\{na,nb,nc\}.\,\{\{na,nb\}^\sharp/x_1,\{nb\}^\sharp/x_2,\{nb,nc\}^\sharp/x_3\}\not\vdash_{ACU}\{na,nb,nc\}^\sharp$$

using Theorem 4.24. If not, there are $a_1,a_2,a_3\in\mathbb{N}$ such that

$$a_1\cdot\{na,nb\}^\sharp+a_2\cdot\{nb\}^\sharp+a_3\cdot\{nb,nc\}^\sharp=\{na,nb,nc\}^\sharp.$$

Since na and nc are included in the multiset on the right-hand-side, $a_1\geq 1$ and $a_2\geq 1$. But then, nb occurs twice on the left-hand-side. Note that the protocol can also be used with the theories A (an associative hash function as originally proposed), AC, and $ACUh$.

4.4.3 Impossibility for *ACUI* and *ACUIh*

The semiring $\boldsymbol{S}_{ACUI}$ is isomorphic to the boolean semiring $(\{F,T\},\vee,F,\wedge,T)$. For a base $B=[n_1,\ldots,n_k]$, the $\boldsymbol{S}_{ACUI}$-module $((\boldsymbol{S}_{ACU})^k,+,0,\cdot)$ is isomorphic to $(\mathcal{P}(B),\cup,\emptyset,\cdot)$ where $\mathcal{P}(B)$ denotes the set of subsets of B and $\cdot$ is defined such that $T\cdot S=S$ and $F\cdot S=\emptyset$.

Theorem 4.27. *There is no derivation trace using the equational theory ACUI that establishes a shared secret. If $tr\in TR_{ACUI}$, $Learn(\mathbb{A},\,m)\in tr$, and $Learn(\mathbb{B},\,m)\in tr$, then $\phi_{tr}\vdash_{ACUI}m$.*

Proof. Let m_1, ..., m_k be the sequence of exchanged messages and $N_{\mathbb{A}}$ and $N_{\mathbb{B}}$ the set of names created by $\mathbb{A}$ and $\mathbb{B}$. Then all messages m known by $\mathbb{A}$ can be written as

$$m = m_{i_1} \cup \ldots \cup m_{i_l} \cup n_{\mathbb{A}}$$

such that i_v is a subsequence of $1, \ldots, k$ and $n_{\mathbb{A}} \subseteq N_{\mathbb{A}}$. Analogously, all messages m known by $\mathbb{B}$ can be written as

$$m = m_{j_1} \cup \ldots \cup m_{j_u} \cup n_{\mathbb{B}}$$

such that j_v is a subsequence of 1, ..., k and $n_{\mathbb{B}} \subseteq N_{\mathbb{B}}$. Hence, we have $m \cap N_{\mathbb{A}} = m_{i_1} \cup \ldots \cup m_{i_l} \cup m_{j_1} \cup \ldots \cup m_{j_u}$, $m \cap N_{\mathbb{B}} = m_{i_1} \cup \ldots \cup m_{i_l} \cup m_{j_1} \cup \ldots \cup m_{j_u}$, and therefore

$$m = (m \cap N_{\mathbb{A}}) \cup (m \cap N_{\mathbb{B}}) = m_{i_1} \cup \ldots \cup m_{i_l} \cup m_{j_1} \cup \ldots \cup m_{j_u} .$$

Therefore, m is deducible from the exchanged messages. $\square$

Theorem 4.28. *There is no derivation trace using the equational theory ACUIh that establishes a shared secret. If $tr \in TR_{ACUIh}$, $Learn(\mathbb{A}, m) \in tr$, and $Learn(\mathbb{B}, m) \in tr$, then $\phi_{tr} \vdash_{ACUIh} m$.*

Proof. The proof for *ACUI* can be generalized to *ACUIh*. For a base $B = [n_1, \ldots, n_k]$, the $\boldsymbol{S}_{ACUIh}$-module is isomorphic to the set $\mathcal{P}(\{h^k(n) | k \in \mathbb{N}, n \in B\})$. In addition to the operations from the *ACUI* theory, the homomorphism h can be applied to a set $\{m_1, \ldots, m_k\}$ to deduce $h^k(\{m_1, \ldots, m_k\}) = \{h^k(m_1), \ldots, h^k(m_k)\}$. Let m_1, ..., m_k be the sequence of exchanged messages and $N_{\mathbb{A}}$ and $N_{\mathbb{B}}$ the set of names created by $\mathbb{A}$ and $\mathbb{B}$. All messages m known by $\mathbb{A}$ can written as

$$m = h^{e_1}(m_{i_1}) \cup \ldots \cup h^{e_l}(m_{i_l}) \cup n_{\mathbb{A}}$$

such that i_v is a subsequence of $1, \ldots k$, $e_v \in \mathbb{N}$, and $n_{\mathbb{A}} \subseteq \{h^k(n) | k \in \mathbb{N}, n \in N_{\mathbb{A}}\}$. Analogously, all messages m known by $\mathbb{B}$ can be written as

$$m = h^{f_1}(m_{j_1}) \cup \ldots \cup h^{f_u}(m_{j_u}) \cup n_{\mathbb{B}}$$

such that j_v is a subsequence of $1, \ldots k$, $f_v \in \mathbb{N}$, and $n_{\mathbb{B}} \subseteq \{h^k(n) | k \in \mathbb{N}, n \in N_{\mathbb{B}}\}$. Then by the same argument as in the previous proof, m is deducible from the exchanged messages since $m = h^{e_1}(m_{i_1}) \cup \ldots \cup h^{e_l}(m_{i_l}) \cup h^{f_1}(m_{j_1}) \cup \ldots \cup h^{f_u}(m_{j_u})$. $\square$

4.5 Combination Results for Impossibility

In Section 4.3.2, we have presented an automated method for deciding impossibility for subterm-convergent theories. Unfortunately, not all theories in cryptography are subterm-convergent. However, many relevant theories can be presented as the disjoint union of a subterm-convergent theory, such as *Sym* or *ASym*, and another theory that is not subterm-convergent, such as *ACUN*.

In this section, we consider an equational theory $\mathcal{E} = (\Sigma, E)$ that is the disjoint union of two equational theories $\mathcal{E}_1 = (\Sigma_1, E_1)$ and $\mathcal{E}_2 = (\Sigma_2, E_2)$, i.e., $E = E_1 \uplus E_2$ and $\Sigma = \Sigma_1 \uplus \Sigma_2$, where E_i only contains equations over Σ_i. We prove that secret establishment for such a theory $\mathcal{E}$ is possible if and only if it is possible in either $\mathcal{E}_1$ or $\mathcal{E}_2$. This allows us, for example, to combine our automatic method for a subterm-convergent subtheory with algebraic methods for the other subtheories.

4.5.1 Factors, Interface Subterms, Normalization

We now define the notions of *sign*, *alien subterm*, *factor*, and *interface subterm* for such a theory. These definitions are adopted from earlier work [56] and [46] on combination results for deducibility. Let $t \in \mathcal{T}_\Sigma(\mathcal{V} \cup \mathcal{N})$, then $sign(t) = i$ if $t = f(t_1, \ldots, t_k)$ for $f \in \Sigma_i$ and $sign(t) = 0$ if $t \in \mathcal{V} \cup \mathcal{N}$. A subterm u of t is *alien* if $sign(u) \neq sign(t)$.

If we say a term is in normal-from, then we mean with respect to $\downarrow_{\mathcal{E}}^{\succ}$.

Definition 4.29. *The factors of a term are its maximal alien subterms. We denote the set of factors of the term t with Fct(t). The interface subterms of a term are the subterms where a sign change occurs, i.e.,*

$$ISt(t) = \{t\} \cup \bigcup_{s \in Fct(t)} ISt(s)$$

Note that the ordered completion $O_{\mathcal{E}}$ of $\mathcal{E}$ corresponds to the disjoint union of the ordered completions $O_{\mathcal{E}_1}$ and $O_{\mathcal{E}_2}$. See [15] for details. We adopt the following three lemmas from [56] without providing our own proofs. These lemmas characterize the interaction between normalization, replacements, and deducibility.

Lemma 4.30. *If all factors of a message m are in normal form, then either* $sign(m) = sign(m\downarrow_{\mathcal{E}}^{\succ})$ *and* $Fct(m\downarrow_{\mathcal{E}}^{\succ}) \subseteq Fct(m) \cup \{n_{min}\}$ *or* $m\downarrow_{\mathcal{E}}^{\succ} \in Fct(m) \cup \{n_{min}\}$

The intuition behind this lemma is that if all factors of a term are normalized, then the normalization of the term does not affect its factors, i.e., either the factors of the normalized term are a subset of the original factors or the normalized term is a factor. Note that in both cases we must account for the case where free variables in the equations introduce n_{min}.

Lemma 4.31. *Let* $m = \zeta[f_1, \ldots, f_k]$ *be a message with factors* f_i *such that* $f_i = f_i\downarrow_{\mathcal{E}}^{\succ}$. *Let* ρ *be a bijective replacement that replaces the factors in m with fresh names. Then* $(m\downarrow_{\mathcal{E}}^{\succ})^\rho = (\zeta[f_1^\rho, \ldots, f_k^\rho])\downarrow_{\mathcal{E}}^{\succ}$.

This lemma states that normalization commutes with the replacement of normalized factors by fresh names. The proof is based on Lemma 4.30 and the fact that we consider consistent theories where names are in normal form. The next lemma states that deduction in the theory $\mathcal{E}_1$ is not affected by replacing interface subterms whose *sign* is 2 by fresh names. Of course, the lemma is also valid for swapped theory indices 1 and 2.

Lemma 4.32. *Let* $\nu\tilde{n}_1.\sigma$ *be a frame and m in* $\mathcal{T}_\Sigma(\mathcal{N})$ *such that m and all terms in range*(σ) *are in normal-form. Let* $F_2 = \{u | u \in ISt(range(\sigma) \cup \{m\}) \wedge sign(u) = 2\}$. *Let* $\tilde{n}_2$ *be a set of names not occurring in* $\nu\tilde{n}_1.\sigma$ *and m such that* $|F_2| = |\tilde{n}_2|$. *Let* $\rho\colon F_2 \to \tilde{n}_2$ *be a bijective replacement. Then* $\nu\tilde{n}_1.\sigma \vdash_{\mathcal{E}_1} m$ *if and only if* $\nu(\tilde{n}_1 \cup \tilde{n}_2).\sigma^\rho \vdash_{\mathcal{E}_1} m^\rho$.

To illustrate theses definitions and results, consider the equational theory $E = (\Sigma, E)$ for $\Sigma = \Sigma_{ACUN} \cup \Sigma_{Sym}$ and $E = E_{ACUN} \cup E_{Sym}$.

Example 4.33. The factors of $m = enc(\langle n_1, n_2\rangle + n_2, n_3)$ are $\langle n_1, n_2\rangle + n_2$ and n_3. The set of interface subterms of $ISt(m)$ is $\{m, \langle n_1, n_2\rangle + n_2, n_3, \langle n_1, n_2\rangle, n_1, n_2\}$. An example of the first case in Lemma 4.30 is $enc(dec(enc(n_1 + n_2, k), k), k')\downarrow_{\mathcal{E}}^{\succ} = enc(n_1 + n_2, k)$ and an example of the second case is $dec(enc(n_1 + n_2, k), k)\downarrow_{\mathcal{E}}^{\succ} = n_1 + n_2$.

4.5.2 Combination Result

We first prove two lemmas that are required for our main result. The first lemma is similar to Lemma 4.8 and states that all interface subterms of learned messages that are not equal to n_{min} must have been learned by one of the agents.

Lemma 4.34. *Let $tr \in TR_{\mathcal{E}}$, $A \in \mathcal{A}$, and $m, u \in \mathcal{M}_\Sigma$ such that $Learn(A, m) \in tr$ and $u \in ISt(m) \setminus \{n_{min}\}$. Then there is a $B \in \mathcal{A}$ that constructs u in tr.*

Proof. We prove the lemma by induction over $TR_{\mathcal{E}}$. For every rule except *App*, the statement of the lemma directly follows from the induction hypothesis. So let $tr \in TR_{\mathcal{E}}$ such that for all $Learn(A,m) \in tr$ and $u \in ISt(m) \setminus n_{min}$, there is a $B \in \mathcal{A}$ that constructs u in tr. Now consider the trace $tr \cdot Learn_{App(f(m_1,\ldots,m_k))}(A, m)$ with $f(m_1,\ldots,m_k){\downarrow}_{\mathcal{E}}^{\succ} = m$. Then we consider two cases separately. First, if $sign(m) = sign(f)$, then

$$Fct(m) \subseteq Fct(f(m_1,\ldots,m_k)) \cup \{n_{min}\}$$

by Lemma 4.30. Hence $ISt(m) \subseteq \{m, n_{min}\} \cup ISt(\{m_1, \ldots, m_k\})$. For $u = m$, there is a construction event. For $u \in ISt(m_i)$, we can apply the induction hypothesis since there is an event $Learn(A, m_i)$ in tr. Second, if $sign(m) \neq sign(f)$, then

$$m \in Fct(f(m_1,\ldots,m_k)) \cup \{n_{min}\} \subseteq \{m_1,\ldots,m_k,n_{min}\} \cup Fct(\{m_1,\ldots,m_k\}).$$

Hence, $ISt(m) \subseteq \{m_1, \ldots, m_k, n_{min}\} \cup ISt(\{m_1, \ldots, m_k\})$ and we can therefore apply the induction hypothesis. ☐

Lemma 4.35. *Let $tr \in TR_{\mathcal{E}}$, then n_{min} is not learned in tr.*

Proof. If there is a trace tr where one of the agents learns n_{min}, then there must be a term $t \in \mathcal{T}_\Sigma(\mathcal{N} \setminus \{n_{min}\})$ with $t =_{\mathcal{E}} n_{min}$. The term t can be obtained from the trace tr by considering the $Learn_{Fresh}$-events and $Learn_{App}$-events that are used to construct t. Since $\mathcal{E}$ is stable under replacement of names by terms, we obtain $t =_{\mathcal{E}} n$ for some name $n \notin names(t) \cup \{n_{min}\}$. But this implies $n =_{\mathcal{E}} n_{min}$ by transitivity which contradicts our assumption that $\mathcal{E}$ is consistent. ☐

Using these lemmas, we now prove our combination result for impossibility.

Theorem 4.36. *Let $\mathcal{E} = (E, \Sigma)$ be the disjoint union of the equational theories $\mathcal{E}_1 = (\Sigma_1, E_1)$ and $\mathcal{E}_2 = (\Sigma_2, E_2)$. If there is $tr \in TR_{\mathcal{E}}$ that establishes a secret, then there is either a trace $tr \in TR_{\mathcal{E}_1}$ or a trace $tr \in TR_{\mathcal{E}_2}$ that establishes a secret.*

Proof. If there is a trace in $TR_{\mathcal{E}}$ that establishes a shared secret, then there is also a minimal trace $tr \in TR_{\mathcal{E}}^{min}$ that establishes a shared secret s. The last event in tr is $Learn_{App(f(m_1,\ldots,m_k))}(\mathbb{B}, s)$ for $f \in \Sigma$ and $m_i \in \mathcal{M}_\Sigma$. We assume without loss of generality that $f \in \Sigma_1$. We show that there is a trace $tr_\rho \in TR_{\mathcal{E}_1}^{min}$ that establishes a translated secret s^ρ.

We first prove for a given minimal trace $tr \in TR_{\mathcal{E}}^{min}$, we can find an injective replacement ρ from $\{a | a \in ISt(tr) \wedge sign(a) = 2\}$ to $\mathcal{N} \setminus (names(tr) \cup \{n_{min}\})$ and a translated trace $tr_\rho \in TR_{\mathcal{E}_1}^{min}$ such that tr_ρ can be obtained from tr by replacing every event $Learn_{App(f(m_1,\ldots,m_k))}(A, m)$ where $sign(m) = 2$ with $Learn_{Fresh}(A, m^\rho)$ and every other event $Ev(A, m)$ with $Ev(A, m^\rho)$.

We prove this by induction on $TR_{\mathcal{E}}^{min}$. The statement obviously holds for the empty trace. Assume that for a given minimal trace tr, there are ρ and tr_ρ with the desired properties. We now consider the rules to extend tr.

Public. A minimal trace does not contain $Learn_{Public}$-events.

Recv. The event $Learn_{Recv}(A, m)$ is added to tr. $tr_\rho \cdot Learn_{Recv}(A, m^\rho) \in TR_{\mathcal{E}_1}^{min}$ since the required *Send*-event must be in tr and the translated *Send*-event in tr_ρ.

Send. The event $Send(A, m)$ is added to tr. $tr_\rho \cdot Send(A, m^\rho) \in TR_{\mathcal{E}_1}^{min}$ since the required *Learn*-event must be in tr and the translated *Learn*-event in tr_ρ.

Fresh. The event $Learn_{Fresh}(A,\ n)$ is added to the trace tr. If $n \notin range(\rho)$, then $tr_\rho \cdot Learn_{Fresh}(A, n) \in TR_{\mathcal{E}_1}^{min}$ has the desired properties since $dom(\rho) \cap \mathcal{N} = \emptyset$. If there is m with $\rho(m) = n$, then we have to modify ρ. Let $n' \in \mathcal{N} \setminus (names(tr_\rho) \cup \{n_{min}\} \cup range(\rho))$ and $\rho' = \rho[m \mapsto n']$. Then ρ' and $tr_{\rho'}$ have the desired properties and $tr_{\rho'} \cdot Learn_{Fresh}(A, n) \in TR_{\mathcal{E}_1}^{min}$.

App. The event $Learn_{App(f(m_1,\ldots,m_k))}(A, m)$ is added to tr. We distinguish two case. First, if $sign(f) = 1$, then $tr_\rho \cdot Learn_{App(f(m_1^\rho,\ldots,m_k^\rho))}(A, m^\rho) \in TR_{\mathcal{E}_1}^{min}$ since

$$m^\rho = (f(m_1,\ldots,m_k){\downarrow}_{\mathcal{E}}^{\succ})^\rho = f(m_1^\rho,\ldots,m_k^\rho){\downarrow}_{\mathcal{E}}^{\succ}$$

by Lemma 4.31. Second, if $sign(f) = 2$, then $sign(m) = 2$. Otherwise, it holds that $m \in Fct(f(m_1, \ldots, m_k)) \cup \{n_{min}\}$. Since $m = n_{min}$ is impossible by Lemma 4.35, $m \in ISt(m_1,\ \ldots,\ m_k)$ and there must be already a construction event for m in tr by Lemma 4.34. Hence, there are two construction events for m in the extended trace. This is only possible if m is the shared secret. But this contradicts our assumption that the shared secret is learned by applying f with $sign(f) = 1$. We therefore choose a fresh name $n \notin (names(tr) \cup range(\rho))$ and define $\rho' = p[m \mapsto n]$. Then ρ' and $tr_{\rho'} \cdot Learn_{Fresh}(A, m^{\rho'}) \in TR_{\mathcal{E}_1}^{min}$ have the desired properties.

This implies that for for every $tr \in TR_{\mathcal{E}}^{min}$ where the shared secret s is established using $f \in \Sigma_1$ in the last event, we can find a replacement ρ such that $tr_\rho \in TR_{\mathcal{E}_1}^{min}$ and tr_ρ establishes the shared secret s^ρ. Since tr_ρ has been obtained from tr by applying ρ to all messages, $sent(tr_\rho) = sent(tr)^\rho$ and the same holds the knowledge of $\mathbb{A}$ and $\mathbb{B}$ in the two traces. Hence, both know s^ρ. Since, $\nu\ restricted(tr).\sigma_{[sent(tr)]} \not\vdash_{\mathcal{E}} s$ which implies $\nu\ restricted(tr).\sigma_{[sent(tr)]} \not\vdash_{\mathcal{E}_1} s$ and $restricted(tr) \cup range(\rho) = restricted(tr_\rho)$, we can use Lemma 4.32 to obtain $\nu\, restricted(tr_\rho).\sigma_{[sent(tr_\rho)]} \not\vdash_{\mathcal{E}_1} s^\rho$. □

4.6 Summary

We have collected all impossibility results from this paper in Table 4.1. Moreover, we have augmented the table with possibility results from the literature, thereby providing an overview of existing results. Note that there are theories where, to the best of our knowledge, the problem is still open. For example, there are no such results for the theory of (nonabelian) groups, blind signatures, and homomorphic encryption.

The results presented are for minimal disjoint theories, in the sense that they cannot sensibly be further decomposed. For disjoint equational theories where secret establishment is impossible, we can apply Theorem 4.36 to obtain an impossibility result for the union of the two theories. For example, secret establishment using the equational theory *Sym* that models symmetric encryption, pairing, and a hash function combined with the theory *ACUN* for XOR is impossible. Here, our combination result allows us to use different proof methods for the subtheories. Namely, our decision procedure for the subterm-convergent theory *Sym* and the proof for group theories for *ACUN*. Note that neither of these methods can be used for the union of the two theories. Another application of our combination result is that we can further optimize the decision procedure from Section 4.3.2. Namely, we can split a subterm-convergent theory into disjoint subtheories that can be checked separately. For example, as presented in the table, the theory *Sym* can be split into the theories for $\mathcal{E}_1$ for pairing, $\mathcal{E}_2$ for symmetric encryption, and $\mathcal{E}_3$ for the theory for the free function symbol h. We can then call FIND-DERIVATION-TRACES for each of the subtheories and need only check deducibility for $\vdash_{\mathcal{E}_i}$ in the given call.

Theory	Possible?	Protocols/Proof technique
Free function symbols (e.g. a hash function h)	No	Subterm-convergent
Pairing	No	Subterm-convergent
Symmetric encryption	No	Subterm-convergent
Signatures	No	Subterm-convergent
Public-key encryption	Yes	Key transport in [154]
A, AC, ACU, $ACUh$	Yes	Key agreement in [148]
$ACUN$, $ACUNh$, AG, AGh	No	Group theories
$ACUI$, $ACUIh$	No	Proof for these monoidal theories
DH-exponentiation	Yes	Key agreement [68] and Key transport [143]

Table 4.1. Impossibility Results and Protocols.

Chapter 5
Analysis of Physical Protocols

In this chapter, we describe our framework for the interactive analysis of physical protocols. The underlying Isabelle/HOL formalization, which contains proofs for all lemmas and theorems in this chapter, can be found at [158]. We first motivate and describe our formal model excluding XOR and overshadowing. Then, we present three case studies. Afterwards, we present our extension with XOR and overshadowing. Finally, we apply the extended framework to the Brands-Chaum distance bounding protocol, modeling the rapid bit-exchange phase by single challenge and response messages.

5.1 Formal Model

In this section, we present our formal model for analyzing physical protocols. First, we list the modeled concepts and our modeling assumptions. Then, we present our model and sketch its formalization in Isabelle/HOL. We start by formalizing agents and transmitters. Then we formalize physical and communication distance, messages, and events and traces. Next, we formalize the network, intruders, and the protocol. Finally, we prove some protocol-independent properties of our formalization. Some technical details of our formalization are described in Section 5.5 and the remaining details can be found in the Isabelle/HOL theory files.

5.1.1 Modeled Concepts

Agents. We consider a set of communicating agents, consisting of honest and dishonest agents. Honest agents follow the protocol rules, whereas dishonest agents (also called intruders) can deviate arbitrarily from the protocol. Each agent has a fixed location and a set of transmitters and receivers. Agents possess initial knowledge, such as their own private keys and the public keys of other agents, which they can use to construct new messages or to analyze received messages.

Network. We model an unreliable network connecting agents' transmitters and receivers as a communication matrix. The matrix describes the connectivity between transmitters and receivers. An agent Alice can send messages directly to an agent Bob if and only if there is a corresponding entry in the communication matrix. The matrix entries express the lower bounds on the signal propagation time from a transmitter to a receiver. They therefore capture not only whether direct communication is possible, but also the different communication technologies with different signal propagation speeds, e.g., radio and ultrasound technologies. Modeling an unreliable network allows us to capture message deletion (jamming) and transmission failures. Our model distinguishes between the topology associated with the agents' locations and the topology associated with the network. Whereas the physical distance corresponds to the Euclidean distance, the network topology describes signal paths not necessarily corresponding to the line-of-sight paths between senders and receivers.

For example, consider network cable rolls or signal reflections. However, to accurately model reality, the communication model must be consistent with basic physical laws. In particular, the smallest transmission time possible between transmitters and receivers corresponds to the time required for line-of-sight transmission. Since these laws are universal, our model applies to any kind of network where the network topology can be described by a fixed communication matrix.

Time. Protocols, such as the authenticated ranging example from Section 2.1.4.2, measure time to make statements about distances. As a result, our model must correctly describe temporal dependencies between related events, such as a send event preceding a receive event and agents must be able to access clocks to associate events with time. We achieve this by tagging every event with a corresponding timestamp. We use rules that account for arbitrary offsets of local clocks to model temporal dependencies and clock access by agents

Intruder Model. In order to reason formally about properties of security protocols, we must precisely define the capabilities of the intruder. We therefore need to specify the intruder's capabilities in terms of network control and cryptographic capabilities. The most prominent and most widely used intruder model is the so called Dolev-Yao model [69]. In this model, an intruder completely controls the communication network in the sense that he can overhear, remove, and delay any message sent by honest agents. Additionally, he can insert any message that he is able to construct according to his cryptographic capabilities. In terms of cryptographic capabilities, the Dolev-Yao intruder implements the so called perfect cryptography assumption.

In our model, the intruder's *cryptographic capabilities* correspond to those of the Dolev-Yao intruder. However, in terms of network control, our communication model is subject to physical restrictions, such as transmission time and network topology. These constraints on communication apply both to honest agents and intruders. An individual intruder can therefore only intercept messages at his location. Moreover, colluding intruders cannot instantaneously exchange known messages. The message exchange between intruders is also subject to limitations incurred by the network topology which is formalized by the communication matrix. This models reality, where the attackers' ability to observe and communicate messages is determined by their locations, mutual distances, and by their transmitters and receivers. Note that these extensions are essential for modeling protocols that involve physical properties of the environment such as time and location. Such protocols fall outside the scope of standard symbolic protocol models based on the Dolev-Yao intruder. This is understandable: the Dolev-Yao model was developed for classical security protocols which do not rely on properties of the physical environment. In Section 5.3, we will extend this model to account for partial overshadowing of messages.

5.1.2 Agents and Transmitters

Agents are either honest or intruders. We model infinitely many agents of each kind by using the set of natural numbers $\mathbb{N}$ as agent identifiers.

$$\textbf{datatype}\ \mathit{agent} = \mathsf{Honest}\ \mathbb{N} \mid \mathsf{Intruder}\ \mathbb{N}$$

We refer to agents using capital letters like A and B. We also write H_A and H_B for honest agents and I_A and I_B for intruders, when we require this distinction. Each agent has a set of transmitters and receivers.

$$\textbf{datatype}\ \mathit{transmitter} = \mathsf{Tx}\ \mathit{agent}\ \mathbb{N}$$

Given an agent A and an index i, the constructor Tx returns a transmitter denoted Tx_A^i. The number of usable transmitters can be restricted by specifying that some transmitters cannot communicate with any receivers. Receivers are formalized analogously.

$$\textbf{datatype } \mathit{receiver} = \mathsf{Rx}\ \mathit{agent}\ \mathbb{N}$$

We use Rx_A^i to denote A's receiver with index i.

5.1.3 Physical and Communication Distance

The function *loc* assigns to each agent A a location $loc_A \in \mathbb{R}^3$. Using the euclidian metric on $\mathbb{R}^3$, we define the physical distance between two agents A and B as $|\mathrm{loc}_A - loc_B|$.

The line-of-sight distance between two agents A and B is the shortest path, taken, for example, by electromagnetic waves when there are no obstacles. We define the line-of-sight communication distance as the time it takes for a radio signal to travel this path using speed of light c, i.e.,

$$\mathit{cdist}_{LoS}(A, B) = \frac{|loc_A - loc_B|}{c}.$$

The value $\mathit{cdist}_{LoS}(A, B)$ only depends on A and B's locations and is independent of the network topology. We model the network topology using the function $\mathit{cdist}_{Net} : \mathit{transmitter} \times \mathit{receiver} \rightarrow \mathbb{R}_{\geq 0} \cup \{\perp\}$. Its value reflects the communication medium used by the given transceivers, obstacles between the transceivers, and other environmental factors. $\mathit{cdist}_{Net}(Tx_A^i, Rx_B^j) = \perp$ denotes that Rx_B^j cannot receive transmissions from Tx_A^i. In contrast, $\mathit{cdist}_{Net}(Tx_A^i, Rx_B^j) = t$, where $t \in \mathbb{R}_{\geq 0}$, denotes that Rx_B^j can receive transmissions from Tx_A^i after a minimum delay of t time units. Since we assume that information cannot propagate faster than with the speed of light, we require that for all A, B, i, and j,

$$\mathit{cdist}_{LoS}(A, B) \leq \mathit{cdist}_{Net}(Tx_A^i, Rx_B^j).$$

In Isabelle/HOL, we model *loc* as an uninterpreted function. That is, we give *loc* a type, but do not provide a concrete interpretation. Similarly, cdist_{Net} is uninterpreted, but we restrict the class of possible interpretations by additionally requiring the previously mentioned property: faster-than-light communication is impossible. Further assumptions about the agents' locations and the network topology required for analyzing protocols can be added as local assumptions in security proofs. As an example of such an additional assumption, consider the ultrasound distance bounding protocol and its security properties described in Section 5.2.2. For the protocol to have the expected security property, we must assume that there is no adversary in a given area (the so called *private space*) around an honest agent. This is therefore modeled as an additional assumption in the corresponding security proof. Hence, our results apply to all possible locations of agents and network topologies that fulfill the corresponding assumptions.

Example 5.1. The following example relates the communication distance and the physical distance. The left side of Figure 5.1 illustrates the nodes and their environment. Here, edges denote line-of-sight connections which correspond to shortest paths in Euclidean space and are labeled with the corresponding values of the cdist_{LoS} function. Note that cdist_{LoS} is defined in terms of the physical location of nodes and neither depends on communication obstacles nor physical properties of the communication medium. Also note that cdist_{LoS} is symmetric.

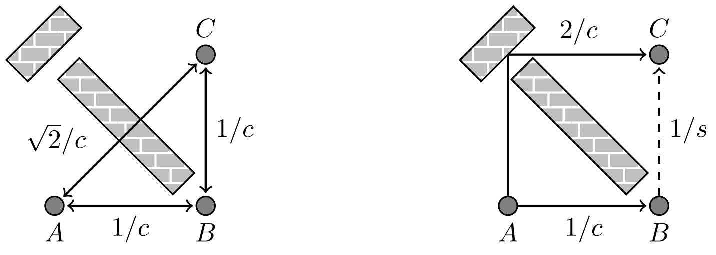


Figure 5.1. Physical (left) and network topology (right).

The right side of Figure 5.1 illustrates the communication distance associated with the network topology, where A possesses only a radio transmitter, B possesses a radio receiver and an ultrasound transmitter, and C possesses a radio receiver and an ultrasound receiver. The dashed line represents the ultrasonic link, where signals travel at the speed of sound s. The diagonal wall in the middle prevents line-of-sight communication from A to C. However, reflections off the wall in the upper left corner enable C to receive the signal. So the two notions of distance only coincide for the link from A to B, which uses line-of-sight communication at the speed of light c.

5.1.4 Messages

A message is either atomic or composed. Atomic messages are agent names, times, numbers, nonces, and keys represented by natural numbers. Composed messages are hashes, pairs, and encrypted messages.

$$\begin{aligned}\textbf{datatype}\ \mathit{msg} = {} & \mathsf{Agent}\ \mathit{agent} \mid \mathsf{Time}\ \mathbb{R} \mid \mathsf{Number}\ \mathbb{N} \mid \mathsf{Nonce}\ \mathit{agent}\ \mathbb{N} \\ & \mid \mathsf{Key}\ \mathit{key} \mid \mathsf{Hash}\ \mathit{msg} \mid \mathsf{Pair}\ \mathit{msg}\ \mathit{msg} \mid \mathsf{Crypt}\ \mathit{key}\ \mathit{msg}\end{aligned}$$

Nonces model random unguessable bitstrings and are tagged with the name of the agent who created them and a unique index. Tagging nonces with the creator's name ensures that nonces created by different agents never collide. Indeed, even colluding intruders must communicate to share a nonce. To ensure that nonces created by the same agent do not collide, the *used* function introduced below is used. Similar to nonces, keys are tagged with a unique value of type *key*, which is a type synonym for $\mathbb{N}$. The set of keys is partitioned into those used for signing, asymmetric encryption, and symmetric encryption. An inverse operator $(\_)^{-1}$ is defined for the three key types. It is the identity function on symmetric keys. The constructor Crypt denotes signing, asymmetric encryption, or symmetric encryption, depending on the key used. We write $\{m\}_k$ for Crypt $k\ m$, $\langle m, n\rangle$ for Pair $m\ n$ and $mac_k(m)$ for $\langle m, \mathsf{Hash}\ \langle k, m\rangle\rangle$. If sk is a signing key, we also write $sig(m, sk)$ instead of $\{m\}_{sk}$.

Given a set of messages, an agent can deduce new messages by decomposing and composing given messages. We formalize this message deduction capability with the inductively defined operator DM: $agent \to msg\ set \to msg\ set$. The rules defining DM are given in Figure 5.2 and specify hashing, projection on pairs, pairing, encryption, message decryption, and the generation of agent names, nonces, times, and numbers. For example, the DEC-rule states that if an agent A can deduce the ciphertext $\{m\}_k$ and the decryption key $(\mathsf{Key}\ k)^{-1}$, then he can also deduce the cleartext m.

$$\textsc{Inj}\ \frac{m \in M}{m \in DM_A(M)} \qquad \textsc{Hash}\ \frac{m \in DM_A(M)}{\mathsf{Hash}\ m \in DM_A(M)} \qquad \textsc{Fst}\ \frac{\langle m_1, m_2\rangle \in DM_A(M)}{m_1 \in DM_A(M)}$$

$$\textsc{Pair}\ \frac{m_1 \in DM_A(M) \quad m_2 \in DM_A(M)}{\langle m_1, m_2\rangle \in DM_A(M)} \qquad \textsc{Snd}\ \frac{\langle m_1, m_2\rangle \in DM_A(M)}{m_2 \in DM_A(M)}$$

$$\textsc{Enc}\ \frac{m \in DM_A(M) \quad \mathsf{Key}\ k \in DM_A(M)}{\{m\}_k \in DM_A(M)} \qquad \textsc{Agent}\ \frac{}{\mathsf{Agent}\ a \in DM_A(M)}$$

$$\textsc{Dec}\ \frac{\{m\}_k \in DM_A(M) \quad (\mathsf{Key}\ k)^{-1} \in DM_A(M)}{m \in DM_A(M)} \qquad \textsc{Nonce}\ \frac{}{\mathsf{Nonce}\ A\ n \in DM_A(M)}$$

$$\textsc{Time}\ \frac{}{\mathsf{Time}\ t \in DM_A(M)} \qquad \textsc{Number}\ \frac{}{\mathsf{Number}\ n \in DM_A(M)}$$

Figure 5.2. Rules defining $DM_A(M)$.

5.1.5 Events and Traces

An *event* corresponds to an agent sending or receiving a message or making a claim.

$$\textbf{datatype}\ \mathit{event} = \mathsf{Send}\ \mathit{transmitter}\ \mathit{msg}\ (\mathit{msg\,list}) \mid \mathsf{Recv}\ \mathit{receiver}\ \mathit{msg} \mid \mathsf{Claim}\ \mathit{agent}\ \mathit{msg}$$

A *trace* is a list of timed events, where a timed event $(t, e) \in \mathbb{R} \times \mathit{event}$ pairs a timestamp with an event. Events are associated to agents and thereby to the agent's location. This association is either direct for Claim-events or indirect via the association of transceivers to agents for Send-events and Recv-events. The timed event $(t_S, \mathsf{Send}\ Tx_A^i\ m\ L)$, for example, denotes that the agent A sent a message m using his transmitter with index i at timepoint t_S and has associated the protocol state L with the event. The list of messages L models protocol state information and can contain messages used to construct m and times of preceding events. Such a Send-event may induce multiple Recv-events of the form $(t_R,\ \mathsf{Recv}\ Rx_B^j\ m)$, where the timepoints t_R and receivers Rx_B^j must be consistent with the network topology. A Claim-event models a belief or a conclusion made by a protocol participant, formalized as a message. For example, after successfully completing a run of the authenticated ranging protocol from Section 2.1.4.2 with Bob, Alice concludes at some time t_C that d_{AB} is an upper bound on her distance to Bob. We model this by adding the event $(t_c, \mathsf{Claim}\ A\ \langle B, d_{AB}\rangle)$ to the trace. The protocol is therefore secure if the claim about the upper bound on the mutual distance is valid for all traces containing such a Claim-event.

Note that the timestamps used in traces use the notion of absolute time. However, agents' clocks may deviate arbitrarily from absolute time. We therefore translate the absolute timestamps to model the agent's local view. We describe this translation in Section 5.1.6.

Knowledge and Used Messages. Each agent A holds some some initial knowledge, denoted by $\mathit{initKnows}_A$, which depends on the executed protocol. In a system run with trace tr, the knowledge of an agent A is defined as the union of the initial knowledge and all received messages, i.e.,

$$\mathit{knows}_A(tr) = \{m \mid \exists k\ \ t.\, (t, \mathsf{Recv}\ Rx_A^k\ m) \in tr\} \cup \mathit{initKnows}_A\,.$$

From the known messages, A can deduce all messages in $DM_A(knows_A(tr))$. For a given term m, the set of (extractable) *parts* and the set of (syntactic) *subterms* are defined inductively by the rules in Figure 5.3.

$$
\begin{array}{ccc}
\dfrac{}{m \in subterms(m)} & \dfrac{\langle m_1, m_2\rangle \in subterms(m)}{m_i \in subterms(m)} & \dfrac{\mathsf{Hash}\ c \in subterms(m)}{c \in subterms(m)} \\[2ex]
\dfrac{\{c\}_k \in subterms(m)}{c \in subterms(m)} & \dfrac{\{c\}_k \in subterms(m)}{k \in subterms(m)} & \\[2ex]
\dfrac{}{m \in parts(m)} & \dfrac{\langle m_1, m_2\rangle \in parts(m)}{m_i \in parts(m)} & \dfrac{\{c\}_k \in parts(m)}{c \in parts(m)}
\end{array}
$$

Figure 5.3. Rules defining *subterms* and *parts*.

We use *subterms* to define the set of messages appearing in a trace *tr*.

$$used(tr) = \{n \mid \exists A\ k\ t\ m.\ (t, \mathsf{Send}\ Tx_A^k\ m) \in tr \wedge n \in subterms(m)\}$$

5.1.6 Network, Intruder, and Protocols

We now describe our rules that inductively define the set of traces $Tr(proto)$ for the execution of a protocol *proto* together with arbitrary intruders sharing a network. The base case, modeled by the Nil-rule in Figure 5.4 states that the empty trace is a valid trace for all protocols. The other rules describe how valid traces can be extended. The rules model the network behavior, the possible actions of the intruder, and the actions taken by honest agents executing the protocol.

$$
\textsc{Net}\ \dfrac{\begin{array}{c} tr \in Tr(proto) \quad t_R \geq maxtime(tr) \\ (t_S, \mathsf{Send}\ Tx_A^i\ m\ L) \in tr \\ cdist_{Net}(Tx_A^i, Rx_B^j) = t_{AB} \\ t_{AB} \neq \bot \quad t_R \geq t_S + t_{AB} \end{array}}{tr \cdot (t_R, \mathsf{Recv}\ Rx_B^j\ m) \in Tr(proto)}
\qquad
\textsc{Nil}\ \dfrac{}{[\,] \in Tr(proto)}
$$

$$
\textsc{Fake}\ \dfrac{\begin{array}{c} tr \in Tr(proto) \quad t \geq maxtime(tr) \\ m \in DM_{I_A}(knows_{I_A}(tr)) \end{array}}{tr \cdot (t, \mathsf{Send}\ Tx_{I_A}^i\ m\ [\,])}
$$

$$
\textsc{Proto}\ \dfrac{\begin{array}{c} tr \in Tr(proto) \quad t \geq maxtime(tr) \quad step \in proto \\ (act, m) \in step(view(H_A, tr), H_A, ctime(H_A, t)) \quad m \in DM_{H_A}(knows_{H_A}(tr)) \end{array}}{tr \cdot (t, translateEv(H_A, act, m))}
$$

Figure 5.4. Rules defining $Tr(proto)$.

Network Rule. The Net-rule models message transmission from transmitters to receivers, constrained by the network topology. A Send-event from a transmitter may induce a Recv-event at a receiver only if the receiver can receive messages from the transmitter. The time between these events is bounded below by the communication distance between the transmitter and the receiver. If there is a Send-event in the trace *tr* and the Net-rule's premises are fulfilled, a corresponding Recv-event is appended to the trace. The restriction on connectivity and transmission delay are ensured by $t_{AB} \neq \bot$ and $t_R \geq t_S + t_{AB}$. Here, t_{AB} is the communication distance between the receiver and transmitter, t_S is the sending time, and t_R is the receiving time.

Note that one Send-event can result in multiple Recv-events at the same receiver at different times. This is allowed because $\mathit{cdist}_{\mathit{Net}}$ models the minimal communication distance and messages may also arrive later, for example due to the reflection of the signal carrying the message. Moreover, a Send-event can result in multiple Recv-events at different receivers, modeling for example broadcast communication. Finally, note that transmission failures and jamming by an intruder, resulting in message loss, are modeled by traces where the NET-rule is not applied for a given Send-event and receiver, even though all premises are fulfilled.

The timestamps associated with Send-events and Recv-events denote the starting times of message transmission and reception. Thus, our network rule captures the latency of links, but not the message-transmission-time, which also depends on the message's size and the transmission speed of the transmitter and the receiver. Some implementation-specific attacks, such as those described in [156] and [49] are therefore not captured in our model. Also note that we do not capture partial overshadowing of messages by the adversary. In Section 5.3, we will adapt the NET-rule to account for this.

The premise $t \geq \mathit{maxtime}(\mathit{tr})$, where t denotes the timestamp associated with the new event and $\mathit{maxtime}(\mathit{tr})$ denotes the latest timestamp in the trace tr, is included in every rule except NIL. It ensures that timestamps increase monotonically within each trace and thereby guarantees that the partial order on events induced by their timestamps is consistent with the order of events in the trace. However, events can happen at the same time.

Intruder Rule. The FAKE-rule in Figure 5.4 captures the intruders' behavior. An intruder can send any message m deducible from his knowledge. We are not required to model the internal state of the intruders since they behave arbitrarily. We use explicit Send-events and Recv-events to model the exchange of information between colluding intruders. With an appropriate $\mathit{cdist}_{\mathit{Net}}$ function, it is possible to model an environment where the intruders are connected by high-speed links, allowing them to carry out wormhole attacks. Restrictions on the degree of cooperation between intruders can be modeled as predicates on traces. Internal and external attackers are both captured since they differ only in their initial knowledge or associated transceivers.

Protocols. In contrast to intruders who can send *arbitrary* deducible messages, honest agents follow the protocol. A protocol is defined by a set of protocol step functions. Each step function takes the local view and time of an agent as input and returns all possible actions consistent with the protocol specification.

There are two types of possible actions. An agent can either send a message using a transmitter with a given id and store the associated protocol data or make a claim.

$$\textbf{datatype}\ \mathit{action} = \mathsf{SendA}\ \mathbb{N}\ (\mathit{msg\,list}) \mid \mathsf{ClaimA}$$

An *action* associated with an agent A and a message m can be translated into the corresponding event with the *translateEv* function.

$$\begin{aligned} \mathit{translateEv}(A, \mathsf{SendA}\ k\ L, m) &= \mathsf{Send}\ \mathit{Tx}_A^k\ m\ L \\ \mathit{translateEv}(A, \mathsf{ClaimA}, m) &= \mathsf{Claim}\ A\ m \end{aligned}$$

Note that there is no Recv-action since message reception by honest agents is already modeled by the NET-rule. A protocol *step* function is therefore of type $agent \times trace \times \mathbb{R} \rightarrow (action \times msg)\ set$. Since the actions of an agent A only depend on his own previous actions and observations, we define A's view of a trace tr as the projection of tr on those events involving A.

$$view(A, tr) = [(ctime(A, t), ev) | (t, ev) \leftarrow tr \wedge occursAt(\mathrm{ev}) = A]$$

The *view* function uses the function *occursAt*, which maps events to associated agents. For instance, $occursAt(\mathsf{Send}\ Tx_A^i\ m\ L) = A$. Since the timestamps of trace events refer to absolute time, the *view* function accounts for the offset of A's clock by translating times using the *ctime* function. Given an agent and a global timestamp, the uninterpreted function $ctime$: $agent \times \mathbb{R} \rightarrow \mathbb{R}$ returns the corresponding local timestamp for the agent's clock. The clock offset of A at global time t is given by $ctime(A, t) - t$. Using the above definitions, we define the PROTO-rule in in Figure 5.4. For a given protocol, specified as a set of the step functions, the PROTO-rule describes all possible actions of honest agents, given their local views of a valid trace tr at a given time t. If all premises are met, the PROTO-rule appends the translated event to the trace. Note that agents' behavior, modeled by the function *step*, is based only on the local clocks of the agents, i.e., agents cannot access the global time. Moreover, the restriction that all messages must be in $DM_{H_A}(knows_A(tr))$ ensures that agents only send messages deducible from their knowledge.

5.1.7 Protocol-Independent Results

Since the set of traces $Tr(proto)$ is parameterized by the protocol description *proto*, our framework allows us to establish protocol-independent results that hold for all protocols or certain classes of protocols. In this section, we present four lemmas about the origin of messages that we will use later when we analyze concrete protocols. The proofs presented in this section follow the formal proofs of the corresponding lemmas as they can be found in our Isabelle/HOL formalization.

Our first lemma specifies a lower bound on the time between when an agent first uses a nonce and another agent later uses the same nonce. The lemma holds whenever the initial knowledge of all agents does not contain any nonces. Note that according to the NONCE-rule in Figure 5.2, agents can only derive nonces tagged with their own identity and all other nonces must be received over the network.

Lemma 5.2. *Let A be an arbitrary (honest or dishonest) agent and let the event $(t_S^A, \mathsf{Send}\ Tx_A^i\ m_A\ L_A) \in tr$ be the first event in the trace tr with $n \in subterms(m_A)$ for the nonce n. If there is another event $(t_S^B$, $\mathsf{Send}\ Tx_B^j\ m_B\ L_B) \in tr$ with $A \neq B$ such that $n \in subterms(m_B)$, then $t_S^B - t_S^A \geq cdist_{LoS}(A, B)$.*

Proof. We prove this by induction on $Tr(proto)$. The statement obviously holds for NIL and NET since these rules do not add Send-events. We now consider the two remaining rules. Let $tr \in Tr(proto)$ and $(t_S^A, \mathsf{Send}\ Tx_A^i\ m_A\ L_A) \in tr$ be the first event that contains the nonce n as a subterm.

FAKE. The event $(t_S^I$, $\mathsf{Send}\ Tx_I^k\ m_I\ [\ ])$ is appended to tr. The only interesting cases occur when $A \neq I$ and $n \in subterms(m_I)$. From the premises of the rule, we know that $m_I \in DM_I(knows_I(tr))$. Since I cannot guess a nonce created by another agent A, the intruder I must have received a message m containing n at some time t_R^I with receiver

Rx^h_I, where $t^I_R \le t^I_S$. Every Recv-event in *tr* is preceded by a corresponding Send-event. So there must be $\left(t^C_S, \mathsf{Send}\ Tx^u_C\ m\ L_C\right) \in tr$ such that $t^I_R - t^C_S \ge cdist_{Net}(Tx^u_C, Rx^h_I)$. If $C = A$, then $t^C_S \ge t^A_S$ since n is first used at time t^S_A and we have $t^I_S - t^A_S \ge cdist_{LoS}(A, I)$ since faster-than-light communication is impossible and $cdist_{Net}(Tx^i_A, Rx^h_I) \ge cdist_{LoS}(A, I)$. If $C \ne A$, we can apply the induction hypothesis to the event $\left(t^C_S, \mathsf{Send}\ Tx^u_C\ m\ L_C\right)$ and obtain $t^C_S - t^A_S \ge cdist_{LoS}(A, C)$. Together with $t^I_R - t^C_S \ge cdist_{LoS}(C, I)$ and $t^I_R \le t^I_S$, we obtain $t^I_S - t^A_S \ge cdist_{LoS}(A, I)$ using the triangle inequality for physical distances.

Proto. The event $(t^B_S, translateEv(B, act, m_B))$ is appended to *tr*. Only the case where $translateEv(B,\ act,\ m_B) = \mathsf{SendA}\ i\ L_B$, $B \ne A$, and $n \in subterms(m_B)$ is interesting. From the premises of the Proto-rule, we have $m_B \in DM_B(knows_B(tr))$ like in the Fake-case. We can therefore proceed analogously. □

The next lemma is similar to Lemma 5.2 and concerns the earliest time when an agent can *receive* a nonce.

Lemma 5.3. *Let A be an arbitrary (honest or dishonest) agent and let $(t^A_S, \mathsf{Send}\ Tx^i_A\ m_A\ L_A) \in tr$ be the first event in the trace tr with $n \in subterms(m_A)$ for the nonce n. If there is another event $(t^B_R,\ \mathsf{Recv}\ Rx^j_B\ m_B) \in tr$ with $A \ne B$ such that $n \in subterms(m_B)$, then $t^B_R - t^A_S \ge cdist_{LoS}(A, B)$.*

Proof. We prove this lemma by induction over $Tr(proto)$. Except for the Net-case, the statement trivially follows from the induction hypothesis since no Recv-events are added. For Net, assume that $(t^A_S, \mathsf{Send}\ Tx^i_A\ m_A\ L_A) \in tr$ is the first event that contains the nonce n and *tr* is extended with $(t^B_R, \mathsf{Recv}\ Rx^j_B\ m_B) \in tr$ such that $A \ne B$ and $n \in subterms(m_B)$. Then there must be an earlier Send-event $(t^C_S,\ \mathsf{Send}\ Tx^k_C\ m_B\ L_C)$ such that $t^B_R - t^C_S \ge cdist_{Net}(Tx^k_C, Rx^j_B)$. From Lemma 5.2, we know that $t^C_S - t^A_S \ge cdist_{LoS}(C, A)$. Since faster-than-light communication is impossible, we obtain $cdist_{LoS}(C,\ B) \le cdist_{Net}(Tx^k_C,\ Rx^j_B)$. Combining the inequalities, we obtain $t^B_R - t^C_S + t^C_S - t^A_S \ge cdist_{LoS}(B, C) + cdist_{LoS}(A, C)$. We can now apply the triangle inequality for physical distances to obtain the inequality $t^B_R - t^A_S \ge cdist_{LoS}(A, B)$. □

The next lemma concerns signatures and their creation time.

Lemma 5.4. *Let A be an honest agent and let $(t^B_S,\ \mathsf{Send}\ Tx^i_B\ m_B\ L_B) \in tr$ such that $sig(m, sk_A) \in subterms(m_B)$, where sk_A denotes the signing key of A and m is an arbitrary message. Then there is $(t^A_S, \mathsf{Send}\ Tx^j_A\ m_A\ L_A) \in tr$ such that $sig(m, sk_A) \in subterms(m_A)$ and $t^B_S - t^A_S \ge cdist_{LoS}(A, B)$.*

This lemma only holds if the initial knowledge of every agent does not contain signing keys of other agents or signatures created by using the signing keys of other agents. Additionally we must assume that protocol messages never contain signing keys of agents as extractable subterms. We formalize such assumptions as predicates on protocols and the initial knowledge.

Proof. (Sketch) The proof is analogous to the proof of Lemma 5.2, but additionally uses the fact that agents other than A cannot sign messages on behalf of A if the signing key never leaks. □

A similar lemma also holds for MACs.

Lemma 5.5. *Let A and B be honest agents and C a different, possibly dishonest, agent. Furthermore, let $(t_S^C, \mathsf{Send}\ Tx_C^i\ m_C\ L_C) \in tr$ such that $mac_{k_{AB}}(m) \in subterms(m_C)$, where k_{AB} denotes the shared key of A and B and m is an arbitrary message. Then there is $(t_S^E,\ \mathsf{Send}\ Tx_E^j\ m_E\ L_E) \in tr$ such that $E \in \{A,\ B\}$, $mac_{k_{AB}}(m) \in subterms(m_E)$ and $t_S^C - t_S^E \geq cdist_{LoS}(E, C)$.*

5.2 Case Studies

In this section, we use our model to analyze the security properties of three protocols: an authenticated ranging protocol, an ultrasonic distance-bounding protocol, and a secure time synchronization protocol. Each protocol uses cryptographic primitives as well as physical characteristics of the communication technology, environment, or network topology, to provide security guarantees. Since the first two protocols estimate distance based on round-trip measurements and bounds on the propagation speed of signals, variable clock offsets can trivially lead to wrong results. Therefore, we only consider those *ctime* functions that model a constant clock error. In the third example, we also restrict ourselves to constant clock errors.

5.2.1 Authenticated Ranging

To define the set of traces for the authenticated ranging protocol from Figure 5.5 explained in Section 2.1.4.2, we formalize the set of protocol steps $protoAR = \{ar_1,\ ar_2,\ ar_3\}$. Each step function $ar_i(tr, A, t)$ yields the possible actions of the agent A executing the protocol step i for his view of the trace tr at time t. We have formalized the steps in Isabelle/HOL using set comprehension, but present the steps here using rule notation for readability. For a fixed tr, A, and t, $ar_i(tr, A, t)$ is defined as the least set closed under the corresponding rule.

1. An honest verifier V can start a protocol run by sending a fresh nonce as a challenge. We use the index r to denote radio transmitters and receivers of honest agents.

$$\frac{(\mathsf{Nonce}\ V\, i) \notin used(tr)}{(\mathsf{SendA}\ r\ [\,], \mathsf{Nonce}\ V\, i) \in ar_1(tr, V, t_S^V)}$$

2. An honest prover P that receives a challenge may send the corresponding response. Note that we use a typed receive for the nonce, but we do not restrict the creator V' of the nonce in any way

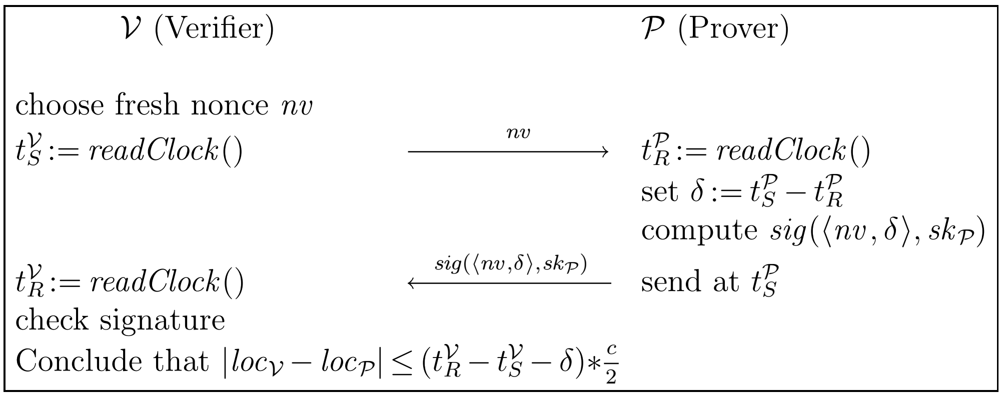
$$\frac{(t_R^P, \mathsf{Recv}\ Rx_P^r\ (\mathsf{Nonce}\ V'\ i)) \in tr}{(\mathsf{SendA}\ r\ [\,], sig(\langle \mathsf{Nonce}\ V'\ i, t_S^P - t_R^P\rangle, sk_P)) \in ar_2(tr, P, t_S^P)}$$

3. The last step introduces a Claim-event. It models the conclusion of a verifier V who received a response to his initial challenge.

$$\frac{\begin{array}{c}(t_S^V, \mathsf{Send}\ Tx_V^r\ (\mathsf{Nonce}\ V\, i)\ [\,]) \in tr \\ (t_R^V, \mathsf{Recv}\ Rx_V^r\ sig(\langle \mathsf{Nonce}\ V\, i, \delta\rangle, sk_P)) \in tr\end{array}}{\left(\mathsf{ClaimA}, \left\langle P, (t_R^V - t_S^V - \delta) * \frac{c}{2}\right\rangle\right) \in ar_3(tr, V, t_C^V)}$$

   The premises state that V has initiated a protocol run and received a response from P. V therefore believes that $(t_R^V - t_S^V - \delta) * \frac{c}{2}$ is an upper bound on his distance to P.

For this protocol, we define the initial knowledge of each agent A to be her own signing key sk_A and the public keys pk_B of all other agents B.


$\mathcal{V}$ (Verifier) | $\mathcal{P}$ (Prover)

choose fresh nonce nv

$t^{\mathcal{V}}_S := readClock()$ $\xrightarrow{nv}$ $t^{\mathcal{P}}_R := readClock()$

set $\delta := t^{\mathcal{P}}_S - t^{\mathcal{P}}_R$

compute $sig(\langle nv, \delta\rangle, sk_{\mathcal{P}})$

$t^{\mathcal{V}}_R := readClock()$ $\xleftarrow{sig(\langle nv,\delta\rangle, sk_{\mathcal{P}})}$ send at $t^{\mathcal{P}}_S$

check signature

Conclude that $|loc_{\mathcal{V}} - loc_{\mathcal{P}}| \le (t^{\mathcal{V}}_R - t^{\mathcal{V}}_S - \delta) * \frac{c}{2}$

Figure 5.5. Authenticated Ranging Protocol.

Security Analysis. As explained in Section 2.1.4.2, the protocol should compute a reliable upper on the physical distance between prover and verifier whenever the intended prover is honest. We therefore state the following theorem.

Theorem 5.6. *Let V and P be honest agents, $tr \in Tr(protoAR)$ and $(t, \mathsf{Claim}\ V\ \langle P, d\rangle) \in tr$, then $d \ge |loc_V - loc_P|$.*

The theorem states that whenever the authenticated ranging protocol successfully terminates for an honest prover and verifier, then the computed distance is an upper bound on the physical distance between the two. Remember that a dishonest prover can shorten the computed distance by lying about δ and considering a dishonest verifier does not make sense since the prover does not obtain any guarantees.

For our proof, we use three of the protocol-independent lemmas about message origination and the fact that the δ sent in the second protocol message is always equal to $t^P_S - t^P_R$, the delay between the Recv-event and the Send-event, provided that P is honest. This follows directly from the definition of ar_2.

Proof. Since only the step ar_3 adds events of the form $(t^V_C, \mathsf{Claim}\ V\ \langle P, d\rangle)$, we do not have to consider the other protocol steps and rules defining $Tr(protoAR)$. From the premises of ar_3 and the definitions of ar_2 and ar_3, we know $(t^V_S, \mathsf{Send}\ Tx^r_V\ (\mathsf{Nonce}\ V\, i)\ [\,])$ is the first event where this nonce is used in tr, $(t^V_R, \mathsf{Recv}\ Rx^r_V\ sig(\langle \mathsf{Nonce}\ V\, i, \delta\rangle, sk_P)) \in tr$, and $d = (t^V_R - t^V_S - \delta) * \frac{c}{2}$. Even though we use global timestamps t^V_R and t^V_S in contrast to ar_3 where local times for V are used, the equation for d still holds since we assume constant clock offsets.

We know there must be a Send-event $(t^C_S, \mathsf{Send}\ Tx^j_C\ sig(\langle \mathsf{Nonce}\ V\, i, \delta\rangle, sk_P))$ by some (possibly dishonest) agent C with $t^V_R - t^C_S \ge cdist_{LoS}(C, V)$. We can now apply Lemma 5.4, to obtain a Send-event $(t^P_S, \mathsf{Send}\ Tx^r_P\ sig(\langle \mathsf{Nonce}\ V\, i, \delta\rangle, sk_P))$ where P sends the signature with $t^C_S - t^P_S \ge cdist_{LoS}(P, C)$. From the definition of ar_2, we know that there must a Recv-event $(t^P_R, \mathsf{Recv}\ Rx^r_P\ \mathsf{Nonce}\ V\, i)$ such that $\delta = t^P_S - t^P_R$. For this event, we can apply Lemma 5.3 to obtain $t^P_R - t^V_S \ge cdist_{LoS}(V, P)$. Combining the (in)equalities and using the triangle inequality, we obtain the following.

$$\begin{aligned} t^V_R - t^V_S - \delta &= t^V_R - t^V_S - t^P_S + t^P_R \\ &= t^V_R - t^C_S + t^C_S - t^P_S + t^P_R - t^V_S \\ &\ge cdist_{LoS}(C, V) + cdist_{LoS}(P, C) + cdist_{LoS}(V, P) \end{aligned}$$

$$\begin{aligned} &= cdist_{LoS}(V, P) + cdist_{LoS}(P, C) + cdist_{LoS}(C, V) \\ &\geq cdist_{LoS}(V, P) + cdist_{LoS}(P, V) \\ &= |loc_V - loc_P| * \frac{2}{c} \end{aligned}$$

Hence, we conclude $d = (t_R^V - t_S^V - \delta) * \frac{c}{2} \geq |loc_V - loc_P|$. □

The specification and verification of this protocol was relatively straightforward since the protocol-independent lemmas could be directly used to obtain the desired inequalities on the times.

5.2.2 Ultrasound Distance Bounding

In our second example, we consider a protocol for *distance bounding* that uses radio signals as well as ultrasound signals to exchange messages between the communicating parties. The goal of the protocol presented in Figure 5.6 is for the verifier $\mathcal{V}$ to determine an upper bound on the distance to a possibly dishonest prover $\mathcal{P}$. $\mathcal{V}$ sends an unpredictable challenge using radio signals and waits for the corresponding response on his ultrasound receiver. Then he measures the round-trip time and computes an upper bound $(t_R^{\mathcal{V}} - t_S^{\mathcal{V}}) * s$ on the distance, where s denotes the speed of sound. Using ultrasound, which is several orders of magnitude slower than radio, she can safely neglect the transmission time of the first message and the time required for signing the response. Furthermore, by using ultrasound, the protocol can be implemented on off-the-shelf devices because time measurements with nanosecond precision are not required. This type of protocol has been proposed in [156] to enable $\mathcal{V}$ to verify a location claim of $\mathcal{P}$.

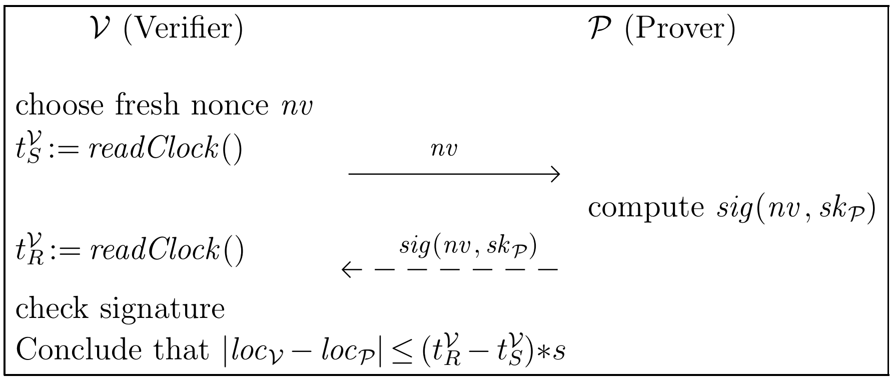

$\mathcal{V}$ (Verifier)
$\mathcal{P}$ (Prover)
choose fresh nonce nv
$t_S^{\mathcal{V}} := readClock()$
nv
compute $sig(nv, sk_{\mathcal{P}})$
$t_R^{\mathcal{V}} := readClock()$
$sig(nv, sk_{\mathcal{P}})$
check signature
Conclude that $|loc_{\mathcal{V}} - loc_{\mathcal{P}}| \leq (t_R^{\mathcal{V}} - t_S^{\mathcal{V}}) * s$

Figure 5.6. Ultrasound distance bounding protocol. Dashed lines denote ultrasound.

We assume that all agents A are equipped with ultrasound transmitters Tx_A^{us} and receivers Rx_A^{us}. Additionally, every agent has a radio transmitter and receiver, Tx_A^r and Rx_A^r. If an ultrasound receiver Rx_A^{us} is able to receive messages from a transmitter Tx_A^i, then the communication distance should reflect that the message cannot be transmitted faster than s. We therefore add the following properties of $cdist_{Net}$ as local assumptions for the security proof.

$$cdist_{Net}(Tx_A^i, Rx_B^{us}) \neq \bot \Rightarrow cdist_{Net}(Tx_A^i, Rx_B^{us}) \geq \frac{|loc_A - loc_B|}{s}$$

The same applies to messages transmitted by ultrasound transmitters Tx_A^{us} and received by receivers Rx_B^i.

$$cdist_{Net}(Tx_A^{us}, Rx_B^i) \neq \perp \Rightarrow cdist_{Net}(Tx_A^{us}, Rx_B^i) \geq \frac{|loc_A - loc_B|}{s}$$

Note that it has recently been shown in [152] that radio signals may induce a current in audio receiver circuits. More precisely, the authors of that paper demonstrated that this technique allows to successfully trigger receive events on ultrasound receivers using radio signals. This would enable trivial attacks against the protocol under consideration. The fact that we need this explicit additional assumption in our model to successfully prove the security of the protocol shows that our model accounts for subtle problems of this kind. However, we shall assume from now on that the countermeasures described in [152] have been implemented and we can keep the assumption.

We now give the set of step functions $protoUDB = \{udb_1, udb_2, udb_3\}$ that defines the ultrasound distance bounding protocol.

1. In the start step udb_1, the verifier initiates a protocol run.

$$\frac{(\mathsf{Nonce}\ V\,i) \notin used(tr)}{(\mathsf{SendA}\ r\ [\,], \mathsf{Nonce}\ V\,i) \in udb_1(V, tr, t_S^V)}$$

2. The reply step udb_2 formalizes the behavior of a prover that responds to an initial challenge. Note that the ultrasound transmitter Tx_B^{us} is used for the response.

$$\frac{(t_R^P, \mathsf{Recv}\ Rx_P^r\ (\mathsf{Nonce}\ V'\ i)) \in tr}{(\mathsf{SendA}\ us\ [\,], sig(\mathsf{Nonce}\ V'\ i, sk_P)) \in udb_2(P, tr, t_S^P)}$$

3. The final step udb_3 introduces a Claim-event when a verifier V receives a response to his initial challenge on his ultrasound receiver Rx_A^{us}.

$$\frac{\begin{array}{c}(t_S^V, \mathsf{Send}\ Tx_V^r\ (\mathsf{Nonce}\ V\,i)) \in tr \\ (t_R^V, \mathsf{Recv}\ Rx_V^{us}\ sig(\mathsf{Nonce}\ V\,i, sk_P)) \in tr\end{array}}{(\mathsf{ClaimA}, \langle P, (t_R^V - t_S^V)*s\rangle) \in udb_3(V, tr, t_C^V)}$$

This models what V concludes from a signal that apparently traveled from P to V using the speed of sound s in fewer than $t_R^V - t_S^V$ time units. Namely, V concludes that $(t_R^V - t_S^V) *s$ is a reliable upper bound on the distance to P.

Security Analysis. The desired security property of the distance bounding protocol is similar to the property of the authenticated ranging protocol proved in Theorem 5.6. Since the prover's computation time is not used for computing the distance, the protocol does not require the *prover* to be honest. We would therefore expect a statement like the following to hold.

Proposition 5.7. *Let V be an honest agent and P an arbitrary agent. Furthermore, consider a valid trace $tr \in Tr(protoUDB)$, where $(t, \mathsf{Claim}\ V\ \langle P, d\rangle) \in tr$. Then it holds that $d \geq |loc_V - loc_P|$, i.e., the distance computed by the verifier is an upper bound on the physical distance between the involved agents.*

However, as shown in [160], this proposition is false without any further assumptions. An attack involving two colluding intruders is shown in Figure 5.7. We use $PS(V)$ to denote the private space of V, which is defined as the largest circle centered at V such that V can ensure that no intruder is inside. To mount the attack, I_D is placed close to P and receives P's reply over ultrasound. I_D then uses a fast radio link to forward it to the second intruder I_C who is close to V. I_C finally delivers the message to V using ultrasound. We have proven in our Isabelle/HOL formalization that this attack is captured in our model by showing that the corresponding attack trace is a valid trace. The inequality involving the communication distances necessary for such an attack to work is

$$\begin{aligned}\frac{|loc_V - loc_P|}{s} > {} & cdist_{Net}(Tx_V^r, Rx_P^r) + cdist_{Net}(Tx_P^{us}, Rx_{I_D}^{us}) + \\ & cdist_{Net}(Tx_{I_D}^r, Rx_{I_C}^r) + cdist_{Net}(Tx_{I_C}^{us}, Rx_V^{us})\,.\end{aligned}$$

If the inequality holds, intruders connected by a fast radio link can speed up ultrasound communication between V and P using their radio link, such that the computed distance is smaller than the real distance between V and P. This attack has been discovered and implemented in [160].

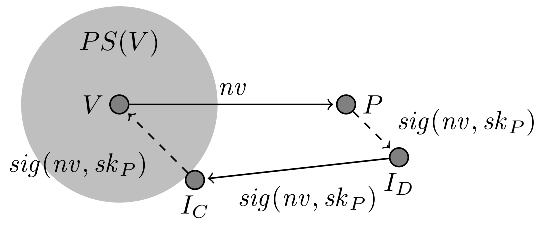


Figure 5.7. Attack on the ultrasound distance bounding protocol.

In light of the above, we prove Proposition 5.7 under an additional assumption: The verifier V can ensure that the prover P is in his private space. The same assumption is used in other protocols, e.g., [156] and [39] for location-based access control and device pairing. It holds, for example, in environments where V can visually verify the absence of nearby intruders. The following inequality ensures that no intruder is closer to the verifier V than the possibly dishonest prover P:

$$\forall D.\, |loc_V - loc_{I_D}| \geq |loc_V - loc_P|.$$

Note that this assumption thwarts Mafia frauds as well as Terrorist frauds (as defined in [34]). Remember that a mafia fraud is an attack where an intruder plays man-in-the-middle between a verifier and an honest prover. Since there is no intruder closer to the prover V than the verifier P, this kind of attack is impossible in our setting. Similarly, a terrorist fraud is an attack where an attacker plays man-in-the-middle between a verifier and a dishonest prover. This would require a second attacker being located closer to the verifier than the dishonest prover P. This setup also trivially violates the private space assumption of the verifier V. Note that Proposition 5.7 also allows for another attack where a dishonest prover I_C hands over all key material to another dishonest prover I_D that is closer to V. Then I_C can execute the protocol claiming to be I_D which results in an invalid upper bound. We will present an adapted definition that accounts for this problem in Section 5.4.

We now restate Proposition 5.7, adding the additional private space assumption, and prove the result.

Theorem 5.8. *Let V be an honest agent and P an arbitrary agent such that $\forall D.\, |loc_V - loc_{I_D}| \geq |loc_V - loc_P|$. Furthermore, consider a valid trace $tr \in Tr(protoUDB)$, where $(t, \mathsf{Claim}\ V\ \langle P, d\rangle) \in tr$. Then $d \geq |loc_V - loc_P|$.*

Proof. We prove this by induction over $Tr(protoUDB)$. Since only the step udb_3 adds events of the form $(t_C^V, \mathsf{Claim}\ V\ \langle P, d\rangle)$, we do not have to consider the other protocol steps and rules defining $Tr(protoUDB)$. From the premises of udb_3 and the definitions of udb_2 and udb_3, we conclude that $(t_S^V, \mathsf{Send}\ Tx_V^r\ (\mathsf{Nonce}\ V\,i)\ [])$ is the first event where this nonce is used in tr, $(t_R^V, \mathsf{Recv}\ Rx_V^{us}\ sig(\mathsf{Nonce}\ V\,i, sk_P)) \in tr$, and $d = (t_R^V - t_S^V) * s$. Even though we use global timestamps t_R^V and t_S^V in contrast to udb_3 where local times for V are used, the equation for d still holds since we assume constant clock offsets.

We know there must be a Send-event $(t_S^C, \mathsf{Send}\ Tx_C^j\ sig(\mathsf{Nonce}\ A\,i, sk_B))$ by some possibly dishonest agent C with $t_R^V - t_S^C \geq cdist_{Net}(Tx_C^i, Rx_V^{us}) \geq |loc_V - loc_C|/s$. We also know that $t_S^C \geq t_S^V$ by Lemma 5.2. We therefore have $d = (t_R^V - t_S^V)*s \geq (t_R^V - t_S^C)*s \geq |loc_A - loc_C|$. If $C = P$, then this concludes the proof. If $C \neq P$, then C must be dishonest and we are also done because $|loc_V - loc_C| \geq |loc_V - loc_P|$ since P is assumed to be located in $PS(V)$. $\square$

Note that our proof does not use the fact that the second protocol message is authenticated by P. Correctness is guaranteed because V ensures that P is in his private space. Therefore even a simplified version of the protocol, where the second message is replaced with $\mathsf{Hash}\ \langle \mathsf{Nonce}\ V\,i, P\rangle$, would be secure under the private-space assumption.

5.2.3 Secure Time Synchronization

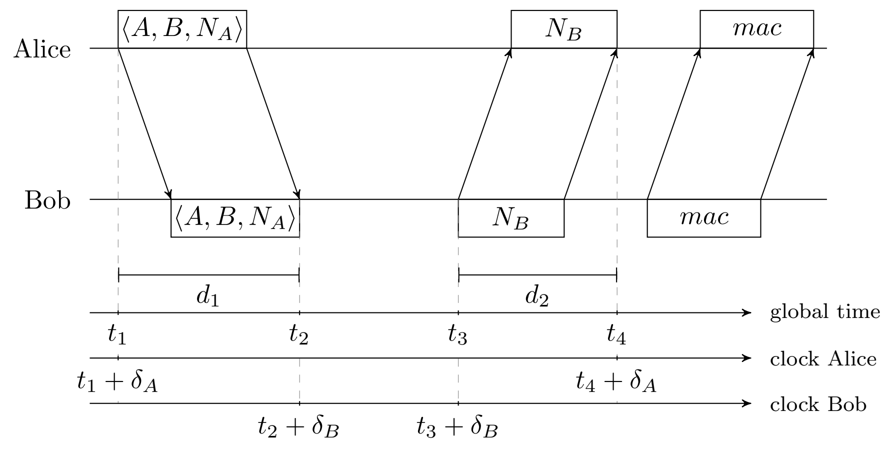
As a final example, we formalize a secure time synchronization protocol presented in [84]. The Enhanced Secure Pairwise Synchronization (E-SPS) protocol achieves pairwise clock synchronization between two honest nodes in the presence of external attackers by computing the relative clock offset between the two nodes. We analyze this protocol under the following assumptions.

Constant clock offset. We assume constant clock offsets for each agent's clock during the execution of the protocol. Namely, for each agent A there is a offset δ_A such that $ctime(A, t) = t + \delta_A$.

Lower bound on message transmission time. We assume a maximal bandwidth for the connecting network. As a consequence, there is a lower bound d_{min} on the message transmission time for any message that contains a nonce. Even in the case of a malicious sender, it is impossible for honest agents to complete reception of a nonce before $t_{start} + d_{min}$, where t_{start} is the start time of the reception.

Upper bound on end-to-end delay. Finally, we assume that there is a maximal end-to-end delay when receiving a message expected by the protocol. The maximal delay d_{max} consists of (1) the time for media access, (2) the time-of-flight, and (3) the message transmission time. In terms of media access, we assume that the hardware is capable of obtaining and inserting timestamps of reception and send events. Therefore we do not have to account for (1) since it does not affect the measurements. We obtain a bound for (2) by assuming a maximal distance between nodes. However, in most cases (2) is negligible compared to (3). For (3), it is possible to define a minimal transmission rate and to use the corresponding maximal delay. A detailed breakdown of the times can be found in [84].

Figure 5.8 depicts the E-SPS protocol as an Alice-and-Bob style sequence diagram, where time passes from left to right. The protocol securely computes an approximation δ_{AB} of the relative clock offset $\delta_A - \delta_B$. To prove that $d_{max} - d_{min}$ is upper bound of the approximation, we formalize the protocol as the set of step functions *protoESPS* $=\{esps_1, esps_2, esps_3, esps_4\}$. Recall that step functions involve only local times of agents, i.e., timestamps associated with trace events are translated to the local time of the corresponding agent.



If $d \le d_{max}$, Alice accepts the last message and concludes that δ_{AB} is an approximation of the relative clock offset such that $|\delta_{AB} - (\delta_A - \delta_B)| \le d_{max} - d_{min}$, where

$$\begin{aligned} t_i^C &= t_i + \delta_C \\ mac &= mac_{k_{AB}}(B, A, N_A, N_B, t_2^B, t_3^B) \\ d &= \frac{((t_2^B - t_1^A) + (t_4^A - t_3^B))}{2} \\ \delta_{AB} &= \frac{((t_2^B - t_1^A) - (t_4^A - t_3^B))}{2}\,. \end{aligned}$$

Figure 5.8. Enhanced secure time synchronization (E-SPS) protocol.

1. The first step $esps_1$ models an initiator A sending a challenge. Here t_1^A denotes the local time measured by A, corresponding to the global time $t_1^A - \delta_A$.

$$\frac{(\mathsf{Nonce}\ A\ i) \notin used(tr)}{(\mathsf{SendA}\ r\ [\,], \langle A, B, \mathsf{Nonce}\ A\ i\rangle) \in esps_1(A, tr, t_1^A)}$$

2. The second step $esps_2$ models an agent B responding to an initial challenge.

$$\frac{\begin{array}{c}(\mathsf{Nonce}\ B\ k) \notin used(tr) \\ t_3^B \geq t_2^B \quad t_2^B \geq t_{2,start}^B + d_{min} \\ (t_{2,start}^B, \mathsf{Recv}\ B\ \langle A, B, \mathsf{Nonce}\ A'\ i' \rangle)\end{array}}{(\mathsf{SendA}\ r\ [A, \mathsf{Nonce}\ A'\ i', t_2^B], \mathsf{Nonce}\ B\ k) \in esps_2(B, tr, t_3^B)}$$

Reception of the challenge starts at $t_{2,start}^B$ and ends at t_2^B, which is least d_{min} time units later. B associates the protocol data $[A$,$\mathsf{Nonce}\ A'\ i'$,$t_2^B]$ with the resulting Send-event.

3. The third step $esps_3$ models B sending the MAC.

$$\frac{(t_3^B, \mathsf{Send}\ Tx_B^r\ (\mathsf{Nonce}\ B\ k)\ [A, \mathsf{Nonce}\ A'\ i', t_2^B]) \in tr}{(\mathsf{SendA}\ r\ [\,], mac_{k_{AB}}(B, A, \mathsf{Nonce}\ A'\ i', \mathsf{Nonce}\ B\ k, t_2^B, t_3^B)) \in esps_3(B, tr, t^B)}$$

Besides the exchanged nonces, B includes the times t_2^B and t_3^B, which denote when the reception of A's challenge ended and when B started sending his response.

4. Finally, step $esps_4$ models A's reception of the responses.

$$\frac{\begin{array}{c}(t_1^A, \mathsf{Send}\ Tx_A^r\ \langle A, B, \mathsf{Nonce}\ A\ i \rangle\ [\,]) \in tr \\ (t_{4,start}^A, \mathsf{Recv}\ Rx_A^r\ (\mathsf{Nonce}\ B'\ j')) \in tr \\ (t_5^A, \mathsf{Recv}\ Rx_A^r\ mac_{k_{AB}}(B, A, \mathsf{Nonce}\ A\ i, \mathsf{Nonce}\ B'\ j', t_2^B, t_3^B)) \in tr \\ t_4^A \geq t_{4,start}^A + d_{min} \quad d \leq d_{max} \quad t^A \geq t_4^A \\ d = ((t_2^B - t_1^A) + (t_4^A - t_3^B)) * \frac{1}{2} \\ \delta_{AB} = ((t_2^B - t_1^A) - (t_4^A - t_3^B)) * \frac{1}{2}\end{array}}{(\mathsf{ClaimA}, \langle B, \delta_{AB} \rangle) \in esps_4(A, tr, t^A)}$$

If A concludes a clock-offset δ_{AB}, A must have received the first response and the corresponding MAC according to the protocol specification. A then computes the delay d and the offset δ_{AB} using his own time measurements and the timestamps received in the messages from B (see Figure 5.8). Finally, A completes the protocol only if $d \leq d_{max}$. Otherwise, the end-to-end delay for the relevant transmissions took too long, which would result in an unreliable estimation of the clock offset.

First note that if neither the challenge nor the response is delayed by an intruder or by the environment, then $d_1 \approx d_2$ for the two transmission delays (see Figure 5.8) and

$$d = \frac{((d_1 + \delta_B - \delta_A) + (d_2 + \delta_A - \delta_B))}{2} = \frac{d_1 + d_2}{2} \leq d_{max}\,.$$

In this case, Alice computes the relative clock offset

$$\delta_{AB} = \frac{((d_1 - d_2) + 2*\delta_B - 2*\delta_A)}{2} \approx \delta_B - \delta_A\,.$$

The upper bound d_{max} bounds the error that the adversary can introduce by delaying either the challenge or the response. We first show that if there is a Claim-event, then there are four corresponding times that satisfy the following (in)equalities.

Lemma 5.9. *Let A and B be different honest agents, let $tr \in Tr(protoESPS)$ arbitrary, and $(t, \mathsf{Claim}\ A\ \langle B, \delta_{AB}\rangle) \in tr$. Then there are times t_1, t_2, t_3, and t_4 such that the following holds:*

1. $t_2 - t_1 \geq d_{min}$
2. $t_3 \geq t_2$
3. $t_4 - t_3 \geq d_{min}$
4. $\delta_{AB} = (((t_2 - \delta_B) - (t_1 + \delta_A)) - ((t_4 + \delta_A) - (t_3 + \delta_B)))*\frac{1}{2}$
5. $((t_2 - t_1) + (t_4 - t_3))*\frac{1}{2} \leq d_{max}$

Proof. From the existence of the Claim-event in the trace, we can conclude that all the premises of $esps_4$ hold. There is a Send-event for a fresh nonce na at time t_1^A. There is a Recv-event for a nonce nb that starts at time $t_{4,start}^A$ and is completed at time t_4^A. Finally, there is a Recv-event of $mac_{k_{AB}}(B, A, na, nb, t_2^B, t_3^B)$. For the global times $t_1 = t_1^A - \delta_A$, $t_4 = t_4^A - \delta_A$, $t_{4,start} = t_{4,start}^A - \delta_A$, $t_2 = t_2^B - \delta_B$, and $t_3 = t_3^B - \delta_B$ we can conclude that 4. and 5. hold and $t_4 - t_{4,start} \geq d_{min}$.

The *mac* must originate from B by Lemma 5.5 because the key k_{AB} is assumed to be a shared secret between A and B and $esps_3$ is the only protocol step that uses the key. Then $esps_2$ must have been executed by B and the reception of the nonce na must have started at a global time $t_{2,start}$ and finished at t_2, where $t_2 - t_{2,start} \geq d_{min}$. By Lemma 5.3, we known that $t_1 \geq t_{2,start}$ and hence $t_2 - t_1 \geq d_{min}$. We also know that $t_3 \geq t_2$ from the premises of $esps_2$. To see that 2. holds, note that the nonce nb is first used at time t_3 and reception of nb starts at $t_{4,start}$. Using Lemma 5.3 again, we obtain $t_{4,start} \geq t_3$. Together with $t_4 - t_{4,start} \geq d_{min}$, this proves $t_4 - t_3 \geq d_{min}$. □

Using these inequalities, it is easy to prove the following security property already stated in Figure 5.8.

Theorem 5.10. *Let A and B be honest agents, $tr \in Tr(protoESPS)$ and $(t, \mathsf{Claim}\ A\ \langle B, \delta_{AB}\rangle) \in tr$, then $|\delta_{AB} - (\delta_B - \delta_A)| \leq d_{max} - d_{min}$.*

Proof. From Lemma 5.9, we know that there are times t_1, t_2, t_3, and t_4 such that conditions 1.–5. are satisfied. We can first use 4. to obtain the new goal

$$|((t_2 - t_1) - (t_4 - t_3))/2| \leq d_{max} - d_{min}.$$

Using 5. we conclude that $(t_2 - t_1) \leq 2*d_{max} - (t_4 - t_3)$ and $(t_4 - t_3) \leq 2*d_{max} - (t_2 - t_1)$. We can therefore use 1. and 3. to obtain $(t_2 - t_1) \leq 2*d_{max} - d_{min}$ and $(t_4 - t_3) \leq 2*d_{max} - d_{min}$. We now perform a case distinction. If $(t_2 - t_1) \geq (t_4 - t_3)$, then

$$\begin{aligned}|((t_2 - t_1) - (t_4 - t_3))/2| &= ((t_2 - t_1) - (t_4 - t_3))/2\\ &\leq ((2*d_{max} - d_{min}) - (t_4 - t_3))/2\\ &\leq d_{max} - d_{min}\end{aligned}$$

because $t_4 - t_3 \geq d_{min}$. If $(t_2 - t_1) < (t_4 - t_3)$, then

$$\begin{aligned}|((t_2 - t_1) - (t_4 - t_3))/2| &= ((t_4 - t_3) - (t_2 - t_1))/2\\ &\leq ((2*d_{max} - d_{min}) - (t_2 - t_1))/2\\ &\leq d_{max} - d_{min}\end{aligned}$$

because $t_2 - t_1 \geq d_{min}$. □

Note that this is a tighter bound on the error than we proved in [21], where the bound was $2*(d_{max} - d_{min})$. Aside from the reasoning about the creation of MACs, which is covered by our protocol independent lemmas, most of the theorem-proving work concerns establishing the desired inequalities between the times of the different events. Most of these inequalities stem from the relation between the different clock offsets of the two nodes.

5.3 Extended Model with *Xor* and Overshadowing

So far, we have analyzed an authenticated ranging protocol and an ultrasound distance bounding protocol. For both protocols, the time required by the prover to compute the response is not critical. For the authenticated ranging protocol, the prover is assumed to be honest and the computation time is subtracted to obtain the time-of-flight. For the ultrasound distance bounding protocol, the computation time is included in the time-of-flight measured by the verifier, but since the reply travels with the speed of sound, it is negligible. In this section, we analyze distance bounding protocols that exclusively use radio transmission. In this case, the computation time is included in the measured time-of-flight and can cause a significant enlargement of the computed distance. To minimize this error, the computation of the response from the challenge must be fast. For this reason, many distance bounding protocols use XOR. Another advantage of using XOR is that the challenge-response phase can be performed bitwise, i.e., the verifier sends the i-th bit of the challenge and waits for the i-th bit of the response before sending the next bit. This can be used as a countermeasure against some implementation specific attacks as described by [156, 49].

To support such protocols, we extend our framework with support for *Xor*. To capture practically relevant attacks, we also extend our model to account for (partial) overshadowing of messages that are in transmission. We do not model all the details of overshadowing in our symbolic model, but enough to discover new attacks and rule out a large class of attack for fixed version of the considered protocols.

5.3.1 Message Theory for *Xor*

We now extend the framework presented in the previous sections with support for *Xor*. To achieve this, we continue as follows. First, we define the type *fmsg* of free messages. The inhabitants of this type are elements of the free term algebra built over atomic messages and the cryptographic operators including *Xor*. Second, we define the equivalence relation $=_{Xor}$ on *fmsg* that models the equalities that hold for *Xor*. Third, we define a normal-form with respect to $=_{Xor}$ using a normalization function. Fourth, we define a new (abstract) type *msg* that is represented by the normal-form messages in *fmsg*. This type is bijective to the quotient type $fmsg/_{=_{Xor}}$. We then lift the functions for message construction and message deduction from *fmsg* to *msg*. Finally, we define the notions of *parts* and *subterms* on *msg*.

Free Messages and $=_{Xor}$. The type *fmsg* of free messages is defined as

$$\begin{aligned}\textbf{datatype}\ \mathit{fmsg} = {} & \mathsf{AGENT}\ \mathit{agent} \mid \mathsf{TIME}\ \mathbb{R} \mid \mathsf{NUMBER}\ \mathbb{N} \mid \mathsf{NONCE}\ \mathit{agent}\ \mathbb{N} \\ & \mid \mathsf{KEY}\ \mathit{key} \mid \mathsf{HASH}\ \mathit{fmsg} \mid \mathsf{PAIR}\ \mathit{fmsg}\ \mathit{fmsg} \mid \mathsf{CRYPT}\ \mathit{key}\ \mathit{fmsg} \\ & \mid \mathit{fmsg} \boxplus \mathit{fmsg} \mid \mathsf{ZERO}\,.\end{aligned}$$

We use uppercase constructors and $\boxplus$ because we want to use the usual operator names for *msg*. The equivalence relation $=_{Xor}$: $\mathit{fmsg} \times \mathit{fmsg} \to \mathit{bool}$ is defined as the equational theory generated by the following equations.

$$\begin{aligned}(x \boxplus y) \boxplus z \simeq x \boxplus (y \boxplus z) \quad & \text{(A)} \quad x \boxplus y \simeq y \boxplus x \quad \text{(C)} \\ x \boxplus \mathsf{ZERO} \simeq x \quad & \text{(U)} \quad x \boxplus x \simeq \mathsf{ZERO} \quad \text{(N)}\end{aligned}$$

Normal-form and normalization function. We define the function $(\_)\!\downarrow : \mathit{fmsg} \to \mathit{fmsg}$ that returns the normal-form of the given term. Our normal-form is defined such that neither (U) nor (N) are applicable from left to right and all applications of $\boxplus$ are nested to the right and sorted with respect to a total order on *fmsg*. To define the normalization function, we first define two helper functions. The helper function $\_ \oplus^? \_ : \mathit{fmsg} \times \mathit{fmsg} \to \mathit{fmsg}$ is defined as $a \oplus^? b = (\text{if } b = \mathsf{ZERO} \text{ then } a \text{ else } a \boxplus b)$. It applies the $\boxplus$ operator, but ensures that the message $a \boxplus \mathsf{ZERO}$, which is not in normal-form, is never created. The helper function $\_ \oplus^\downarrow \_ : \mathit{fmsg} \times \mathit{fmsg} \to \mathit{fmsg}$ is defined in Figure 5.9 and also applies $\boxplus$ to its arguments. It ensures that the resulting message is in normal-form whenever its arguments are already in normal-form.

$$\begin{aligned}x \oplus^\downarrow \mathsf{ZERO} &= x \\ \mathsf{ZERO} \oplus^\downarrow x &= x \\ (a_1 \boxplus a_2) \oplus^\downarrow (b_1 \boxplus b_2) &= \text{if } a_1 = b_1 \text{ then } a_2 \oplus^\downarrow b_2 \\ &\quad\ \text{else if } a_1 < b_1 \text{ then } a_1 \oplus^? (a_2 \oplus^\downarrow (b_1 \boxplus b_2)) \\ &\quad\ \text{else } b_1 \oplus^? ((a_1 \boxplus a_2) \oplus^\downarrow b_2) \\ (a_1 \boxplus a_2) \oplus^\downarrow b &= \text{if } a_1 = b \text{ then } a_2 \\ &\quad\ \text{else if } a_1 < b \text{ then } a_1 \oplus^? (a_2 \oplus^\downarrow b) \\ &\quad\ \text{else } b \boxplus (a_1 \boxplus a_2) \\ a \oplus^\downarrow (b_1 \boxplus b_2) &= (b_1 \boxplus b_2) \oplus^\downarrow a \\ a \oplus^\downarrow b &= \text{if } a = b \text{ then } \mathsf{ZERO} \text{ else if } a < b \text{ then } a \boxplus b \text{ else } b \boxplus a\end{aligned}$$

Figure 5.9. The $\oplus^\downarrow$ function.

Using these definitions, we can now define $\downarrow$ as follows.

$$\begin{aligned}(\mathsf{HASH}\ m)\!\downarrow &= \mathsf{HASH}\ (m\!\downarrow) \\ (\mathsf{PAIR}\ m_1\ m_2)\!\downarrow &= \mathsf{PAIR}\ (m_1\!\downarrow)\ (m_2\!\downarrow) \\ (\mathsf{CRYPT}\ k\ m)\!\downarrow &= \mathsf{CRYPT}\ k\ (m\!\downarrow) \\ (a \boxplus b)\!\downarrow &= (a\!\downarrow) \oplus^\downarrow (b\!\downarrow) \\ x\!\downarrow &= x\end{aligned}$$

We have proved that $\downarrow$ is sound and complete with respect to $=_{Xor}$ in the following sense.

Lemma 5.11. *For all $s, t \in \mathit{fmsg}$, $s =_{Xor} t$ if and only if $s\!\downarrow = t\!\downarrow$.*

Since we define the normalization function $\downarrow$ directly, we can use Isabelle's automatic termination prover and the message $m\downarrow$ is obviously unique. The direction $s\downarrow = t\downarrow$ implies $s =_{Xor} t$ of Lemma 5.11 follows from the intermediate result $s =_{Xor} s\downarrow$ and is not that hard to prove. The other direction is much harder to prove and requires several intermediate definitions and lemmas. For example, it was useful to inductively define a predicate nf that characterizes normal-form messages and show that $\mathit{nf}(a)$ if and only if $a = a\downarrow$. This allowed us to use the automatically derived induction principle for nf for all messages $a\downarrow$.

An alternative approach would be to to use ordered rewriting or rewriting modulo AC and to define $\downarrow$ as the fixpoint of the one-step rewriting relation. This would require manual confluence and termination proofs for the rewriting relation. On the other hand, the direction $s =_{Xor} t$ implies $s\downarrow = t\downarrow$ in Lemma 5.11 would follow immediately from confluence.

The *msg* Type, *DM*, *parts*, and *subterms*. We use the declaration

$$\textbf{typedef } \mathit{msg} = \{m \in \mathit{fmsg} \mid m\downarrow = m\}$$

to define the abstract type msg that is represented by the subset of normal-form messages in fmsg. The **typedef** declaration also defines two functions $\mathit{Abs}_{\mathit{msg}}\colon \mathit{fmsg} \rightarrow \mathit{msg}$ and $\mathit{Rep}_{\mathit{msg}}\colon \mathit{msg} \rightarrow \mathit{fmsg}$ and axioms that state that $\mathit{Abs}_{\mathit{msg}}$ is a bijection from the representing set $\{m \mid m\downarrow = m\}$ to msg and $\mathit{Rep}_{\mathit{msg}}$ is a bijection from msg to $\{m \mid m\downarrow = m\}$. For m with $m\downarrow \neq m$, $\mathit{Abs}_{\mathit{msg}}(m)$ is unspecified. Using these functions, we can lift the constructors from fmsg to msg as follows.

$$\begin{aligned}
\mathsf{Nonce}\ a\ i &= \mathit{Abs}_{\mathit{msg}}(\mathsf{NONCE}\ a\ i)\\
\mathsf{Pair}\ m_1\ m_2 &= \mathit{Abs}_{\mathit{msg}}(\mathsf{PAIR}\ (\mathit{Rep}_{\mathit{msg}}(m_1))\ (\mathit{Rep}_{\mathit{msg}}(m_2)))\\
\mathsf{Hash}\ m &= \mathit{Abs}_{\mathit{msg}}(\mathsf{HASH}\ (\mathit{Rep}_{\mathit{msg}}(m)))\\
\mathsf{Xor}\ a\ b &= \mathit{Abs}_{\mathit{msg}}((\mathit{Rep}_{\mathit{msg}}(a) \boxplus \mathit{Rep}_{\mathit{msg}}(b))\downarrow)\\
\mathsf{Zero} &= \mathit{Abs}_{\mathit{msg}}(\mathsf{ZERO})
\end{aligned}$$

Note that except for the definition of Xor, the argument to $\mathit{Abs}_{\mathit{msg}}$ is already in normal form since all constructors except for $\boxplus$ are free with respect to $=_{Xor}$. For Xor, we use $\downarrow$ to ensure that the argument to $\mathit{Abs}_{\mathit{msg}}$ is in normal-form because the result would be unspecified otherwise. The definitions for Agent, Int, Real, and Key are analogous to Nonce. In the following, we write $a \oplus b$ for $(\mathsf{Xor}\ a\ b)$ and 0 for Zero.

We can now define the message deduction function $\mathit{DM}\colon \mathit{agent} \rightarrow \mathit{msg}\ \mathit{set} \rightarrow \mathit{msg}\ \mathit{set}$. As can be seen in Figure 5.10, the definition is analogous to the definition for the free message algebra without Xor. Note that the XOR-rule uses the Xor constructor for the type msg of normal-form messages.

$$\begin{array}{cc}
\textsc{Inj}\ \dfrac{m \in M}{m \in \mathit{DM}_A(M)} & \ldots \\[2ex]
\ldots & \textsc{Number}\ \dfrac{}{\mathsf{Number}\ n \in \mathit{DM}_A(M)} \\[2ex]
\textsc{Xor}\ \dfrac{a \in \mathit{DM}_A(M) \quad b \in \mathit{DM}_A(M)}{a \oplus b \in \mathit{DM}_A(M)} & \textsc{Zero}\ \dfrac{}{0 \in \mathit{DM}_A(M)}
\end{array}$$

Figure 5.10. Message deduction function DM for Xor. The missing rules are equal to Figure 5.2.

To define the *subterms* and *parts* functions on *msg*, we first extend the definitions from Figure 5.3 to account for $\boxplus$ to obtain the functions *fsubterms* and *fparts* on *fmsg*. For example, $(\mathsf{NONCE}\ A\ \ i) \in \mathit{fparts}((\mathsf{NONCE}\ A\ \ i) \boxplus (\mathsf{NONCE}\ A\ \ i))$. Then, we define *parts* as

$$\mathit{parts}(m) = \{\mathit{Abs}_{\mathit{msg}}(a) \,|\, a \in \mathit{fparts}(\mathit{Rep}_{\mathit{msg}}(m))\}.$$

The function *subterms* is defined analogously. To illustrate these definitions, we consider the message $m = (\mathsf{Nonce}\ A\ \ i) \oplus (\mathsf{Nonce}\ A\ \ i)$. Unfolding the definition of $\oplus$, which is an abbreviation for Xor, we see that

$$m = \mathit{Abs}_{\mathit{msg}}((\mathit{Rep}_{\mathit{msg}}(\mathsf{Nonce}\ A\ \ i) \boxplus \mathit{Rep}_{\mathit{msg}}(\mathsf{Nonce}\ A\ \ i))\!\downarrow) = \mathit{Abs}_{\mathit{msg}}(\mathsf{ZERO}) = 0.$$

Hence, $\mathit{parts}(m) = \mathit{parts}(0) = \{0\}$. This is exactly the definition that we want because we are only interested in the *parts* of the normal form of a message. For non-*Xor* messages, we know that those must have been used in the construction of the message.

5.3.2 Network Rule with Overshadowing

Our Fake and Net rules allow the intruder to receive, delete, and send messages. Ignoring the aspects of timing and network connectivity, giving the intruder the additional capability to modify messages that are in transmission is not necessary since he can always receive and delete such a message m, deduce a new message m' using m and other known messages, and finally send the message m'. Since our model captures both timing and network connectivity, this reasoning does not apply. We therefore extend our model in this section to account for various modifications of messages in transmission.

Message Manipulations on the Wireless Channel. As shown in [147], if no cryptographic integrity protection is employed, it is possible for an adversary to perform symbol flipping and partial overshadowing of messages sent by honest agents. To perform these modifications, it is not necessary for the adversary to known the original message m. He just has to send the right modification signal at the right time. Then the receiver receives the superposition of the modification signal and the original signal that encodes the message m, which results in the reception of a message m'. We will consider two types of message modification. First, the intruder can completely overshadow the component of a pair with a known message, even if the remaining components are not known. For example, if an honest agent sends $\langle nv,\, P\rangle$, then the adversary can overshadow the P and cause the reception of $\langle nv,\, I\rangle$, even if he does not know nv. Second, the intruder can modify an unknown message m into an unknown message m' such that the Hamming-distance between m and m' is low. For example, if the identities P and I differ only in a few positions, then the adversary can use symbol flipping or overshadowing to modify $nv \oplus P$ to $nv \oplus I$, even if he does not know nv. To achieve this, he computes the positions i where P and P' differ, then he guesses nv_i for these positions, and finally overshadows these positions with $nv_i \oplus P'_i$. We will later show in Figure 5.15 how this can capability can lead to attacks.

Formalization of Message Manipulations. To formalize these message manipulation capabilities, we require two definitions. We define the *components* of a message m as

$$\mathit{components}(m) = \begin{cases} \mathit{components}(m_1) \cup \mathit{components}(m_2) & \text{if } m = \langle m_1, m_2\rangle \\ \{m\} & \text{otherwise} \end{cases}.$$

We define the set LHW of messages that might have have a low Hamming-weight as the least set closed under the rules in Figure 5.11.

$$\frac{}{\mathsf{Agent}\ A \in LHW} \qquad \frac{}{\mathsf{Number}\ n \in LHW} \qquad \frac{}{\mathsf{Time}\ t \in LHW}$$

$$\frac{m_1 \in LHW \quad m_2 \in LHW}{m_1 \oplus m_2 \in LHW} \qquad \frac{}{0 \in LHW}$$

Figure 5.11. The set LHW is the least set closed under these rules.

This definition captures that constants and the *Xor* of constants might have a low Hamming-weight. For nonces, hashes, encryptions, or the *Xor* of such messages where they do not cancel out, we assume the probability of a low Hamming-weight is negligible. Note that two messages m_1 and m_2 have a small *Hamming-distance* if $m_1 \oplus m_2 \in LHW$. Using these definitions, we define the extended network rule ExtNet in Figure 5.12.

$$\frac{tr \in Tr(proto) \qquad \begin{array}{l} \forall u \in components(m). \\ \quad \exists t_S\ A\ L\ m'\ v. \\ \qquad v \in components(m') \\ \qquad \wedge \big(t_S, \mathsf{Send}\ Tx_A^j\ m'\ L\big) \in tr \\ \qquad \wedge u \oplus v \in LHW \\ \qquad \wedge\ cdist_{Net}\big(Tx_A^j, Rx_B^i\big) \neq \bot \\ \qquad \wedge t_R - t_S \geq cdist_{Net}\big(Tx_A^j, Rx_B^i\big) \end{array}}{tr \cdot \big(t_R, \mathsf{Recv}\ Rx_B^i\ m\big) \in Tr(proto)}$$

Figure 5.12. The extended network rule that allows for partial overshadowing of messages.

The timing constraints are similar to the original Net rule, but we do not require that there is *one* corresponding Send-event for the received message m. We require that for each component u of m, there is a corresponding Send-event for a message m' with component v such that the Hamming-distance between u and v is small. This reflects that a message that contains v can be received as u if both messages are close with respect to the Hamming-distance. Note that we do not require any intruder sends for such low Hamming-distance modifications. An alternative viewpoint is that we allow for a small number of bit-errors in the transmission.

5.4 Case Study: The Brands-Chaum Protocol

In this section, we apply our framework with *Xor* and the network rule that accounts for overshadowing to the Brands-Chaum distance bounding protocol. We first formalize the protocol and define the desired security notion. Then we show that there is an attack, discuss various fixes, and prove the security of the fixed variants.

The Brands-Chaum protocol was the first proposed distance bounding protocol. Its goal is to prevent Mafia fraud and distance fraud as defined in Section 2.1.4.3. The protocol is given in Figure 5.13 and proceeds as follows. First, the prover commits to a fresh nonce np. After receiving the commitment, the verifier starts a rapid bit-exchange phase where he sends a single challenge bit nv_i of his freshly chosen nonce nv. The prover responds with $nv_i \oplus np_i$ and the verifier measures the roundtrip time. Since only a single bit is transmitted and the computation on the prover's side is cheap, delays that do not stem from the traveling times of the two signals are minimized. After the rapid bit-exchange phase is completed, the prover sends an authentication message that consists of the opening of the commitment and a signature on nv and the response $nv \oplus np$. Finally, after receiving the authentication messages, the verifier computes the upper bound on the distance as the maximal roundtrip time from the rapid bit-exchange multiplied by c divided by 2.

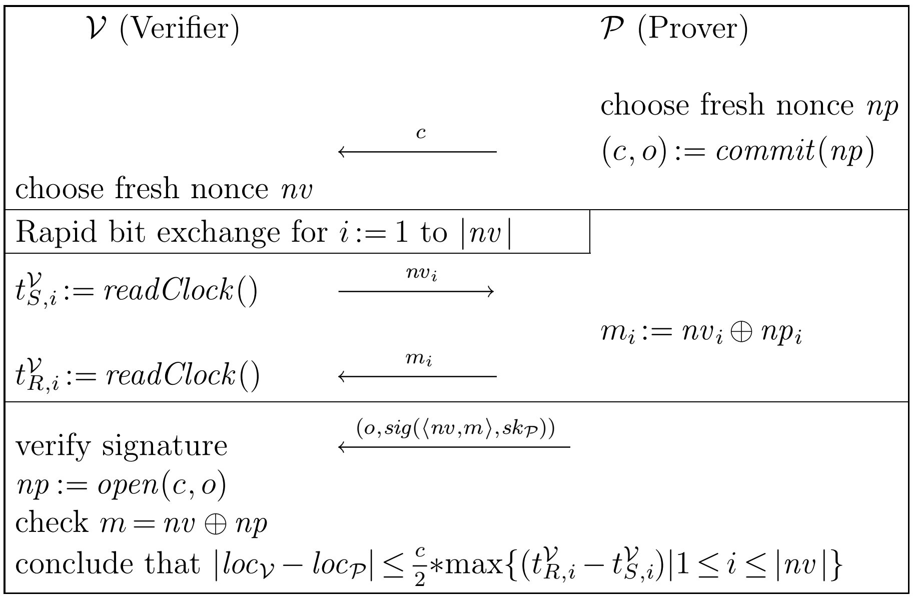

$\mathcal{V}$ (Verifier)
$\mathcal{P}$ (Prover)
choose fresh nonce np
$(c, o) := commit(np)$
c
choose fresh nonce nv
Rapid bit exchange for $i := 1$ to $|nv|$
$t^{\mathcal{V}}_{S,i} := readClock()$
nv_i
$m_i := nv_i \oplus np_i$
$t^{\mathcal{V}}_{R,i} := readClock()$
m_i
$(o, sig(\langle nv, m\rangle, sk_{\mathcal{P}}))$
verify signature
$np := open(c, o)$
check $m = nv \oplus np$
conclude that $|loc_{\mathcal{V}} - loc_{\mathcal{P}}| \leq \frac{c}{2} * \max\{(t^{\mathcal{V}}_{R,i} - t^{\mathcal{V}}_{S,i}) | 1 \leq i \leq |nv|\}$

Figure 5.13. The Brands-Chaum distance bounding protocol.

Since our non-probabilistic model cannot deal with the exchange of single random bits, we abstract away from the rapid bit-exchange and formalize this phase as the exchange of a single challenge and a single response. Note that for this protocol, the k-th bit of the challenge and response is completely independent of the earlier bits exchanged in the rapid bit-exchange phase. Hence, it seems reasonable to assume that we do not miss attacks by forcing the adversary to come up with the whole challenge or response at once if he wants to interfere with the challenge-response phase.

We formalize the protocol as the set $protoBS = \{bs_1, bs_2, bs_3, bs_4, bs_5\}$ of protocol steps.

1. The first step bs_1 formalizes the commitment by the prover. We use a hash function $h(m)$ to model the bit commitment scheme, i.e., $commit(m) = (h(m), m)$.

$$\frac{(\mathsf{Nonce}\ P\ i) \notin used(tr)}{(\mathsf{SendA}\ r\ [\mathcal{P}_1, \mathsf{Nonce}\ P\ i], \mathsf{Hash}\ (\mathsf{Nonce}\ P\ i)) \in bs_1(P, tr, t)}$$

   The step sends the hash of a fresh nonce and stores the protocol state $[\mathcal{P}_1, \mathsf{Nonce}\ P\ i]$, which denotes that the first step of the prover was executed with the given nonce.

2. The second step bs_2 formalizes the reception of the commitment and the sending of the challenge by the verifier.

$$\frac{\begin{array}{c}(t^V_R, \mathsf{Recv}\ Rx^r_V\ com) \in tr\\ (\mathsf{Nonce}\ V\ j) \notin used(tr)\end{array}}{(\mathsf{SendA}\ r\ [\mathcal{V}_1, \mathsf{Nonce}\ V\ j, com], \mathsf{Nonce}\ V\ j) \in bs_2(V, tr, t)}$$

   Similarly to the first step, the new event is associated with the protocol state denoting that the first step of the verifier was executed with the given nonce and commitment.

3. The third step bs_3 formalizes the response to the challenge by the prover.

$$\frac{\begin{array}{c}(t^P_R, \mathsf{Recv}\ Rx^r_P\ nv) \in tr\\ (t^P_S, \mathsf{Send}\ Tx^r_P\ com\ [\mathcal{P}_1, np]) \in tr\end{array}}{(\mathsf{SendA}\ r\ [\mathcal{P}_2, np, nv], nv \oplus np) \in bs_3(P, tr, t)}$$

   Note that in contrast to the earlier protocol formalizations without Xor, we use an untyped receive of the nonce nv. This captures more attacks and is also a more realistic assumption with Xor. Even if type tags are used to distinguish messages of different types, the Xor of a nonce and two identities usually has the same type tag as a nonce.
4. The fourth step bs_4 formalizes the delayed authentication of the response by the prover.

$$\frac{(t^P_S, \mathsf{Send}\ Tx^r_P\ \mathrm{resp}\ [\mathcal{P}_2, np, nv]) \in tr}{(\mathsf{SendA}\ r\ [\mathcal{P}_3, np, nv], sig(\langle nv, nv \oplus np\rangle, sk_P)) \in bs_4(P, tr, t)}$$

5. The fifth step bs_5 formalizes the claim by the receiver after receiving both the response to the challenge and the authentication message.

$$\frac{\begin{array}{c}(t^V_S, \mathsf{Send}\ Tx^r_V\ nv\ [\mathcal{V}_1, nv, com]) \in tr\\ (t^V_{\mathrm{Resp}}, \mathsf{Recv}\ Rx^r_V\ m) \in tr\\ (t^V_{\mathrm{Auth}}, \mathsf{Recv}\ Rx^r_V\ sig(\langle nv, m\rangle, sk_P)) \in tr\\ np = m \oplus \mathrm{nv} \quad com = \mathsf{Hash}\ np\end{array}}{\big(\mathsf{ClaimA}, \big\langle P, (t^V_{Resp} - t^V_S) * \frac{c}{2}\big\rangle\big) \in bs_5(V, tr, t)}$$

   Given the response m and his own nonce nv, the verifier can compute the prover's nonce np as $m \oplus nv$ and check the correctness of the commitment com.

We now define the desired security property of distance bounding protocols.

Definition 5.12. *Let proto be a distance bounding protocol, V an honest agent, and P an arbitrary agent. Furthermore, consider a valid trace $tr \in Tr(proto)$, where $(t, \mathsf{Claim}\ V\ \langle P, d\rangle) \in tr$. The protocol proto is* honest-prover-secure (hp-secure) *if P honest implies $d \geq |loc_V - loc_P|$. It is* dishonest-honest-prover-secure (dp-secure) *if P dishonest implies that there is a dishonest agent P' with $d \geq |loc_V - loc_{P'}|$.*

If P is honest, the theorem guarantees that the distance computed by the verifier is an upper bound on the physical distance between the involved agents. If P is dishonest, we obtain only the weaker guarantee that the computed distance is an upper bound on the distance between the verifier and *some* dishonest prover.

By adapting the definition in the dishonest prover case, we account for the possibility that a dishonest agent P provides another dishonest agent P' with his key material and thereby allows P' to execute the distance bounding protocol posing as P. This allows P to claim the distance between P' and V.

We do not consider a scenario where collusion between dishonest agents is limited, e.g., provers that cooperate once, but do not want to give away long-term key material or help other provers with future attacks. In such a scenario, it might be possible to achieve a stronger security property where a dishonest prover cannot claim the distance of another dishonest prover. The security of a protocol in such a setting is called resistance against Terrorist fraud. See Section 2.1.4.4 for a discussion of this issue.

5.4.1 Security Analysis

We now analyze the security of the Brands-Chaum distance bounding protocol. Note that leaving out the commitment allows for a simple distance fraud attack where a dishonest prover jumps the gun and sends some reply m before receiving nv. The nonce np can be later computed as $m \oplus nv$. Also note that if the verifier can receive a reflection of his own challenge nv as a response, there is another distance fraud attack where a dishonest prover P commits to 0 and the verifier accepts his own challenge $nv = nv \oplus 0$ as a response from P. More generally, the prover P can commit on a message l with low Hamming-weight and use message modification to cause the reception of $nv \oplus l$ at the verifier. To prevent these attacks, we assume from now on that no radio transceiver can receive its own signal, i.e., for all A, $cdist_{Net}(Tx^r_A, Rx^r_A) = \bot$.

Theorem 5.13. *The protocol protoBS is hp-secure, but not dp-secure.*

Proof. (Sketch) If there is a an event $\mathsf{Claim}\ V\ \langle P, d\rangle$ for an honest prover P, then V must have sent the challenge $\mathsf{Nonce}\ V j$ at some time t^V_S and received a response $(\mathsf{Nonce}\ V j) \oplus np$ at time t^V_R. Since V also receives a signature on the challenge and response by P, we know that $np = \mathsf{Nonce}\ P k$ for some k. From the protocol rules, we know that P must have received $\mathsf{Nonce}\ V j$ at some time t^P_R and sent the response $m = (\mathsf{Nonce}\ V j) \oplus (\mathsf{Nonce}\ P k)$ at some later time t^P_S. Using Lemma 5.3, we obtain $t^P_R - t^V_S \geq cdist_{LoS}(V, P)$ since $\mathsf{Nonce}\ V j$ is not used before t^V_S. We also know that the response m is the first message with $\mathsf{Nonce}\ P k \in parts(m)$ since the nonce is only used in the commitment earlier on, where it is protected by the hash. We have proved a lemma similar to Lemma 5.3, but with *parts* instead of *subterms*. Using this lemma, we obtain $t^V_R - t^P_S \geq cdist_{LoS}(P, V)$. Combining the two inequalities and $t^P_S \geq t^P_R$, we obtain the desired result. □

To see that the protocol is not dp-secure, consider the attack shown in Figure 5.14. Here, a dishonest prover hijacks the RBE phase of an honest prover. We coin such an attack a distance hijacking attack [60]. This type of attack is not covered by the three classes of attacks usually considered, but it is not hard to see that in scenarios such as the location-based access control system described in Section 2.1.4.3, it is a valid threat that a distance bounding protocol should prevent.

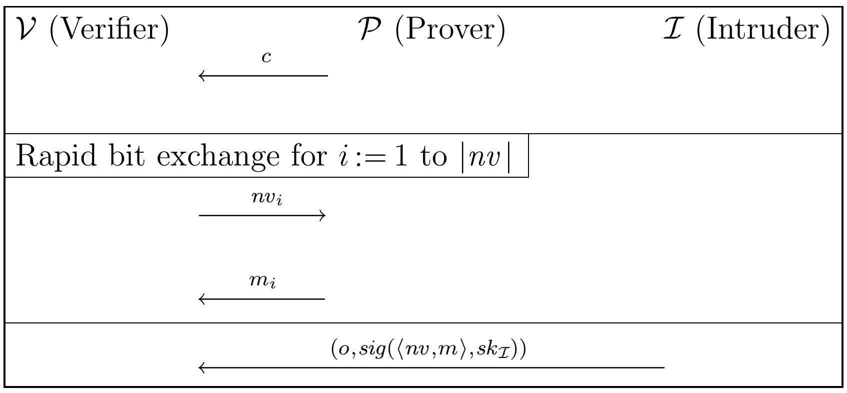

$\mathcal{V}$ (Verifier)
$\mathcal{P}$ (Prover)
$\mathcal{I}$ (Intruder)
c
Rapid bit exchange for $i := 1$ to $|nv|$
nv_i
m_i
$(o, sig(\langle nv, m\rangle, sk_{\mathcal{I}}))$

Figure 5.14. Distance Hijacking attack on Brands-Chaum protocol.

For the attack to work, we assume that the intruder $\mathcal{I}$ can either compute the opening o from the commit c and the messages exchanged in the rapid bit-exchange phase or overhears the authentication message from $\mathcal{P}$, but ensures that it does not arrive at $\mathcal{V}$.

Assuming that the distances between $\mathcal{V}$ and $\mathcal{P}$, $\mathcal{V}$ and $\mathcal{I}$, and $\mathcal{P}$ and $\mathcal{I}$ are 5, 6, and 1 respectively, the attack is captured by the following trace in $Tr(protoBS)$.

$$\begin{array}{ll}
[& (0, \mathsf{Send}\ Tx^r_{\mathcal{P}}\ (\mathsf{Hash}\ (\mathsf{Nonce}\ \mathcal{P}\ 0))\ [\mathcal{P}_1, \mathsf{Nonce}\ \mathcal{P}\ 0]),\\
& (5, \mathsf{Recv}\ Rx^r_{\mathcal{V}}\ (\mathsf{Hash}\ (\mathsf{Nonce}\ \mathcal{P}\ 0))),\\
& (6, \mathsf{Send}\ Tx^r_{\mathcal{V}}\ (\mathsf{Nonce}\ \mathcal{V}\ 0)\ [\mathcal{V}_1, \mathsf{Nonce}\ \mathcal{V}\ 0, \mathsf{Hash}\ (\mathsf{Nonce}\ \mathcal{P}\ 0)]),\\
& (11, \mathsf{Recv}\ Rx^r_{\mathcal{P}}\ (\mathsf{Nonce}\ \mathcal{V}\ 0)),\\
& (11, \mathsf{Send}\ Tx^r_{\mathcal{P}}\ ((\mathsf{Nonce}\ \mathcal{P}\ 0) \oplus (\mathsf{Nonce}\ \mathcal{V}\ 0))\ [\mathcal{P}_2, \mathsf{Nonce}\ \mathcal{P}\ 0, \mathsf{Nonce}\ \mathcal{V}\ 0]),\\
& (12, \mathsf{Recv}\ Rx^r_{\mathcal{I}}\ (\mathsf{Nonce}\ \mathcal{V}\ 0)),\\
& (12, \mathsf{Recv}\ Rx^r_{\mathcal{I}}\ ((\mathsf{Nonce}\ \mathcal{P}\ 0) \oplus (\mathsf{Nonce}\ \mathcal{V}\ 0))),\\
& (16, \mathsf{Recv}\ Rx^r_{\mathcal{V}}\ ((\mathsf{Nonce}\ \mathcal{P}\ 0) \oplus (\mathsf{Nonce}\ \mathcal{V}\ 0))),\\
& (17, \mathsf{Send}\ Tx^r_{\mathcal{I}}\ sig(\langle \mathsf{Nonce}\ \mathcal{V}\ 0, (\mathsf{Nonce}\ \mathcal{P}\ 0) \oplus (\mathsf{Nonce}\ \mathcal{V}\ 0)\rangle, sk_{\mathcal{I}})\ []),\\
& (23, \mathsf{Recv}\ Rx^r_{\mathcal{V}}\ sig(\langle \mathsf{Nonce}\ \mathcal{V}\ 0, (\mathsf{Nonce}\ \mathcal{P}\ 0) \oplus (\mathsf{Nonce}\ \mathcal{V}\ 0)\rangle, sk_{\mathcal{I}})),\\
& \big(0, \mathsf{Claim}\ \mathcal{V}\ \big\langle \mathcal{I}, (16-6) * \tfrac{c}{2} \big\rangle\big) \quad]
\end{array}$$

The security property for distance bounding protocols that we use here and that captures distance hijacking, Mafia fraud, and distance fraud was introduced in [20]. The first distance hijacking attacks where independently published by the author together with Schaller, Capkun, and Basin in [20] and Malladi in [118]. In [20], the attack on a distance bounding protocol proposed by Meadows et al. in [123] was called an impersonation attack. It was jointly discovered with Cas Cremers while the author was working on formalizing the protocol in this framework. Cremers then discovered later on [60] that similar attacks also apply to the CRCS protocol and the Brands-Chaum protocol. The class of distance hijacking attacks was defined and explored in [60].

5.4.2 Distance Hijacking Attack on Wrongly Fixed Version

One possible way to prevent the distance hijacking attack on the Brands-Chaum protocol is to incorporate the prover's identity P into the rapid bit-exchange phase. A cheap way to achieve this is to modify the response to $nv \oplus np \oplus P$. Since V knows nv and receives a commitment to np beforehand, he can compare the identities in the response and the signature.

The described changes yield the protocol $protoBS^{wf} = \{bs_1, bs_2, bs_3^{wf}, bs_4^{wf}, bs_5^{wf}\}$ where the first two steps remain unchanged and the final three steps are defined as follows.

3. In the fast response of the prover, we use $\oplus$ to combine the identity with the two nonces.

$$\frac{(t_R^P, \mathsf{Recv}\ Rx^r_P\ nv) \in tr \qquad (t_S^P, \mathsf{Send}\ Tx^r_P\ com\ [\mathcal{P}_1, np]) \in tr}{(\mathsf{SendA}\ r\ [\mathcal{P}_2, np, nv], nv \oplus np \oplus P) \in bs^{wf}{}_3(P, tr, t)}$$

4. In the authenticated response of the prover, we have to account for the modified fast-

response.

$$\frac{(t_S^P, \mathsf{Send}\ Tx_P^r\ \mathrm{resp}\ [\mathcal{P}_2, np, nv]) \in tr}{(\mathsf{SendA}\ r\ [\mathcal{P}_3, np, nv], sig(\langle nv, nv \oplus np \oplus P\rangle, sk_P)) \in bs_4^{wf}(P, tr, t)}$$

5. In the final step of the verifier, we also account for the modified fast-response.

$$\frac{\begin{array}{c}(t_S^V, \mathsf{Send}\ Tx_V^r\ nv\ [\mathcal{V}_1, nv, com]) \in tr \\ (t_{\mathrm{Resp}}^V, \mathsf{Recv}\ Rx_V^r\ m) \in tr \\ (t_{\mathrm{Auth}}^V, \mathsf{Recv}\ Rx_V^r\ sig(\langle nv, m\rangle, sk_P)) \in tr \\ np = m \oplus nv \oplus P \quad com = \mathsf{Hash}\ np\end{array}}{\left(\mathsf{ClaimA}, \left\langle P, (t_{Resp}^V - t_S^V) * \frac{c}{2}\right\rangle\right) \in bs_5^{wf}(V, tr, t)}$$

The protocol is still hp-secure and the changes prevent the concrete attack from Figure 5.14. Nevertheless, the protocol does not provide dp-security. Assume that the identities of $\mathcal{P}$ and a dishonest prover $\mathcal{I}$ only differ in the k-th bit. Then the distance hijacking attack given in Figure 5.15 is possible. It is clear that the attack works whenever the Hamming-distance between $\mathcal{P}$ and $\mathcal{I}$ is small enough such that it is possible to guess the required number of bits.

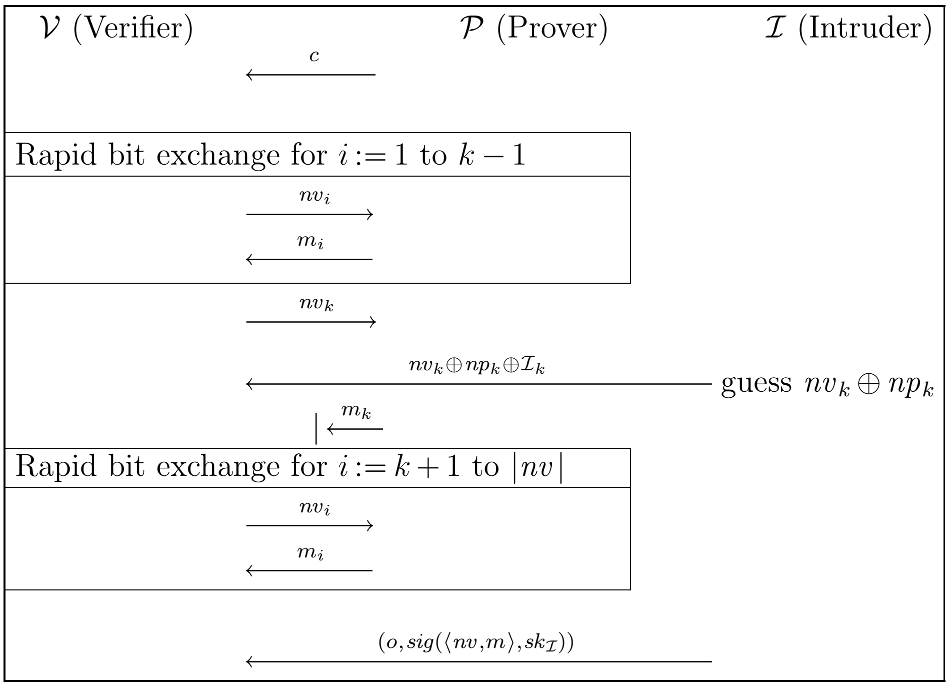

$\mathcal{V}$ (Verifier)
$\mathcal{P}$ (Prover)
$\mathcal{I}$ (Intruder)
c
Rapid bit exchange for $i := 1$ to $k-1$
nv_i
m_i
nv_k
$nv_k \oplus np_k \oplus \mathcal{I}_k$
guess $nv_k \oplus np_k$
m_k
Rapid bit exchange for $i := k+1$ to $|nv|$
nv_i
m_i
$(o, sig(\langle nv, m\rangle, sk_{\mathcal{I}}))$

Figure 5.15. Distance Hijacking attack on wrongly fixed version of Brands-Chaum.

Assuming that the distances between $\mathcal{V}$ and $\mathcal{P}$, $\mathcal{V}$ and $\mathcal{I}$, and $\mathcal{P}$ and $\mathcal{I}$ are 5, 6, and 1 respectively, the attack is captured by the following trace in $Tr(protoBS)$.

$$
\begin{array}{l}
[\ \ (0, \mathsf{Send}\ Tx^r_{\mathcal{P}}\ (\mathsf{Hash}\ (\mathsf{Nonce}\ \mathcal{P}\ 0))\ [\mathcal{P}_1, \mathsf{Nonce}\ \mathcal{P}\ 0]),\\
\quad (5, \mathsf{Recv}\ Rx^r_{\mathcal{V}}\ (\mathsf{Hash}\ (\mathsf{Nonce}\ \mathcal{P}\ 0))),\\
\quad (6, \mathsf{Send}\ Tx^r_{\mathcal{V}}\ (\mathsf{Nonce}\ \mathcal{V}\ 0)\ [\mathcal{V}_1, \mathsf{Nonce}\ \mathcal{V}\ 0, \mathsf{Hash}\ (\mathsf{Nonce}\ \mathcal{P}\ 0)]),\\
\quad (11, \mathsf{Recv}\ Rx^r_{\mathcal{P}}\ (\mathsf{Nonce}\ \mathcal{V}\ 0)),\\
\quad (11, \mathsf{Send}\ Tx^r_{\mathcal{P}}\ ((\mathsf{Nonce}\ \mathcal{P}\ 0) \oplus (\mathsf{Nonce}\ \mathcal{V}\ 0) \oplus \mathcal{P})\ [\mathcal{P}_2, \mathsf{Nonce}\ \mathcal{P}\ 0, \mathsf{Nonce}\ \mathcal{V}\ 0]),\\
\quad (12, \mathsf{Recv}\ Rx^r_{\mathcal{I}}\ (\mathsf{Nonce}\ \mathcal{V}\ 0)),\\
\quad (12, \mathsf{Recv}\ Rx^r_{\mathcal{I}}\ ((\mathsf{Nonce}\ \mathcal{P}\ 0) \oplus (\mathsf{Nonce}\ \mathcal{V}\ 0))),\\
\quad (16, \mathsf{Recv}\ Rx^r_{\mathcal{V}}\ ((\mathsf{Nonce}\ \mathcal{P}\ 0) \oplus (\mathsf{Nonce}\ \mathcal{V}\ 0) \oplus \mathcal{I})),\\
\quad (17, \mathsf{Send}\ Tx^r_{\mathcal{I}}\ sig(\langle \mathsf{Nonce}\ \mathcal{V}\ 0, (\mathsf{Nonce}\ \mathcal{P}\ 0) \oplus (\mathsf{Nonce}\ \mathcal{V}\ 0) \oplus \mathcal{I}\rangle, sk_{\mathcal{I}})\ []),\\
\quad (23, \mathsf{Recv}\ Rx^r_{\mathcal{V}}\ sig(\langle \mathsf{Nonce}\ \mathcal{V}\ 0, (\mathsf{Nonce}\ \mathcal{P}\ 0) \oplus (\mathsf{Nonce}\ \mathcal{V}\ 0) \oplus \mathcal{I}\rangle, sk_{\mathcal{I}})),\\
\quad \big(0, \mathsf{Claim}\ \mathcal{V}\ \big\langle \mathcal{I}, (16-6) * \tfrac{c}{2} \big\rangle\big)\ \ ]
\end{array}
$$

Note that even though the message $nv \oplus np \oplus \mathcal{P}$ is transmitted by the prover at time 11, the verifier receives the different message $nv \oplus np \oplus \mathcal{I}$ at time 16. This is possible since the Hamming-distance between the two messages might be small, i.e.,

$$nv \oplus np \oplus \mathcal{P} \oplus nv \oplus np \oplus \mathcal{I} = \mathcal{P} \oplus \mathcal{I} \in \mathrm{LHW}.$$

5.4.3 Security of Version with Explicit Binding

The approach to bind the rapid bit-exchange phase to the prover's identity is valid. The problem with the previous fix is the use of *Xor* to achieve this binding. We therefore use a hash function and replace the response $nv \oplus np$ from the original protocol by $nv \oplus h(np, \mathcal{P})$. The previous attack is not possible anymore since even if the Hamming-distance between $\mathcal{P}$ and $\mathcal{I}$ is small, the Hamming-distance between $h(np, \mathcal{P})$ and $h(np, \mathcal{I})$ is not small in general. Note that we can now drop the commitment without becoming susceptible to the distance fraud attack described earlier. If $\mathcal{I}$ jumps the gun and sends some response m before receiving nv, then he has to find some np such that $m = nv \oplus h(np, \mathcal{I})$. This is not possible for a non-invertible hash function.

We therefore define the protocol as $protoBS^{expl} = \{bs_1^{expl}, bs_2^{expl}, bs_3^{expl}, bs_4^{expl}\}$.

1. There is no commitment, the first step bs_1 formalizes the sending of the challenge by the verifier.

$$\frac{(\mathsf{Nonce}\ V\, j) \notin used(tr)}{(\mathsf{SendA}\ r\ [\mathcal{V}_1, \mathsf{Nonce}\ V\, j], \mathsf{Nonce}\ V\, j) \in bs_1^{expl}(V, tr, t)}$$

   The new event is associated with the protocol state denoting that the first step of the verifier has been executed with the given nonce.

2. The second step bs_2 formalizes the response to the challenge by the prover.

$$\frac{\begin{array}{c}(\mathsf{Nonce}\ P\, i) \notin used(tr)\\ (t_R^P, \mathsf{Recv}\ Rx_P^r\ nv) \in tr\end{array}}{(\mathsf{SendA}\ r\ [\mathcal{P}_1, \mathsf{Nonce}\ P\, i, nv], nv \oplus \mathsf{Hash}\ \langle \mathsf{Nonce}\ P\, i, P\rangle) \in bs_2^{expl}(P, tr, t)}$$

   Instead of sending the *Xor* of the verifier and prover nonces, the prover sends the *Xor* of the verifier's nonce and the hash of the prover's nonce and the prover's identity.

3. The third step bs_3 formalizes the delayed authentication of the response by the prover.

$$\frac{(t_S^P, \mathsf{Send}\ Tx_P^r\ \text{resp}\ [\mathcal{P}_1, np, nv]) \in tr}{(\mathsf{SendA}\ r\ [\mathcal{P}_2, np, nv], sig(\langle nv, np\rangle, sk_P)) \in bs_3^{expl}(P, tr, t)}$$

The prover signs np instead of the response for this variant of the protocol since there is no way to compute np from $nv \oplus h(np, P)$.

4. The fourth step bs_4 formalizes the claim by the receiver after receiving both the response to the challenge and the authentication message.

$$\frac{\begin{array}{c}(t_S^V, \mathsf{Send}\ Tx_V^r\ nv\ [\mathcal{V}_1, nv]) \in tr \\ (t_{\text{Resp}}^V, \mathsf{Recv}\ Rx_V^r\ m) \in tr \\ (t_{\text{Auth}}^V, \mathsf{Recv}\ Rx_V^r\ sig(\langle nv, np\rangle, sk_P)) \in tr \\ m = \text{nv} \oplus \mathsf{Hash}\ \langle np, P\rangle\end{array}}{\left(\mathsf{ClaimA}, \left\langle P, (t_{Resp}^V - t_S^V) * \frac{c}{2}\right\rangle\right) \in bs_4^{expl}(V, tr, t)}$$

The commitment check is replaced by a check that the fast response is the Xor of nv and the hash.

Theorem 5.14. *The protocol $protoBS^{impl}$ is hp-secure and dp-secure.*

Proof. (Sketch) The proof of hp-security is similar to the proof for the original Brands-Chaum protocol. For dp-security we proceed as follows. If there is a an event $\mathsf{Claim}\ V\ \langle I, d\rangle$ for a dishonest prover I, then V must have sent the challenge $\mathsf{Nonce}\ V\ j$ at some time t_S^V and received a response $(\mathsf{Nonce}\ V\ j) \oplus (\mathsf{Hash}\ \langle np,\ I\rangle)$ at time t_R^V. There must be some agent C who performs the corresponding send of a message containing the component $m' = (\mathsf{Nonce}\ V\ j) \oplus (\mathsf{Hash}\ \langle np,\ I\rangle) \oplus l$ with $l \in LHW$ at time t_S^C such that $t_R^V - t_S^C \geq cdist_{LoS}(C, V)$. By Lemma 5.2, we also know that $t_S^C - t_S^V \geq cdist_{LoS}(V, C)$ and hence $t_R^V - t_S^V \geq cdist_{LoS}(C, V) * 2$. We now perform a case distinction on C. If C is honest, then only step bs_2^{expl} send a message $nv \oplus (\mathsf{Hash}\ \langle \mathsf{Nonce}\ C\ j, C\rangle)$ that might match this message. Since $\mathsf{Hash}\ \langle \mathsf{Nonce}\ C\ j, C\rangle$ can not appear in nv since the nonce is fresh, we have $\mathsf{Hash}\ \langle \mathsf{Nonce}\ C\ j, C\rangle \in parts(m')$ which is impossible. C must therefore be a dishonest agent and we can conclude from $t_R^V - t_S^V \geq cdist_{LoS}(C, V) * 2$ that $(t_R^V - t_S^V) * \frac{c}{2} \geq |loc_V - loc_C|$. □

Note that for this variant, we did not require the assumption that a radio transceiver can never receive its own transmitted signal since $h(x, y)$ never has low Hamming-weight.

5.4.4 Security of Version with Implicit Binding

Instead of including $\mathcal{P}$'s identity in the rapid bit-exchange phase, it is also possible to include the identity in the commitment message. Instead of $commit(np)$, the prover sends $commit(np, \mathcal{P})$. Since the commitment is hiding, nobody else can commit on np and a different identity before the rapid bit-exchange starts. The resulting protocol $protoBS^{impl} = \left\{bs_1^{impl}, bs_2, bs_3, bs_4, bs_5^{impl}\right\}$ is identical to the original protocol except for the first and last step.

1. In the commit step of the prover, we include the identity P in the hash.

$$\frac{(\mathsf{Nonce}\ P\ i) \notin used(tr)}{(\mathsf{SendA}\ r\ [\mathcal{P}_1, \mathsf{Nonce}\ P\ i], \mathsf{Hash}\ \langle \mathsf{Nonce}\ P\ i, P\rangle) \in bs_1^{impl}(P, tr, t)}$$

5. In the final step of the verifier, we adapt the check of the commitment.

$$\frac{\begin{array}{c}(t^V_S, \mathsf{Send}\ Tx^r_V\ nv\ [\mathcal{V}_1, nv, com]) \in tr \\ (t^V_{\text{Resp}}, \mathsf{Recv}\ Rx^r_V\ m) \in tr \\ (t^V_{\text{Auth}}, \mathsf{Recv}\ Rx^r_V\ sig(\langle nv, m\rangle, sk_P)) \in tr \\ np = m \oplus \text{nv} \quad com = \mathsf{Hash}\ \langle np, P\rangle\end{array}}{\left(\mathsf{ClaimA}, \left\langle P, (t^V_{Resp} - t^V_S) * \frac{c}{2}\right\rangle\right) \in bs^{impl}_5(V, tr, t)}$$

Theorem 5.15. *The protocol protoBS^{impl} is hp-secure and dp-secure.*

Proof. (Sketch) The proof of hp-security is similar to the proof for the original Brands-Chaum protocol. For dp-security we proceed as follows. If there is a an event $\mathsf{Claim}\ V\ \langle I, d\rangle$ for a dishonest prover I, then V must have sent the challenge $\mathsf{Nonce}\ V\ j$ at some time t^V_S and received a response $m = (\mathsf{Nonce}\ V\ j) \oplus np$ at time t^V_R. V must also have received a commitment $\mathsf{Hash}\ \langle np, I\rangle$ before t^V_R. Since the commitment on np is received before $\mathsf{Nonce}\ V\ j$ is first used, we know that $\mathsf{Nonce}\ V\ j \in parts(m)$. There must be some agent C who performs the corresponding send of a message containing the component $m' = (\mathsf{Nonce}\ V\ j) \oplus np \oplus l$ with $l \in LHW$ at time t^C_S such that $t^V_R - t^C_S \geq cdist_{LoS}(C, V)$. If C is dishonest, then we can finish the proof analogously to the proof for the version with explicit binding. If C is honest, then m' can be either sent in the challenge step or in the response step. If it is sent in the challenge step and $C = V$, then this contradicts our assumption that no agent can receive a radio signal that he transmitted himself. If $C \neq V$, then $m' = \mathsf{Nonce}\ C\ k$ cannot contain $\mathsf{Nonce}\ V\ j$. If m' is sent in the response step, then it is of the form $nv \oplus (\mathsf{Nonce}\ C\ k)$ and $(\mathsf{Nonce}\ C\ k) \in parts(m')$ since nv cannot contain this nonce such that it cancels out. Since $m = m' \oplus l$ and the nonce is not of low Hamming-weight, we know that $(\mathsf{Nonce}\ C\ k) \in parts(np)$ and $(\mathsf{Nonce}\ V\ j) \in parts(nv)$. Hence $t^C_S \geq t^V_S$ and I must have committed to np before t^S_C where $\mathsf{Nonce}\ C\ k$ was first used outside of $\mathsf{Hash}\ \langle \mathsf{Nonce}\ C\ k, C\rangle$. Since it is impossible to obtain $\mathsf{Hash}\ \langle np, I\rangle$ from $\mathsf{Hash}\ \langle \mathsf{Nonce}\ C\ k, C\rangle$ such that $\mathsf{Nonce}\ C\ k \in parts(np)$, this is a contradiction. □

5.5 Isabelle Formalization

We briefly survey our Isabelle/HOL formalization [158]. Our framework consists of the following theories, depicted in Figure 5.16 along with their dependencies.

Message Theory. Our message theory models a free term algebra and is based on Paulson's work [140]. For the analysis of distance bounding protocols, we have extended the theory with support for XOR.

Geometric Properties of $\mathbb{R}^3$. Since agent's locations are vectors in $\mathbb{R}^3$, we use the formalization of real numbers provided in Isabelle's standard library.

Parameterized Communication Systems. Rules describe the network properties, possible intruder actions, and the protocol steps. Together, these inductively define the possible set of traces.

Protocol Formalizations. These are given by sets of step functions, formalizing the actions taken by agents running the protocol. For a given protocol, we instantiate the inductive rules with the corresponding step functions to obtain all possible execution traces. Security properties of the protocol are then proved by induction or using the inherited protocol-independent properties, where possible.

Protocol Independent Properties. Parameterizing the set of possible traces by protocol step functions allows us to prove protocol independent system properties.

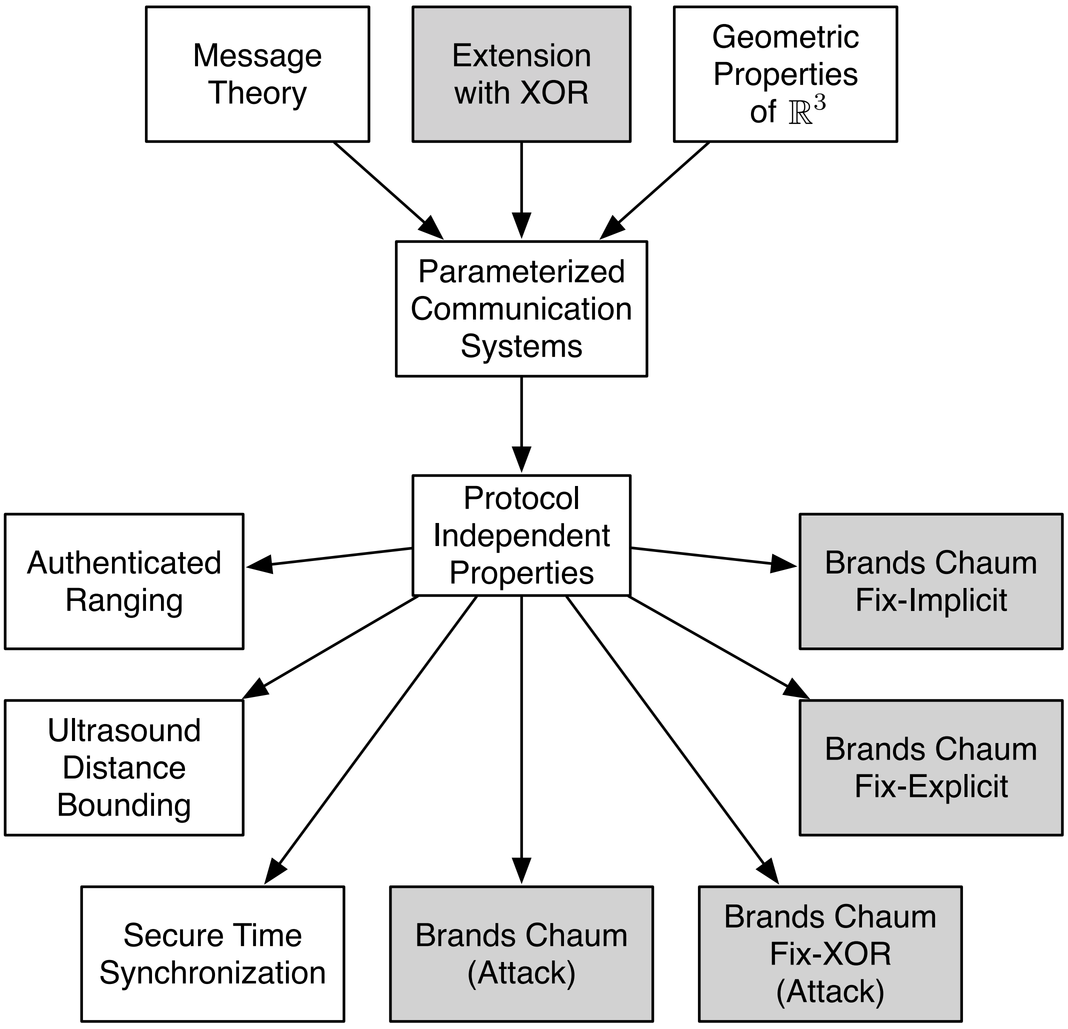

Message Theory
Extension with XOR
Geometric Properties of $\mathbb{R}^3$
Parameterized Communication Systems
Protocol Independent Properties
Authenticated Ranging
Brands Chaum Fix-Implicit
Ultrasound Distance Bounding
Brands Chaum Fix-Explicit
Secure Time Synchronization
Brands Chaum (Attack)
Brands Chaum Fix-XOR (Attack)

Figure 5.16. Theories and Dependencies of our Isabelle Formalization.

Most of our formalization consists of general results applicable to arbitrary protocols. For the basic version without XOR, the security proofs of the concrete protocols are therefore comparably small. For the case studies with XOR, we could not reuse so many protocol independent results and reasoning about message deduction for XOR was considerably harder than for the free algebra. Table 5.1 compares the sizes of the different parts of the formalization.

Theory	Lines of Code	Lemmas	Definitions	Pages
Message Theory	2221	229	18	43
(Extension with XOR)	3668	238	21	137
Geometric Properties of $\mathbb{R}^3$	272	13	6	6
Parameterized Communication Systems	633	51	4	12
Protocol Independent Properties	2342	57	4	39
Authenticated Ranging	489	57	6	11
Ultrasound Distance Bounding	681	15	6	15
Secure Time Synchronization	961	20	10	22
Brands Chaum (Attack)	635	7	10	6
Brands Chaum Fix-XOR (Attack)	662	7	10	6
Brands Chaum Fix-Explicit	1389	16	9	30
Brands Chaum Fix-Implicit	2407	35	9	41
Total	16360	745	113	368

Table 5.1. Statistics about our Isabelle Formalization.

Chapter 6
Related Work

In this chapter, we discuss related work. First, we discuss related work for Chapter 3 and compare our approach to existing approaches for security protocol analysis. Then, we discuss related work for our impossibility results from Chapter 4. Finally, we discuss related work to our framework for the analysis of physical protocols from Chapter 5.

6.1 Security Protocol Analysis

In the following we compare existing approaches to security protocol analysis with respect to three different aspects. First, we compare the different formalisms to define cryptographic operators, their properties, and the message deduction capabilities of the adversary. We focus on the equational theories used to model DH exponentiation and bilinear pairings. Second, we compare the different formalisms to specify protocols and define their executions. Third, we compare the different formalism to specify security properties. Afterwards we compare three existing approaches for the unbounded analysis of security protocols in more detail: ProVerif [28], Maude-NPA [77], and approaches based on backwards search such as Athena [162], CPSA [150], and Scyther [62].

6.1.1 Cryptographic Messages

Existing models for cryptographic messages can be roughly partitioned into those with and without *explicit destructors*. In models without explicit destructors, there are no function symbols for decryption and projection and these operations are captured by message deduction rules. Protocol steps use pattern matching to denote that a message is decrypted or a signature is checked. Since there are no function symbols for destructors, these approaches can use the free algebra. Models with explicit destructors represent operations that deconstruct messages by function symbols and equations. These approaches use terms modulo the equational theory generated by the equations for the destructors. The message deduction capabilities of the adversary correspond to applying function symbols modulo this equational theory. In Chapter 5, we use a model without explicit destructors. In Chapter 3 and Chapter 4, we use explicit destructors. We also show that by computing the variants of protocol and message deduction rules, we can get rid of the equational theory induced by a subterm-convergent rewriting system.

Equational Theories for Diffie-Hellman

Existing symbolic approaches that support Diffie-Hellman exponentiation differ in the supported operations and the equations that are taken into account. The simplest equational theory uses a fixed base g and is generated by the equation $(\mathsf{g}\hat{~}x)\hat{~}y \simeq (\mathsf{g}\hat{~}y)\hat{~}x$. This theory is used by Goubault-Larrecq [90] who uses resolution modulo theories such as this one to verify DH protocols. The theory is also used by Blanchet et al. [31] in ProVerif. As a next step, the restriction to a fixed base can be lifted by replacing g with a variable:

$$(g\hat{~}x)\hat{~}y \simeq (g\hat{~}y)\hat{~}x \quad (\mathbf{DH1}).$$

This theory accounts for arbitrary nestings of exponentiations, for instance, it holds that $((\mathsf{g}\hat{~}\mathsf{a})\hat{~}\mathsf{b})\hat{~}\mathsf{c} \approx ((\mathsf{g}\hat{~}\mathsf{c})\hat{~}\mathsf{a})\hat{~}\mathsf{b}$. This theory is used in Maude-NPA [76] and in the reduction results by Meadows and Lynch [117] and Mödersheim [132] who show that reasoning modulo this theory can be reduced to syntactic reasoning for restricted classes of protocols.

By adding the function symbol $\_^{-1}$ for inverses of exponents and the equations

$$(x\hat{~}y)\hat{~}(y^{-1}) \simeq x \; (\mathbf{DH2}) \quad \text{and} \quad (x^{-1})^{-1} \simeq x \quad (\mathbf{DH3}),$$

we obtain the equational theory used by Küsters and Truderung [108]. An alternative approach is to model the exponents as an abelian group with the operations $(*, \_^{-1}, 1)$ and to combine the equational theory AG for abelian groups with

$$(g\hat{~}x)\hat{~}y \simeq g\hat{~}(x*y) \; (\mathbf{DH1'}) \quad \text{and} \quad g\hat{~}1 \simeq g \quad (\mathbf{DH2'}).$$

This is the equational theory that we use in Chapter 3 and the same equational theory is also used in the decidability results for secrecy with respect to a bounded number of protocol sessions by Shmatikov [131] and Chevalier et al. [45]. In their works, Shmatikov does not allow the adversary to multiply exponentiations and Chevalier et al. allow products only in exponents. With the restriction of Chevalier et al., Küsters and Truderung [108] show that the adversary for the equational theory generated by AG, **DH1'**, and **DH2'** is equally powerful to the adversary for the equational theory generated by **DH1–3**. For all of the theories mentioned so far, unification is decidable, see for example Meadows and Narendran [122].

For theories that go beyond the previously mentioned ones, for example by modeling multiplication of DH group elements and the equation

$$(g_1*g_2)\hat{~}x \simeq (g_1\hat{~}x)*(g_2\hat{~}x) \quad (\mathbf{M1}),$$

the situation is different. For many theories that include **M1** and **DH1** or **DH1'**, it has been shown by Kapur et al. [100, 99, 101] that unification is undecidable. Furthermore, there is is no such theory for which a unification algorithm is known to exist. To circumvent the problems with unification, Dougherty and Guttman [70] propose an alternative approach to perform manual security proofs. Their proof technique abstracts away from concrete exponentiations and reasons about vectors of integers that characterize how many times secret values occur in exponents. Their approach uses an order-sorted term algebra and models the exponents as a field.

Equational Theories for Bilinear Pairings

Intuitively an equational theory for bilinear pairings models two DH groups $\mathbb{G}_1$ and $\mathbb{G}_2$ and a bilinear map $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. If the equational theory does not fix the bases of the DH groups, then multiple generators of the same group $\mathbb{G}_1$ can be used. This is, for example, used by identity-based key exchange protocols, which use hashes of identities as generators. Symbolic approaches where the base is not fixed usually also allow to model families $\hat{e}_i\colon \mathbb{G}_1^i \times \mathbb{G}_1^i \to \mathbb{G}_2^i$ of bilinear maps and DH groups. The main difference between the symbolic models of bilinear pairings used in existing approaches is the model of the DH groups. The equational theory used by Pankova and Laud [138] extends the equational theory for DH exponentiation used by Küsters and Truderung [108] and is generated by **DH1–3** for the groups $\mathbb{G}_2^i$, similar equations for the groups $\mathbb{G}_1^i$, and the two equations

$$\hat{e}(x,y) \simeq \hat{e}(y,x) \quad \text{and} \quad \hat{e}([z]x,y) \simeq \hat{e}(y,x)\hat{\ } z$$

to model the properties of the bilinear map $\hat{e}$. They do not fix the base and support both multiple generators and families of bilinear pairings. Our treatment of bilinear pairings is similar to theirs. The main difference is that we use a representation of DH exponentiation where products of exponents are represented as terms. This is useful in our search since the symbolic term $\mathtt{g}\hat{\ }x$, where x can be instantiated with a product of arbitrary size, cannot be represented as a term in their term algebra. Kremer et al. [104] consider static equivalence for bilinear pairings and use a theory with fixed bases for $\mathbb{G}_1$ and $\mathbb{G}_2$, but model addition of exponents and multiplication in $\mathbb{G}_1$ and $\mathbb{G}_2$. Note that their method does not rely on unification.

6.1.2 Protocol Specification and Execution

Most classical security protocols have a simple control flow and can be specified by finite sequences of receive-steps and send-steps. Even though many approaches use such a protocol specification language, there are many different ways to define the corresponding protocol executions. For example, Cremers [61] defines a trace-based operational semantics, Maude-NPA [77] uses multiset rewriting modulo $\mathcal{E}$ to define the underlying semantics, and strand spaces [83] use a partially ordered semantics.

To support protocols that have a more complicated control flow or utilize mutable global state, there are three popular approaches. First, set rewriting or multiset rewriting focuses on the state changes performed by protocol steps. This approach is used, for example, by Avispa [10] and Cervesato et al. [41]. Guttman [91] proposes an extension of the strand space model with multiset rewriting to model state that is shared between different protocol sessions. Second, process calculi such as the applied pi calculus [3] focus on processes and their communication. Although there are encodings of mutable global state for such process calculi, direct support is more convenient in some settings. Hence, extensions of the applied pi calculus with mutable state have been proposed by Arapinis et al. [9]. Third, protocol implementations in *C* or *F#* can be used directly as specifications. This approach has been followed for example by Goubault-Larrecq and Parrennes [89] and Bhargavan et al. [25].

6.1.3 Property Specification Languages

The standard properties considered in security protocol analysis often fall into the class of secrecy properties or authentication properties [116]. There are different ways to specify these. For example, bad states can specified by symbolic terms [77, 10, 150] or by formulas [30, 162]. To handle authentication properties, support for inequalities or negation is usually required. Additionally, some methods provide support for temporal operators. For example, Corin et al. [54] and Armado et al. [11] propose methods that support propositional, linear temporal logics with past operators for analyzing protocols with respect to a bounded number of sessions.

6.1.4 ProVerif

Horn theory based approaches [168] such as ProVerif [28] were originally designed to reason about secrecy and similar properties that can be encoded as derivability in Horn theories. The approach is based on the following idea. Given a Horn theory $\mathcal{T}$ that encodes the initial knowledge of the adversary, the message deduction rules, and the protocol, the message s is secret if the fact $\mathsf{K}(s)$ denoting that the message s is known to the adversary is not derivable in $\mathcal{T}$. To obtain such a theory $\mathcal{T}$, the following steps are performed. First, the initial knowledge of the adversary is encoded by facts $\mathsf{K}(m)$. Second, the message deduction rules available to the adversary are encoded by clauses of the form

$$\mathsf{K}(m_1), ..., \mathsf{K}(m_k) \rightarrow \mathsf{K}(m),$$

which model that m is deducible from m_1, ..., m_k. Third, the protocol is modeled by additional message deduction clauses, abstracting away from the protocol state. For example, the clause $\mathsf{K}(enc(x, \mathrm{k_{AB}})) \rightarrow \mathsf{K}(h(x))$ models a protocol step that receives a message encrypted with the fixed symmetric key $\mathrm{k_{AB}}$ and replies with the hash of the plaintext. The abstraction employed for protocols is sound for secrecy, but can introduce false attacks since the protocol clause can be applied multiple times, even if the corresponding step can be only executed once. Based on this approach and various extensions [30, 31], ProVerif can analyze protocols specified in the applied pi calculus, authentication properties specified as correspondence properties, and also supports some equational theories that do not include associative and commutative operators.

Küsters and Truderung lifted this restriction and developed support for XOR [109] and a theory of DH exponentiation [108] that includes inverses (see Section 6.1.1). Their approach is based on the transformation of Horn theories. More precisely, they reduce derivability of a fact F in a Horn theory $\mathcal{T}$ modulo an equational theory for XOR (respectively DH) to *syntactic derivability* of a fact F' in a Horn theory $\mathcal{T}'$. To analyze syntactic derivability, they can then use ProVerif. For DH exponentiation, their reduction is restricted to exponent-ground Horn theories. A Horn theory is exponent-ground, if (roughly) no variables occur in exponents. Because of this restriction, only authentication properties that can be stated as reachability properties can be analyzed. Using standard (sound) techniques for Horn theory based protocol analysis like nonce-abstraction and considering only a fixed number of agents [50], they analyze a wide range of AKE protocols with respect to secrecy and Unknown Key Share (UKS) Attacks. Pankova and Laud [138] have recently extended this approach with support for bilinear pairings. Their approach inherits the restriction to exponent-ground theories and authentication properties that can be stated as reachability properties. The reduction from derivability modulo a theory modeling DH to syntactic derivability roughly corresponds to our usage of the finite variant property to switch from reasoning modulo $\mathcal{DH}$ to reasoning modulo AC.

Proverif has also been extended [31] to support the verification of properties that cannot be formalized as trace properties such as voting privacy, security against offline guessing, and strong secrecy. For these, reasoning about observational equivalence between different processes is required.

6.1.5 Maude-NPA

The Maude-NPA tool [77] is based on backwards narrowing modulo an equational theory $\mathcal{E}$. In contrast to our search, which is based on causal dependencies and allows us to represent different interleavings with a single constraint system, backwards narrowing represents executions by (sequences of) symbolic states. Since considering all interleavings would be too expensive, Maude-NPA employs various state space reduction techniques [78] to ignore some interleavings. The backwards narrowing employed by Maude-NPA supports all equational theories $\mathcal{E}$ with a finite variant decomposition $(\Sigma, \mathcal{R}, \mathcal{AX})$ where $\mathcal{AX}$ is one of AC, ACI, or C. For backwards narrowing, Maude-NPA uses folding variant narrowing to perform equational unification modulo $\mathcal{E}$. While the backwards narrowing is very general, some of the state space reduction techniques, such as grammar generation, have to be adapted for new equational theories $\mathcal{E}$. This is roughly similar to our approach, where support for new theories $\mathcal{E}$ that have a finite variant decomposition still requires the development of normal message deduction rules and normal-form conditions. Erbatur et al. [75] present a variation of backwards narrowing modulo $\mathcal{E}$ where certain parts of the symbolic state have to remain irreducible. This is similar to our usage of $\mathcal{R}$,$\mathcal{AX}$-variants of rules and our pruning of constraint systems that contain reducible node constraints with **N1**.

6.1.6 Athena, CPSA, Scyther

Athena [162], CPSA [150], Scyther [62], and its proof-generating version scyther-proof [125] are closest to our approach. The main difference is that our property specification language is more expressive and they are restricted to linear role scripts and the free algebra. In our terminology, they all start with an initial constraint system that characterizes all possible attacks and search for models of the constraint system that are also executions of the considered protocol. In the search, they exploit causal dependencies and add protocol steps that must have been executed and messages that must have been deduced. The reasoning about protocol steps and their ordering is similar to ours. The main difference here is that in Scyther and Athena, different protocol threads can never be identified later on in the search, which is possible in CPSA, scyther-proof, and Tamarin. For reasoning about message deduction, the differences between these approaches are more pronounced. CPSA uses authentication tests, which focus on the origination of nonces and encryptions included in messages. Like Tamarin, Athena, Scyther, and scyther-proof use a mixed forward and backward search and enforce that the same message is never learned twice. The main difference between these approaches and Tamarin is that Tamarin represents the proof tree for message deduction explicitly because the restriction that the same message is never learned twice is not sufficient for equational theories such as DH. Note that Scyther has been extended with support for a large [22], but fixed number of compromising adversary models.

6.2 Impossibility Results

We first summarize impossibility results for protocols. Then, we list some impossibility results for cryptography. Finally, we discuss related work on security relations.

6.2.1 Impossibility Results and Bounds for Protocols

There are few existing impossibility results for security protocols. Pereira and Quisquater [141] prove the generic insecurity for a class of authenticated group key agreement protocols. Micciancio and Panjwani [130] consider the related question of lower bounds on the communication complexity of security protocols. They establish a lower bound on the communication complexity for a class of multicast key distribution protocols. Closest to our work is the result by Muniz and Laud [87] that proves that it is impossible to devise perennial message recognition protocols using XOR and hashing alone.

6.2.2 Impossibility Results in Cryptography

Canetti and Fischlin [36] prove the impossibility of constructing two party bit commitment protocols in the (plain) UC model. Backes et al. [17] show that it is impossible to obtain cryptographic soundness results in the sense of Blackbox Reactive Simulatability (RSIM) or Universal Composability (UC) for standard symbolic models of hash functions as free function symbols. Backes and Pfitzmann [16] also show a similar result for modeling XOR.

6.2.3 Formal Models of Security Relations

Several authors have investigated sufficient conditions for establishing security relations between agents based on knowledge and properties of cryptographic operators. Maurer and Schmid [120] present a channel calculus that describes how an insecure channel can be transformed into a channel providing security guarantees. The transformations rely on other channels with given properties that are used in combination with cryptographic operators. Boyd [33] presents a formal model using the language Z. His model builds on the abstract types *users* and *keys*, where communication channels are modeled as relations on the set of users and security properties of channels are predicates on the distribution of keys. Boyd then defines a similar set of secure channel transformations to those presented by Maurer and Schmid [120]. In terms of impossibility, both papers propose that any secure channel transformation must be based on a previously existing security relation, i.e., you cannot get security from nothing. Whereas Maurer and Schmid [120] propose this as an axiom, Boyd [33] proves that it is a property of his abstract model.

6.3 Protocol Analysis with Time, Network Topology, and Location

We now summarize formal approaches that address aspects of time, network topology, and location. Whereas the related works are restricted to specific types of protocols and address at most one or two of these aspects, our model combines all three aspects and is therefore applicable to a wider range of protocols. For example, there has been, to the best of our knowledge, no formal analysis of an ultrasound distance bounding protocol before. Such an analysis obviously requires a model that takes into account time, nodes' locations, and a network topology that reflects properties of different communication media.

Time

Most approaches formalizing time like Delzanno and Ganty [64] or Evans and Schneider [80] focus on timestamps, which are used to reason about key-expiration, e.g., in protocols like Kerberos. These models are based on discrete time, whereas Sharp and Hansen [161] use dense time in their extension of Strand Spaces with time. Corin et al. [53] use timed automata [6] to model timing attacks and timing issues like timeouts and retransmissions in security protocols. Gorrieri et al. [88] use a real-time process algebra to model and analyze μ-TESLA. The protocol is proved to achieve a time-dependent form of integrity for the messages sent by the broadcast source, abstracting away from the network and the topology.

Network Topology

Network topology has been considered in formal approaches for analyzing routing protocols in ad-hoc networks. In Acs et al. [4], a mathematical framework for the analysis of routing protocols is proposed. Nanz and Hankin [133] present a process calculus with broadcast communication that accounts for network connectivity and analyze route consistency for ad-hoc routing protocols. Arnaud and Cortier [13] use a related process calculus and devise a decision procedure for analyzing routing protocols with respect to a bounded number of sessions and a fixed network topology. Cortier et al. [55] prove that it is sufficient to consider four node topologies for certain types of properties. Yang and Baras [169] present a semi-decision procedure that is used to find attacks on route stability.

Closely related is the notion of secure neighbor discovery (see for example [139]). In this setting, a node must detect its direct communication partners, for example, as a basis for topology information used for routing. It has been shown by Poturalski et al. [144] that under certain assumptions, there is no protocol that can achieve this objective.

Location and Distance

Node location has, to our knowledge, only been used in informal proofs. For example, Sastry et al. [156] propose a protocol for verifying location claims based on ultrasonic communication and provide an informal proof of its security and reliability. Avoine et al. [14] present a framework for classifying different attack scenarios for distance bounding protocols. Other approaches only formalize the related notion of relative distance. In Meadows et al. [123], an authentication logic is extended to handle relative distance and is used to prove the security of a newly proposed distance bounding protocol. Here, the distance between two nodes is axiomatically defined as the minimal time-of-flight of a message from the verifier to the prover and back. Different signal propagation speeds are not captured in the model. Dürholz et al. [74] have developed a computational model to analyze RFID distance-bounding protocols in the single-prover single-verifier setting. Malladi [118] extends existing constraint solving techniques to discover attacks on distance bounding protocols considering a fixed number of sessions.

Chapter 7
Conclusion

This thesis consists of three parts and we discuss conclusions and possible future work for each part separately.

Automated Protocol Analysis

For our approach, we have used a security protocol model based on multiset rewriting modulo an equational theory $\mathcal{E}$ and two-sorted first-order logic [47] as a starting point. To develop our constraint solving algorithm for falsification and unbounded verification, the following ingredients where key. The switch to dependency graphs allowed us to obtain a joint representation of protocol steps and proof trees for message deduction. In our search, this allowed us to represent sets of dependency graphs symbolically by constraints. In the search, we use temporal variables and ordering constraints to represent multiple interleavings at once. For message deduction, the dependency graphs contain proof *graphs* where sharing of subproofs is explicit and message deduction for all deduced messages is considered jointly. This, and the switch from $\mathcal{DH}$-instances to AC-instances of message deduction rules using a finite variant decomposition of $\mathcal{E}$, allowed us to develop a proof theory for message deduction in our setting. The proof theory allows us to ignore non-normal proofs in the search.

We have implemented our algorithm and demonstrated that it works well for AKE protocols, tripartite group key exchange protocols, and identity-based AKE protocols. Elsewhere [124, 107], it has been shown that it works well for classical security protocols and protocols with loops and global mutable state. In the future, we plan to evaluate our approach with additional case studies from different areas such as website authorization [18].

Unfortunately, we do not have any formal results about the termination behavior of the algorithm yet. It would be worthwhile to investigate two questions. First, we believe the algorithm can be used to decide secrecy for a bounded number of sessions with respect to the equational theories considered in this thesis. Using formulas that ensure the uniqueness of protocol steps, it is straightforward to encode bounded session scenarios in our approach. Second, it might be possible to prove termination with respect to an unbounded number of protocol sessions for restricted classes of protocols such as [149, 8, 32]. This might require adapting the search strategy employed for message deduction.

An obvious next step is adding support for more equational theories $\mathcal{E}$. Here, we can distinguish two different types of theories.

1. Equational theories $\mathcal{E}$ that have a finite variant decomposition $\mathcal{R}$,$\mathcal{AX}$, e.g., the theory for XOR or a theory that models blind signatures.

2. Equational theories $\mathcal{E}$ that do not have a finite variant decomposition. Since we have to reason about symbolic equality, this would still require an $\mathcal{E}$-unification algorithm or at least an algorithm that works for those unification problems that occur during protocol analysis. As an example for such theories, consider a theory modeling homomorphic encryption or a theory that models DH exponentiation with multiplication of group elements.

For 1., we can investigate the $\mathcal{R}$,$\mathcal{AX}$-variants of the message deduction rules and define normal forms for message deduction proofs. For 2., we must use $\mathcal{E}$-unification instead of $\mathcal{AX}$-unification and we have to investigate the $\mathcal{E}$-instances of message deduction rules to define normal forms. To use $\mathcal{E}$-unification, it might be necessary to identify unification problems that occur in protocol analysis and develop special purpose unification algorithms for these cases if $\mathcal{E}$-unification is undecidable in general.

We think multiset rewriting is a good way to *reason* about security protocols in our approach, similar to how ProVerif uses Horn clauses during protocol analysis. It is also convenient to specify access to global mutable state and (asynchronous) communication using multiset rewriting. For sequential computations with local state, the multiset rewriting approach can be cumbersome. Two alternatives would be to add support for a process calculus as an input language or to provide translators from source-code, e.g., in *C* or a functional language, to our dialect of multiset rewriting.

Our approach is currently limited to trace properties. To widen the scope to privacy-type properties, it would be interesting to investigate if our techniques can be applied to equivalences such as observational equivalence or trace equivalence.

Impossibility Results

We have initiated the systematic study of impossibility results for secret establishment protocols. We have presented three different kinds of results. First, we gave a formal model for proving impossibility results for secret establishment for cryptographic operations described by equational theories. We used this model to give the first formal impossibility proof for symmetric encryption in the symbolic setting. Afterwards, we generalized this result to necessary and sufficient conditions for the impossibility of secret establishment for any subterm-convergent theory. This directly yields a decision procedure and constitutes a first step towards machine assisted analysis of impossibility. Second, we adapt algebraic methods to prove the impossibility of secret establishment for group theories including XOR and abelian groups. Finally, we proved a combination result that enables modular impossibility proofs.

As future work, we plan to investigate other equational theories where the impossibility question is still open. We would also like to investigate other security properties, such as authentication and perfect forward secrecy, as well as different adversary models. Another interesting question is whether our labeling technique and decision procedure could be used for protocol synthesis. Finally, we would like to investigate interpretations of our results in a computational setting. Here, the cryptographic faithfulness [24] of equational theories with respect to computational algebras seems like a good starting point.

Physical Protocols

We have presented a formal approach to modeling and verifying security protocols involving physical properties. Our model captures dense time, agent locations, and physical properties of the communication network. To our knowledge, this is the first formal model that combines these aspects. This model has enabled us to formalize protocols, security properties, and environmental assumptions that are not amenable to formal analysis using other existing approaches. We have used our model to verify security properties of four different protocols and showed that our model captures relay attacks by distributed intruders and distance hijacking attacks.

There are several directions for future work. First, we could specialize our model to the analysis of distance bounding protocols and prove that attacks in our model imply the existence of attacks in simpler models with fewer details. Given such a simplified model, it might be possible to perform automated analysis of distance bounding protocols, for example with TAMARIN. Second, to extend the scope to distance bounding protocols similar to the Hancke-Kuhn [92] protocol, where the rapid bit-exchange phase uses precomputed answer-bits for the possible challenge-bits instead of XOR, we could search for equational theories that would allow us to model this. Third, we could extend our model to handle Terrorist fraud. Finally, it would be worthwhile to investigate if and how our model could be adapted to the computational setting.

Appendix A
Proofs for Chapter 3

In this appendix, we present the missing proofs from Chapter 3. Note that we perform the proofs with respect to the extension with bilinear pairings and AC operators.

A.1 Proofs for Verification Theory

We present proofs for properties of multiplication, of normal dependency graphs, and formulas.

A.1.1 Proofs for Multiplication

Conditions **P1**–**P6** from Definition 3.4 were introduced to restrict protocols from creating new products that can be extracted by the adversary. We will now precisely define the notion of "creating new products" and prove that protocol rules satisfy this requirement. To achieve this, we define the following notion of factors.

$$factors(t) = \begin{cases} factors(a) \cup factors(b) & \text{if } t = a{*}b \\ factors(s) & \text{if } t = s^{-1} \\ \{t, t^{-1}\} & \text{otherwise} \end{cases}$$

Since we cannot forbid all occurrences of products in rules, we first define a set of positions where no term can be extracted from. In the following, we use *exp* , *smult*, and *inv* to denote (infix versions of) the function symbols $\_\hat{}\_$, $[\_]\_$, and $\_^{-1}$ for exponentiation, scalar multiplication, and inversion.

Definition A.1. A position p is an *inaccesssible* in t if there is p' such that either $root(t|_{p'}) = exp$ and $p' \cdot [2]$ is above or equal to p or $root(t|_{p'}) = smult$ and $p' \cdot [1]$ is above or equal to p. We extend this definition to facts, sequences of facts, and multiset rewriting rules in the expected way. We say a position is *accessible* if it is not inaccessible.

Definition A.2. An *accessible product position* in a term t is an accessible position p in t such that $t|_p$ is a product. We use $app(t)$ to denote this set of positions. The *accessible factors* of a term t are then defined as $afactors(t) = \bigcup_{p \in app(t)} factors(t|_p)$. Analogously, we define the *accessible variable positions* p in t such that $t|_p \in \mathcal{V}_{msg}$ and denote them with $avp(t)$. The *accessible variables* of a term t are then defined as $avars(t) = \bigcup_{p \in avp(t)} \{t|_p\}$. We extend these notions to facts, sequences of facts, and multiset rewrite rules in the expected way.

We can now define the required condition on multiset rewriting rules that do not "construct accessible products". Note that since we want to allow unrestricted usage of exponentiation, scalar multiplication, bilinear maps, and inversion, the condition is slightly weaker and only ensures that such rules do not create accessible products with new factors. This is exactly the property we require to simplify the message deduction.

Definition A.3. A multiset rewriting rule $l \,-\![a]\!\!\rightarrow r$ is *factor-restricted* if for all $\downarrow_{\mathcal{RBP}_e}$-normal substitutions σ,

$$afactors((r\sigma)\downarrow_{\mathcal{RBP}_e}) \subseteq_{ACC} afactors((l\sigma)\downarrow_{\mathcal{RBP}_e}).$$

A protocol P is factor-restricted if all $l \,-\![a]\!\!\rightarrow r \in P$ are factor-restricted.

Since the definition of factor-restricted contains an universal quantification over all substitutions, it is hard to check in practice. We therefore prove that our syntactic restrictions on protocols imply factor-restricted.

Lemma A.4. *All protocols P are factor-restricted.*

Before we can prove this lemma, we have to prove some auxiliary results about the interaction of *avars*, *factors*, and *afactors* with normalization and instantiation.

Definition A.5. We define the set of *products* as

$$Prod = \{inv^k(a{*}b) | k \in \mathbb{N} \wedge a, b \in \mathcal{T}\}.$$

Lemma A.6. *For all terms t, either $t \in Prod$ and $factors(t)\downarrow_{\mathcal{RBP}_e} \subseteq_{ACC} afactors(t)\downarrow_{\mathcal{RBP}_e}$ or $t \notin Prod$ and $factors(t)\downarrow_{\mathcal{RBP}_e} \subseteq_{ACC} \{t, t^{-1}\}\downarrow_{\mathcal{RBP}_e}$. Hence $factors(t)\downarrow_{\mathcal{RBP}_e} \subseteq_{ACC} \{t, t^{-1}\}\downarrow_{\mathcal{RBP}_e} \cup afactors(t)\downarrow_{\mathcal{RBP}_e}$.*

Proof. Let t' such that $t = inv^k(t')$ and $root(t') \neq inv$. We perform a case distinction on $root(t')$. If $t = a{*}b$, then $t \in Prod$ and

$$factors(t) = factors(a) \cup factors(b) \subseteq afactors(t).$$

If $root(t') \notin \{*, inv\}$, then $t \notin Prod$ and

$$factors(t)\downarrow_{\mathcal{RBP}_e} =_{ACC} \{t', t'^{-1}\}\downarrow_{\mathcal{RBP}_e} =_{ACC} \{t, t^{-1}\}\downarrow_{\mathcal{RBP}_e}. \qquad \square$$

The next lemma charaterizes the new factors that can be introduced by a rewrite step.

Lemma A.7. *For all terms t and t' such that $t \rightarrow_{\mathcal{RBP}_e, ACC} t'$, it holds that*

$$factors(t')\downarrow_{\mathcal{RBP}_e} \subseteq_{ACC} factors(t)\downarrow_{\mathcal{RBP}_e} \cup afactors(t)\downarrow_{\mathcal{RBP}_e} \cup \{1\}.$$

Proof. We prove this by induction over terms. First, note that the base cases for variables and names hold since no rewrite rule is applicable.

- $t = inv(s)$: If t is rewritten below the root position, then $t' = inv(s')$ for some s' such that $s \rightarrow_{\mathcal{RBP}_e, ACC} s'$ and we can conclude the case as follows.

$$\begin{array}{lll} & factors(t')\downarrow_{\mathcal{RBP}_e} & \\ =_{ACC} & factors(s')\downarrow_{\mathcal{RBP}_e} & [\,\text{since } t' = inv(s')\,] \\ \subseteq_{ACC} & factors(s)\downarrow_{\mathcal{RBP}_e} \cup afactors(s)\downarrow_{\mathcal{RBP}_e} \cup \{1\} & [\,\text{by IH}\,] \\ =_{ACC} & factors(t)\downarrow_{\mathcal{RBP}_e} \cup afactors(t)\downarrow_{\mathcal{RBP}_e} \cup \{1\} & [\,\text{since } t = inv(s)\,] \end{array}$$

If t is rewritten at the root position, then we have to consider the applicable rules from $\mathcal{RBP}_e$, which are all included in $\mathcal{RDH}$. For all rewriting steps $t \rightarrow_{\mathcal{RBP}_e,ACC} t'$ that apply one of these rules at the root position, it is easy to see that $\mathit{factors}(t') =_{ACC} \mathit{factors}(t)$.

- $t = g(s_1, \ldots s_k)$ for $g \notin \{\mathit{inv}, *, \sharp, \mathit{exp}, \mathit{smult}\}$: If t is rewritten below the root position, then $\mathit{root}(t') = g$, $t, t' \notin \mathit{Prod}$ and therefore
$$\begin{aligned} & \mathit{factors}(t')\downarrow_{\mathcal{RBP}_e} \\ =_{ACC} & \{t', t'^{-1}\}\downarrow_{\mathcal{RBP}_e} \\ =_{ACC} & \{t, t^{-1}\}\downarrow_{\mathcal{RBP}_e} \\ =_{ACC} & \mathit{factors}(t)\downarrow_{\mathcal{RBP}_e}. \end{aligned}$$
If t is rewritten at the root position, then there is a substitution σ and a rewriting rule $l \rightarrow r$ from $\mathcal{ST}$ such that $t =_{ACC} l\sigma$ and $t' =_{ACC} r\sigma$. We have to consider two cases. First, r is a ground term built over $\Sigma_{\mathcal{ST}}$. Then t, $t' \notin \mathit{Prod}$ and the same reasoning as in the previous case applies. Second, r is a proper subterm of l and $t' = t|_p$ for some accessible position p. To see that p is accessible, first note that no position strictly above p is a variable position in l, l does not contain $*$ or $\sharp$, and p is a valid position in l. Hence, all positions $\tilde{p}$ strictly above p are valid positions in l and $\mathit{root}(t|_{\tilde{p}}) = \mathit{root}(l|_{\tilde{p}}) \notin \{\mathit{exp}, \mathit{smult}\}$ for all such positions and therefore p accessible. If $t \in \mathit{Prod}$, then $\mathit{factors}(t')\downarrow_{\mathcal{RBP}_e} \subseteq_{ACC} \mathit{afactors}(t')\downarrow_{\mathcal{RBP}_e} \subseteq_{ACC} \mathit{afactors}(t)\downarrow_{\mathcal{RBP}_e}$ by Lemma A.6. If $t \notin \mathit{Prod}$, then
$$\mathit{factors}(t')\downarrow_{\mathcal{RBP}_e} =_{ACC} \{t', t'^{-1}\}\downarrow_{\mathcal{RBP}_e} =_{ACC} \{t, t^{-1}\}\downarrow_{\mathcal{RBP}_e} =_{ACC} \mathit{factors}(t)\downarrow_{\mathcal{RBP}_e}.$$
- $t = a \sharp b$: Since there is no rewriting rule applicable at the root position, we can assume without loss of generality that $t' = a' \sharp b$ such that $a \rightarrow_{\mathcal{RBP}_e,ACC} a'$. Then, the reasoning is analogous to the previous case where t is rewritten below the root position.
- $t = \hat{e}(a, b)$: If t is rewritten below the root position, then the reasoning is analogous to the previous cases. Otherwise, $a = [u]v$ and $t' = \hat{e}(v, b) \hat{\ } u$. Then
$$\begin{aligned} & \mathit{factors}(t')\downarrow_{\mathcal{RBP}_e} \\ =_{ACC} & \{t', t'^{-1}\}\downarrow_{\mathcal{RBP}_e} \\ =_{ACC} & \{t, t^{-1}\}\downarrow_{\mathcal{RBP}_e} \\ =_{ACC} & \mathit{factors}(t)\downarrow_{\mathcal{RBP}_e}. \end{aligned}$$
- $t = a \hat{\ } b$: There are three possibilities, either t is rewritten below the root position, at the root position with the rule $(x \hat{\ } y) \hat{\ } z \rightarrow x \hat{\ } (y * z)$, or at the root position with the rule $x \hat{\ } 1 \rightarrow x$. For the the first two cases and the third case with a not a product, $t' \notin \mathit{Prod}$ and
$$\begin{aligned} & \mathit{factors}(t')\downarrow_{\mathcal{RBP}_e} \\ =_{ACC} & \{t', t'^{-1}\}\downarrow_{\mathcal{RBP}_e} \\ =_{ACC} & \{t, t^{-1}\}\downarrow_{\mathcal{RBP}_e} \\ =_{ACC} & \mathit{factors}(t)\downarrow_{\mathcal{RBP}_e}. \end{aligned}$$
For the third case with $a \in \mathit{Prod}$, we have $t' = a$ and
$$\mathit{factors}(a)\downarrow_{\mathcal{RBP}_e} \subseteq_{ACC} \mathit{afactors}(a)\downarrow_{\mathcal{RBP}_e} \subseteq_{ACC} \mathit{afactors}(t)\downarrow_{\mathcal{RBP}_e}$$
since a accessible in t.

- $t = [b]a$: Analogous to the previous case.
- $t = s_1 {*} s_2$: If t is rewritten below the root position, then we assume without loss of generality that $t' =_{ACC} s_1' {*} s_2$ for some s_1' with $s_1 \rightarrow_{\mathcal{RBP}_e, ACC} s_1'$. Then we can conclude the case as follows.

$$\begin{array}{lll} & \mathit{factors}(t')\downarrow_{\mathcal{RBP}_e} & \\ =_{ACC} & \mathit{factors}(s_1')\downarrow_{\mathcal{RBP}_e} \cup \mathit{factors}(s_2)\downarrow_{\mathcal{RBP}_e} & [\, t' = s_1' {*} s_2 \,] \\ \subseteq_{ACC} & \mathit{factors}(s_1)\downarrow_{\mathcal{RBP}_e} \cup \mathit{afactors}(s_1)\downarrow_{\mathcal{RBP}_e} \cup \{1\} \cup \mathit{factors}(s_2)\downarrow_{\mathcal{RBP}_e} & [\, \text{by IH} \,] \\ \subseteq_{ACC} & \mathit{factors}(t)\downarrow_{\mathcal{RBP}_e} \cup \mathit{afactors}(t)\downarrow_{\mathcal{RBP}_e} \cup \{1\} & [\, t = s_1 {*} s_2 \,] \end{array}$$

If t is rewritten at the root position, then we have to consider the applicable rules in $\mathcal{RDH}$. For all rewriting steps $t \rightarrow_{\mathcal{RBP}_e, ACC} t'$ that apply one of these rules at the root position, it is easy to see that $\mathit{factors}(t') \subseteq_{ACC} \mathit{factors}(t) \cup \{1\}$. □

Lemma A.8. *For all terms t and t' such that $t \rightarrow_{\mathcal{RBP}_e, ACC} t'$, it holds that*

$$\mathit{afactors}(t')\downarrow_{\mathcal{RBP}_e} \subseteq_{ACC} \mathit{afactors}(t)\downarrow_{\mathcal{RBP}_e} \cup \{1\}.$$

Proof. We prove this by induction over terms. First, note that the base cases for variables and names hold since no rewrite rule is applicable.

- $t = \mathit{inv}(s)$: If t is rewritten below the root position, then $t' = \mathit{inv}(s')$ for some s' such that $s \rightarrow_{\mathcal{RBP}_e, ACC} s'$ and we can conclude the case as follows.

$$\begin{array}{lll} & \mathit{afactors}(t')\downarrow_{\mathcal{RBP}_e} & \\ =_{ACC} & \mathit{afactors}(s')\downarrow_{\mathcal{RBP}_e} & [\, \text{since } t' = \mathit{inv}(s') \,] \\ \subseteq_{ACC} & \mathit{afactors}(s)\downarrow_{\mathcal{RBP}_e} \cup \{1\} & [\, \text{by IH} \,] \\ =_{ACC} & \mathit{afactors}(t)\downarrow_{\mathcal{RBP}_e} \cup \{1\} & [\, \text{since } t = \mathit{inv}(s) \,] \end{array}$$

  If t is rewritten at the root position, then we have to consider the applicable rules from $\mathcal{RBP}_e$, which are all included in $\mathcal{RDH}$. For all rewriting steps $t \rightarrow_{\mathcal{RBP}_e, ACC} t'$ that apply one of these rules at the root position, it is easy to see that $\mathit{afactors}(t') =_{ACC} \mathit{afactors}(t)$.
- $t = g(s_1, \ldots s_k)$ for $g \notin \{\mathit{inv}, *, \sharp, \mathit{exp}, \mathit{smult}\}$: If t is rewritten below the root position, then $t' = g(s_1', \ldots, s_k')$ such that there is i with $s_i \rightarrow_{\mathcal{RBP}_e, ACC} s_i'$ and $s_j =_{ACC} s_j'$ for all $i \neq j$. We can prove this case as follows.

$$\begin{array}{lll} & \mathit{afactors}(t')\downarrow_{\mathcal{RBP}_e} & \\ =_{ACC} & \displaystyle\bigcup_{i=1}^{k} \mathit{afactors}(s_i')\downarrow_{\mathcal{RBP}_e} & [\, \text{since } t' = g(s_1', \ldots, s_k') \,] \\ =_{ACC} & \displaystyle\bigcup_{i=1}^{k} \mathit{afactors}(s_i)\downarrow_{\mathcal{RBP}_e} \cup \{1\} & [\, \text{by IH} \,] \\ =_{ACC} & \mathit{afactors}(t)\downarrow_{\mathcal{RDH}_e} \cup \{1\} & [\, \text{since } t = g(s_1, \ldots, s_k) \,] \end{array}$$

  If t is rewritten at the root position, then then there is a substitution σ and a rewriting rule $l \rightarrow r$ from $\mathcal{ST}$ such that $t =_{ACC} l\sigma$ and $t' =_{ACC} r\sigma$. We distinguish two cases. If r is ground, then $\mathit{afactors}(t') = \emptyset$. Otherwise, r is a proper subterm of l and $t' = t|_p$ for an accessible position p in t such that no position strictly above p has the root symbol $*$ in t. Hence $\mathit{afactors}(t')\downarrow_{\mathcal{RBP}_e} \subseteq_{ACC} \mathit{afactors}(t)\downarrow_{\mathcal{RBP}_e} \cup \{1\}$.

- $t = a\sharp b$: Since there is no rewriting rule applicable at the root position, we can assume without loss of generality that $t' = a'\sharp b$ such that $a \rightarrow_{\mathcal{RBP}_e, ACC} a'$. Then, the reasoning is analogous to the previous case where t is rewritten below the root position.
- $t = \hat{e}(a, b)$: If t is rewritten below the root position, then the reasoning is analogous to the previous cases. Otherwise, $a = [u]v$ and $t' = \hat{e}(v, b)\hat{}u$. Then

$$\begin{array}{lll} & \mathit{afactors}(t')\downarrow_{\mathcal{RBP}_e} & \\ =_{ACC} & \mathit{afactors}(v)\downarrow_{\mathcal{RBP}_e} \cup \mathit{afactors}(b)\downarrow_{\mathcal{RBP}_e} & [\text{since } t' = \hat{e}(v,b)\hat{}u \text{ and } u \text{ not accessible}] \\ =_{ACC} & \mathit{afactors}(t)\downarrow_{\mathcal{RDH}_e}. & [\text{since } t = \hat{e}([u]v, b) \text{ and } u \text{ not accessible}] \end{array}$$

- $t = a\hat{}b$: If t is rewritten below the root position, then we distinguish two cases. First, if b is rewritten, then $\mathit{afactors}(t) = \mathit{afactors}(a) = \mathit{afactors}(t')$ implies the desired inclusion. Second, if $t' = a'\hat{}b$, then we can apply the induction hypothesis similar to the previous cases.

  If it is rewritten at the root position, then it is either rewritten with the rule $x\hat{}1 \rightarrow x$ or the rule $(x\hat{}y)\hat{}z \rightarrow x\hat{}(y{*}z)$. In the first case, $\mathit{afactors}(t) = \mathit{afactors}(a) = \mathit{afactors}(t')$ since $t' = a$. In the second case, $a = g\hat{}c$ for some g and c and $t' = g\hat{}(c{*}b)$. We can conclude this case as follows.

$$\begin{array}{lll} & \mathit{afactors}(t')\downarrow_{\mathcal{RBP}_e} & \\ =_{ACC} & \mathit{afactors}(g)\downarrow_{\mathcal{RBP}_e} & [\text{since } c{*}b \text{ not accessible in } g\hat{}(c{*}b)] \\ =_{ACC} & \mathit{afactors}(t)\downarrow_{\mathcal{RDH}_e}. & [\text{since } c \text{ and } b \text{ not accessible in } (g\hat{}c)\hat{}b] \end{array}$$

- $t = [b]a$: Analogous to the previous case.
- $t = s_1{*}s_2$: If t is rewritten below the root position, then we assume without loss of generality that $t' =_{ACC} s_1'{*}s_2$ for some s_1' with $s_1 \rightarrow_{\mathcal{RBP}_e, ACC} s_1'$. Then we can conclude the case as follows.

$$\begin{array}{lll} & \mathit{afactors}(t')\downarrow_{\mathcal{RBP}_e} & \\ =_{ACC} & \mathit{afactors}(s_1')\downarrow_{\mathcal{RBP}_e} \cup \mathit{afactors}(s_2)\downarrow_{\mathcal{RBP}_e} \cup & [t' = s_1'{*}s_2] \\ & \mathit{factors}(s_1')\downarrow_{\mathcal{RBP}_e} \cup \mathit{factors}(s_2)\downarrow_{\mathcal{RBP}_e} & \\ \subseteq_{ACC} & \mathit{afactors}(s_1)\downarrow_{\mathcal{RBP}_e} \cup \mathit{afactors}(s_2)\downarrow_{\mathcal{RBP}_e} \cup & [\text{by IH}] \\ & \mathit{factors}(s_1')\downarrow_{\mathcal{RBP}_e} \cup \mathit{factors}(s_2)\downarrow_{\mathcal{RBP}_e} \cup \{1\} & \\ \subseteq_{ACC} & \mathit{afactors}(s_1)\downarrow_{\mathcal{RBP}_e} \cup \mathit{afactors}(s_2)\downarrow_{\mathcal{RBP}_e} \cup & [\text{by Lemma A.7}] \\ & \mathit{factors}(s_1)\downarrow_{\mathcal{RBP}_e} \cup \mathit{factors}(s_2)\downarrow_{\mathcal{RBP}_e} \cup \{1\} & \\ \subseteq_{ACC} & \mathit{afactors}(t)\downarrow_{\mathcal{RBP}_e} \cup \{1\} & [t = s_1{*}s_2] \end{array}$$

  If t is rewritten at the root position, then we have to consider the applicable rules in $\mathcal{RDH}$. For all rewriting steps $t \rightarrow_{\mathcal{RBP}_e, ACC} t'$ that apply one of these rules at the root position. It is easy to see that $\mathit{afactors}(t') \subseteq_{ACC} \mathit{afactors}(t) \cup \{1\}$. □

Lemma A.9. *For all terms t, it holds that*

$$\mathit{afactors}(t\downarrow_{\mathcal{RBP}_e}) \subseteq_{ACC} \mathit{afactors}(t)\downarrow_{\mathcal{RBP}_e}.$$

Proof. Directly follows from t $\downarrow_{\mathcal{RBP}_e}$-normal or $t \rightarrow^{+}_{\mathcal{RBP}_e, ACC} t\downarrow_{\mathcal{RBP}_e}$ using Lemma A.8 and the fact that for a product s that is $\downarrow_{\mathcal{RBP}_e}$-normal $1 \notin \mathit{factors}(s)$. □

Lemma A.10. *For all terms t, it holds that*

$$\mathit{factors}(t{\downarrow_{\mathcal{RBP}_e}}) \subseteq_{ACC} \mathit{factors}(t){\downarrow_{\mathcal{RBP}_e}} \cup \mathit{afactors}(t){\downarrow_{\mathcal{RBP}_e}} \cup \{1\}.$$

Proof. We prove this by induction over the length of the rewriting sequence

$$t_1 \to_{\mathcal{RBP}_e,ACC} \dots \to_{\mathcal{RBP}_e,ACC} t_k$$

where t_k ${\downarrow_{\mathcal{RBP}_e}}$-normal.

The base case clearly holds and assuming the hypothesis for $k-1$, we conclude with

$$\begin{array}{ll} & \mathit{factors}(t_k) \\ \subseteq_{ACC} & \mathit{factors}(t_2){\downarrow_{\mathcal{RBP}_e}} \cup \mathit{afactors}(t_2){\downarrow_{\mathcal{RBP}_e}} \cup \{1\} \quad [\text{by IH}] \\ \subseteq_{ACC} & \mathit{factors}(t_1){\downarrow_{\mathcal{RBP}_e}} \cup \mathit{afactors}(t_1){\downarrow_{\mathcal{RBP}_e}} \cup \{1\}. \quad [\text{Lemma A.7 and Lemma } A.8] \end{array}$$

□

Lemma A.11. *For all terms t and ${\downarrow_{\mathcal{RBP}_e}}$–normal substitutions σ, it holds that*

$$\mathit{afactors}(t\sigma){\downarrow_{\mathcal{RBP}_e}} =_{ACC} (\mathit{afactors}(t)\sigma \setminus \mathit{Prod}){\downarrow_{\mathcal{RBP}_e}} \cup \left(\bigcup_{x \in \mathit{avars}(t)} \mathit{afactors}(x\sigma){\downarrow_{\mathcal{RBP}_e}} \right).$$

Proof. Let σ arbitrary and

$$F_t = (\mathit{afactors}(t)\sigma \setminus \mathit{Prod}){\downarrow_{\mathcal{RBP}_e}} \cup \left(\bigcup_{x \in \mathit{avars}(t)} \mathit{afactors}(x\sigma){\downarrow_{\mathcal{RBP}_e}} \right).$$

Note that

$$\begin{array}{rl} & \bigcup_{i=1}^{k} \mathit{afactors}(s_i) =_{ACC} \mathit{afactors}(t) \quad \text{and} \\ & \bigcup_{i=1}^{k} \mathit{avars}(s_i) =_{ACC} \mathit{avars}(t) \\ \text{implies} & \bigcup_{i=1}^{k} F_{s_i} =_{ACC} F_t \qquad (1) \end{array}$$

We prove $\mathit{afactors}(t\sigma){\downarrow_{\mathcal{RBP}_e}} =_{ACC} F_t$ by induction on t.

- $t = x$ for $x \in \mathcal{V}_{msg}$: Since $\mathit{afactors}(x) = \emptyset$ and $\mathit{avars}(x) = \{x\}$, we have $F_x = \mathit{afactors}(x\sigma){\downarrow_{\mathcal{RBP}_e}}$.
- $t = x$ for $x \in \mathcal{V}_{pub} \cup \mathcal{V}_{fr}$: Both sides are equal to the empty set since σ well-sorted.
- $t \in \mathsf{PN} \cup \mathsf{FN}$: Both sides are equal to the empty set.
- $t = \mathit{inv}(s)$:

$$\begin{array}{rll} \mathit{afactors}(t\sigma){\downarrow_{\mathcal{RBP}_e}} =_{ACC} & \mathit{afactors}(s\sigma){\downarrow_{\mathcal{RBP}_e}} & \\ =_{ACC} & F_s & [\text{IH}] \\ =_{ACC} & F_t\,. & [(1)] \end{array}$$

- $t = g(s_1, ..., s_k)$ for $g \notin \{*, exp, inv, smult\}$:

$$\begin{aligned} afactors(t\sigma){\downarrow_{\mathcal{RBP}_e}} &=_{ACC} \left(\bigcup_{i=1}^{k} afactors(s_i\sigma){\downarrow_{\mathcal{RBP}_e}}\right) \\ &=_{ACC} \left(\bigcup_{i=1}^{k} F_{s_i}\right) && [\,\text{IH}\,] \\ &=_{ACC} F_t. && [\,(1)\,] \end{aligned}$$

- $t = a\hat{\ }b$:

$$\begin{aligned} afactors(t\sigma){\downarrow_{\mathcal{RBP}_e}} &=_{ACC} afactors(a\sigma){\downarrow_{\mathcal{RBP}_e}} \\ &=_{ACC} F_a && [\,\text{IH}\,] \\ &=_{ACC} F_t. && [\,(1)\,] \end{aligned}$$

- $t = [b]a$: Analogous to previous case.
- $t = s_1 * s_2$: Let $J \subseteq \{1, 2\}$ such that $j \in J$ iff $s_i\sigma \notin Prod$. We first show that

$$F_{s_1} \cup F_{s_2} \cup \left(\bigcup_{j \in J} \{s_j\sigma, (s_j\sigma)^{-1}\}{\downarrow_{\mathcal{RBP}_e}}\right) =_{ACC} F_t \qquad (2)$$

  - "$\subseteq$": If $J = \emptyset$, then we are done since $avars(s_1) \cup avars(s_2) \subseteq avars(t)$, $afactors(s_1) \cup afactors(s_2) \subseteq afactors(t)$ and therefore $F_{s_1} \cup F_{s_2} \subseteq F_t$. Now let $j \in J$ arbitrary. Then $s_j\sigma \notin Prod$ and hence $s_j \notin Prod$. Then there are k and u such that $s_j = inv^k(u)$ and $root(u) \notin \{inv, *\}$. Note that $u\sigma \notin Prod$ and $u^{-1}\sigma \notin Prod$. Therefore $\{u, u^{-1}\} = factors(s_j) \subseteq afactors(t)$ and

$$\begin{aligned} \{s_j\sigma, (s_j\sigma)^{-1}\}{\downarrow_{\mathcal{RBP}_e}} &=_{ACC} \{u\sigma, u^{-1}\sigma\}{\downarrow_{\mathcal{RBP}_e}} \\ &\subseteq_{ACC} (afactors(t)\sigma \setminus Prod){\downarrow_{\mathcal{RBP}_e}} \\ &\subseteq_{ACC} F_t\,. \end{aligned}$$

  - "$\supseteq$": Since

$$\begin{aligned} F_t =_{ACC}\ & (afactors(t)\sigma \setminus Prod){\downarrow_{\mathcal{RBP}_e}} \cup \left(\bigcup_{x \in avars(t)} afactors(x\sigma){\downarrow_{\mathcal{RBP}_e}}\right) \\ =_{ACC}\ & \left(\bigcup_{i=1}^{2} (afactors(s_i)\sigma \setminus Prod){\downarrow_{\mathcal{RBP}_e}}\right) \\ & \cup \left(\bigcup_{i=1}^{2} (factors(s_i)\sigma \setminus Prod){\downarrow_{\mathcal{RBP}_e}}\right) \\ & \cup \left(\bigcup_{i=1}^{2} \left(\bigcup_{x \in avars(s_i)} afactors(x\sigma){\downarrow_{\mathcal{RBP}_e}}\right)\right), \end{aligned}$$

  we have to show that for $i \in \{1, 2\}$

$$(factors(s_i)\sigma \setminus Prod){\downarrow_{\mathcal{RBP}_e}} \subseteq_{ACC} F_{s_1} \cup F_{s_2} \cup \left(\bigcup_{j \in J} \{s_j\sigma, (s_j\sigma)^{-1}\}{\downarrow_{\mathcal{RBP}_e}}\right).$$

Let $i \in \{1,\, 2\}$ arbitrary. We distinguish two cases. First, if $s_i \in \mathit{Prod}$, then $\mathit{factors}(s_i) \subseteq \mathit{afactors}(s_i)$ and hence $(\mathit{factors}(s_i)\sigma \setminus \mathit{Prod}){\downarrow_{\mathcal{RBP}_e}} \subseteq_{ACC} F_{s_i}$.

Second, if $s_i \notin \mathit{Prod}$, then there are k and u such that $s_i = \mathit{inv}^k(u)$ and $\mathit{root}(u) \notin \{*, \mathit{inv}\}$. Hence $\mathit{factors}(s_i) = \{u,\, u^{-1}\}$ and either $u\sigma \in \mathit{Prod}$ or not. If $u\sigma \in \mathit{Prod}$, then $u^{-1}\sigma \in \mathit{Prod}$ and hence $(\mathit{factors}(s_i)\sigma \setminus \mathit{Prod}){\downarrow_{\mathcal{RBP}_e}} = \emptyset$. If $u\sigma \notin \mathit{Prod}$, then $s_i\sigma \notin \mathit{Prod}$ and hence $i \in J$. Then we can conclude the case with

$$\begin{aligned}
(\mathit{factors}(s_i)\sigma \setminus \mathit{Prod}){\downarrow_{\mathcal{RBP}_e}} &=_{ACC} \{u\sigma, u^{-1}\sigma\}{\downarrow_{\mathcal{RDH}_e}}\\
&=_{ACC} \{s_i\sigma, (s_i\sigma)^{-1}\}{\downarrow_{\mathcal{RDH}_e}}\\
&=_{ACC} F_{s_1} \cup F_{s_2} \cup \left(\bigcup_{j \in J} \{s_j\sigma, (s_j\sigma)^{-1}\}{\downarrow_{\mathcal{RBP}_e}} \right) .
\end{aligned}$$

Using the previous result, we can now concluded the proof as follows.

$$\begin{aligned}
& \mathit{afactors}(t\sigma){\downarrow_{\mathcal{RBP}_e}} \\
=_{ACC}\ & \bigcup_{i=1}^{2} (\mathit{afactors}(s_i\sigma){\downarrow_{\mathcal{RBP}_e}} \cup \mathit{factors}(s_i\sigma){\downarrow_{\mathcal{RBP}_e}}) && [\, t = s_1 * s_2 \,]\\
\subseteq_{ACC}\ & \bigcup_{i=1}^{2} (\mathit{afactors}(s_i\sigma){\downarrow_{\mathcal{RBP}_e}}) \cup \left(\bigcup_{j \in J} \{s_j\sigma, (s_j\sigma)^{-1}\}{\downarrow_{\mathcal{RBP}_e}} \right) && [\text{ by Lemma A.6}\,]\\
\subseteq_{ACC}\ & F_{s_1} \cup F_{s_2} \cup \left(\bigcup_{j \in J} \{s_j\sigma, (s_j\sigma)^{-1}\}{\downarrow_{\mathcal{RBP}_e}} \right) && [\text{by IH}\,]\\
\subseteq_{ACC}\ & F_t && [\,(2)\,]
\end{aligned}$$

□

Proof. (of Lemma A.4) Let $l \dashv[a]\mapsto r$ be an arbitrary protocol rule and σ an arbitrary ${\downarrow_{\mathcal{RBP}_e}}$-normal substitution. Then, the following holds and $l \dashv[a]\mapsto r$ is therefore factor-restricted.

$$\begin{aligned}
& \mathit{afactors}((r\sigma){\downarrow_{\mathcal{RBP}_e}}) \\
\subseteq_{ACC}\ & \mathit{afactors}(r\sigma){\downarrow_{\mathcal{RBP}_e}} && [\text{by Lemma A.9}\,]\\
=_{ACC}\ & (\mathit{afactors}(r)\sigma \setminus \mathit{Prod}){\downarrow_{\mathcal{RBP}_e}} \cup \left(\bigcup_{x \in \mathit{avars}(r)} \mathit{afactors}(x\sigma){\downarrow_{\mathcal{RBP}_e}} \right) && [\text{by Lemma A.11}\,]\\
\subseteq_{ACC}\ & (\mathit{afactors}(l)\sigma \setminus \mathit{Prod}){\downarrow_{\mathcal{RBP}_e}} \cup \left(\bigcup_{x \in \mathit{avars}(r)} \mathit{afactors}(x\sigma){\downarrow_{\mathcal{RBP}_e}} \right) && [\text{no } * \text{ in } l \dashv[a]\mapsto r\,]\\
\subseteq_{ACC}\ & (\mathit{afactors}(l)\sigma \setminus \mathit{Prod}){\downarrow_{\mathcal{RBP}_e}} \cup \left(\bigcup_{x \in \mathit{avars}(l)} \mathit{afactors}(x\sigma){\downarrow_{\mathcal{RBP}_e}} \right) && [\,(*)\,]\\
=_{ACC}\ & \mathit{afactors}(l\sigma){\downarrow_{\mathcal{RBP}_e}} && [\text{by Lemma A.11}\,]\\
=_{ACC}\ & \mathit{afactors}((l\sigma){\downarrow_{\mathcal{RBP}_e}}) && [\text{ since } (l\sigma){\downarrow_{\mathcal{RBP}_e}} = l\sigma\,]
\end{aligned}$$

(*) Since $l \dashv[a]\mapsto r$ is a protocol rule, it holds that $\mathit{vars}(r) \subseteq \mathit{vars}(l) \cup \mathcal{V}_{pub}$ and l does not contain smult or exp , or $l \dashv[a]\mapsto r$ is an instance of such a rule. Then, $\mathit{avars}(r) \subseteq \mathit{vars}(l) \cap \mathcal{V}_{msg} = \mathit{avars}(l)$ since l does not contain smult or exp . □

A.1.2 Proofs for Normal Dependency Graphs

In this section we prove an extended version of Lemma 3.19 for bilinear pairing and AC operators.

Lemma A.12. *For all protocols P,*

$$\{\overline{trace(dg)} \mid dg \in dgraphs_{AC}(\lceil P \cup MD \rceil^{\mathcal{RBP}_e}_{insts}) \wedge dg \downarrow_{\mathcal{RBP}_e}\text{-}normal\} = \overline{trace(ndgraphs(P))}.$$

Definition A.13. We define the *known messages* of a dependency graph dg as

$$known(dg) = \{m \mid \text{exists conclusion fact } \mathsf{K}(m) \text{ in } dg\}.$$

We define the *d-known messages* of a normal dependency graph ndg as

$$known^{d}(ndg) = \{m \mid \text{exists conclusion fact } \mathsf{K}^{d}(m) \text{ in } ndg\}$$

for $d \in \{\Uparrow, \Downarrow^{\mathsf{d}}, \Downarrow^{\mathsf{e}}\}$.

We define the *known messages* for a normal dependency graph ndg as

$$known^{\Updownarrow}(ndg) = known^{\Uparrow}(ndg) \cup known^{\Downarrow^{\mathsf{d}}}(ndg) \cup known^{\Downarrow^{\mathsf{e}}}(ndg).$$

We define the *available state-conclusions of a dependency graph dg* as

$$stfacts(dg) = \{f \in cfacts(dg) | \forall m d. f \neq \mathsf{K}(m) \wedge f \neq \mathsf{K}^{d}(m)\}$$

where $cfacts(dg)$ denotes the consumable facts in dg. We define the *created messages* of a dependency graph dg as

$$created(dg) = \{n \mid \text{exists conclusion fact } \mathsf{Fr}(n) \text{ in } dg\}.$$

A normal dependency graph $ndg' = (I', D')$ is a *deduction extension* of $ndg = (I, D)$ if I is a prefix of I', $D \subseteq D'$, $trace(ndg) = trace(ndg')$, $stfacts(ndg) = stfacts(ndg')$, and $created(ndg) = created(ndg')$. If there is a deduction extension such that a message m is known, we write m is *deducible*.

In the following, we use F$\Downarrow$ and F$\Uparrow$ to denote the construction and deconstruction rules for the corresponding function symbols. For example, we use FST$\Downarrow$ to denote the deconstruction rule for fst.

Lemma A.14. *For all $ndg \in ndgraphs(P)$, conclusions (i, u) in ndg with conclusion fact f and terms $t \in afactors(f)$, there is a conclusion (j, v) in ndg with $j < i$ and conclusion fact $\mathsf{K}^{d}(m)$ such that $m \in_{ACC} \{t, (t^{-1})\downarrow_{\mathcal{RBP}_e}\}$*

Proof. We prove by induction on normal dependency graphs that the property holds. The property obviously holds for $([\,], \emptyset)$. Let $ndg = (I, D) \in ndgraphs(P)$ arbitrary, $l \text{-}[a]\!\!\rightarrow r$ $\downarrow_{\mathcal{RBP}_e}$-normal, and D' such that $ndg' = (I \cdot l \text{-}[a]\!\!\rightarrow r, D \uplus D') \in ndgraphs(P)$. We perform a case distinction on $l \text{-}[a]\!\!\rightarrow r$.

- If $l \text{-}[a]\!\!\rightarrow r \in ginsts_{ACC}(\text{FRESH})$, then there is nothing to show since $f = \mathsf{Fr}(n)$ for a fresh name n and $factors(n) = \emptyset$.

- If $l \dashv\!\![a]\!\!\mapsto r \in \mathit{ginsts}_{ACC}(\lceil P\rceil^{\mathcal{RBP}_e}_{\mathit{insts}})$, then there is a substitution σ that is grounding for some $l' \dashv\!\![a']\!\!\mapsto r'$ in P such that $l \dashv\!\![\,a\,]\!\!\mapsto r =_{ACC} ((l' \dashv\!\![a']\!\!\mapsto r')\sigma){\downarrow_{\mathcal{RBP}_e}}$. Since P is factor-restricted,

$$\begin{aligned} \mathit{afactors}(r) &=_{ACC} \mathit{afactors}((r'\sigma){\downarrow_{\mathcal{RBP}_e}}) \\ &\subseteq_{ACC} \mathit{afactors}((l'\sigma){\downarrow_{\mathcal{RBP}_e}}) =_{ACC} \mathit{afactors}(l). \end{aligned}$$

  Hence, for all $j \in \mathit{idx}(r)$ and $t \in \mathit{afactors}(r_j)$, there is $k \in \mathit{idx}(l)$ such that $t \in \mathit{afactors}(l_k)$. Because the dependency must be satisfied by some conclusion in *ndg*, there is a conclusion c with a conclusion fact that is equal to l_k. We can therefore use the induction hypothesis

- If $l \dashv\!\![a]\!\!\mapsto r \in \mathit{ginsts}_{ACC}(\mathit{ND})$, then all rules except for the multiplication rule, Send, Recv, Coerce, the construction rules for fresh and public names, and the multiplication rules are of the form $[F_1(m_1), ..., F_k(m_k)] \dashv\!\![]\!\!\mapsto [F(f(m_1, ..., m_k){\downarrow_{\mathcal{RBP}_e}})]$ with $f \neq *$ and m_i ${\downarrow_{\mathcal{RBP}_e}}$-normal. We can use Lemma A.9 to obtain

$$\begin{aligned} \mathit{afactors}(f(m_1, ..., m_k){\downarrow_{\mathcal{RBP}_e}}) &\subseteq_{ACC} \mathit{afactors}(f(m_1, ..., m_k)){\downarrow_{\mathcal{RBP}_e}} \\ &\subseteq_{ACC} \bigcup_{i=1}^{k} \mathit{afactors}(m_i) \end{aligned}$$

  and use the induction hypothesis. For the multiplication case, note that the only new afactors are the inputs and their inverses and we can therefore use the induction hypothesis too. For the remaining rules, either $\mathit{afactors}(l) =_{ACC} \mathit{afactors}(r)$ or $\mathit{afactors}(r) = \emptyset$. □

Lemma A.15. *For all $ndg \in \mathit{ndgraphs}(P)$ and conclusion facts $\mathsf{K}^{\Downarrow y}(m)$, there is a deduction extension ndg' that contains a conclusion fact $\mathsf{K}^{\Uparrow}(m')$ with $m =_{ACC} m'$.*

Proof. We have to consider two cases. First, if there is a conclusion fact $\mathsf{K}^{\Uparrow}(m')$ with $m =_{ACC} m'$ in *ndg*, then we do not have to extend *ndg*. If there is no such conclusion c' in *ndg*, we use induction on m.

- If $\mathit{root}(m)$ is not invertible and not equal to $\sharp$, we can use Coerce directly. Note that because of **N2**, m is not a product.
- If $\mathit{root}(m) = \sharp$, then we can extract all $s \in \mathit{elems}(m)$ using the deconstruction rule for $\sharp$, convert the s by the induction hypothesis, and then use the construction rule $\sharp$ to build m.
- If $\mathit{root}(m)$ invertible, we can proceed as in the previous case. □

Lemma A.16. *Let $ndg \in \mathit{ndgraphs}(P)$, $m \in \mathit{known}{\Updownarrow}(ndg)$, m ${\downarrow_{\mathcal{RBP}_e}}$-normal, and $m \notin \mathit{Prod}$, then there is a deduction extension ndg' of ndg with $(m)^{-1}{\downarrow_{\mathcal{RBP}_e}} \in_{ACC} \mathit{known}{\Updownarrow}(ndg')$.*

Proof. We distinguish wether m is an inverse or not. If m is an inverse, then there is t such that $m = t^{-1}$ and t is not a product. Hence, $(m^{-1}){\downarrow_{\mathcal{RBP}_e}} =_{ACC} t$ and $t^{-1} \in_{ACC} \mathit{known}{\Downarrow}(ndg)$ or $t^{-1} \in_{ACC} \mathit{known}{\Uparrow}(ndg)$. In the first case, we can use Inv⇓ to deduce t since t is not a product. In the second case, t^{-1} must be the conclusion of an Inv⇑ rule, which implies that $t \in \mathit{known}{\Updownarrow}(ndg)$.

If m is no inverse, then $(m^{-1})\downarrow_{\mathcal{RBP}_e} =_{ACC} m^{-1}$, as m is no product. Due to Lemma A.15, there is an extension ndg' of ndg such that $m \in known\Uparrow(ndg')$. Hence, we can use Inv$\Uparrow$ to deduce m^{-1} □

Lemma A.17. *Let $ndg \in ndgraphs(P)$, t $\downarrow_{\mathcal{RBP}_e}$-normal, and for all $m \in factors(t)$, $m \in_{ACC} known\Updownarrow(ndg')$ or $(m^{-1})\downarrow_{\mathcal{RBP}_e} \in_{ACC} known\Updownarrow(ndg')$, then there is a deduction extension ndg' of ndg with $t \in_{ACC} known\Updownarrow(ndg')$.*

Proof. We prove the lemma by induction over t.

- $t \in \mathsf{FN} \cup \mathsf{PN}$: Since $factors(t) = \{t, t^{-1}\}$, we can use Lemma A.16 to obtain ndg'.
- $t = f(u_1, \ldots, u_k)$ for $f \notin \{inv, *\}$: The reasoning is the same as in the previous case.
- $t = u^{-1}$: Since $factors(t) = factors(u)$, we can use the induction hypothesis to obtain ndg' such that u is known. We can therefore use Lemma A.15 to deduce $\mathsf{K}^{\Uparrow}(u)$ (if required) and then deduce u^{-1} by applying Inv$\Uparrow$.
- $t = (u_1 * \ldots * u_k) * (u_{k+1} * \ldots * u_{k+l})^{-1}$: Since $factors(t) = \{u_i | 1 \leq i \leq k + l\}$, either u_i or u_i^{-1} is known in ndg for all i. If only u_i^{-1} is known, we can deduce u_i by Lemma A.16. Hence, we can apply the corresponding multiplication rule to the u_i to deduce t.

□

Lemma A.18. *For all $ndg \in ndgraphs(P)$ and $s, t \in known\Updownarrow(ndg)$, there is a deduction extension ndg' of ndg with $(s*t)\downarrow_{\mathcal{RBP}_e} \in_{ACC} known\Updownarrow(ndg')$.*

Proof. By Lemma A.17, it is sufficient to show that for all $m \in factors((s*t)\downarrow_{\mathcal{RBP}_e})$, there is a deduction extension of ndg where m or $(m^{-1})\downarrow_{\mathcal{RBP}_e}$ known. First, note that the following holds for $factors((s*t)\downarrow_{\mathcal{RBP}_e})$.

$$\begin{array}{lr}
\quad factors((s*t)\downarrow_{\mathcal{RBP}_e}) & \\
\subseteq_{ACC} \{1\} \cup factors(s*t)\downarrow_{\mathcal{RBP}_e} \cup afactors(s*t)\downarrow_{\mathcal{RBP}_e} & [\text{by Lemma A.10}] \\
\subseteq_{ACC} \{1\} \cup factors(s) \cup factors(t) \cup afactors(s) \cup afactors(t) & [\text{simplify}] \\
\subseteq_{ACC} \{1\} \cup \{s, (s^{-1})\downarrow_{\mathcal{RBP}_e}\} \cup \{t, (t^{-1})\downarrow_{\mathcal{RBP}_e}\} & [\text{by Lemma A.6}] \\
\quad \cup\, afactors(s) \cup afactors(t) &
\end{array}$$

Since 1, s, $(s^{-1})\downarrow_{\mathcal{RBP}_e}$, t, $(t^{-1})\downarrow_{\mathcal{RBP}_e}$, $afactors(s)$, and $afactors(t)$ are already known or deducible, this means all elements of $factors((s*t)\downarrow_{\mathcal{RBP}_e})$ are deducible. □

We will now prove a lemma that will be used to show that *drules* includes all required deconstruction rules for the rewrite rules in $\mathcal{ST}$.

Lemma A.19. *For all $ndg \in ndgraphs(P)$, $t \in known\Updownarrow(ndg)$ and valid positions p in t such that $root(t|_{p'}) \neq *$ for all p' above or equal to p, either*

a) there is a position $\tilde{p} \neq []$ strictly above p such that $t|_{\tilde{p}} \in_{ACC} known\Downarrow(ndg)$ and $t|_{p'} \in_{ACC} known\Updownarrow(ndg)$ for all valid positions p' in t that have a sibling above or equal to $\tilde{p}$, or

b) $t|_p \in_{ACC} known\Updownarrow(ndg)$ and $t|_{p'} \in_{ACC} known\Updownarrow(ndg)$ for all valid positions p' in t that have a sibling above or equal to p.

Proof. Let *ndg* and t arbitrary. We prove the statement by induction over positions p such that $root(t|_{p'}) \neq *$ for all positions p'. For the empty position [], b) clearly holds since $t|_{[]} = t$ and [] has no siblings. For the induction step, we first assume that a) holds for the position p. Then a) also holds for all valid positions $p' = p \cdot [i]$ in t. Now assume that b) holds for p. If $t|_p$ is $\Uparrow$-known and not $\Downarrow$-known, then it cannot be the conclusion of the Coerce rule and must be the conclusion of a construction rule. Then $t|_p$ is either the conclusion of a multiplication or a construction rule of the form $[\mathsf{K}^{\Uparrow}(s_1), \ldots, \mathsf{K}^{\Uparrow}(s_k)] \,{-}[]{\rightarrow}\, [\mathsf{K}^{\Uparrow}(f(s_1, \ldots, s_k))]$. In the first case, $root(t|_p) = *$, which contradicts our assumptions. In the second case, b) also holds for all $p' = p \cdot [i]$ with $1 \leq i \leq k$ because $t|_{p \cdot [i]} = s_i \in_{ACC} known{\Updownarrow}(ndg)$ and the terms at sibling positions of p' are also known. If $t|_p$ is $\Downarrow$-known and p satisifies b), then for all valid positions $p' = p \cdot [i]$ in t, p' satisifies a). □

Proof. (Proof of Lemma A.12) We prove both inclusions separately.

$\subseteq_{ACC}$: We show by induction over dependency graphs that for all $dg \in dgraphs_{ACC}(\lceil P \cup MD \rceil_{insts}^{\mathcal{RBP}_e})$ with dg $\downarrow_{\mathcal{RBP}_e}$-normal, there is $ndg \in ndgraphs(P)$ such that

$$known(dg) \subseteq_{ACC} known{\Updownarrow}(ndg) \qquad (1)$$
$$stfacts(dg) \subseteq^{\sharp}_{ACC} stfacts(ndg) \qquad (2)$$
$$created(dg) =_{ACC} created(ndg) \qquad (3)$$
$$\overline{trace(dg)} =_{ACC} \overline{trace(ndg)}. \qquad (4)$$

This clearly holds for $dg = ([], \emptyset)$. Let $dg = (I, D) \in dgraphs_{ACC}(\lceil P \cup MD \rceil_{insts}^{\mathcal{RBP}_e})$ with dg $\downarrow_{\mathcal{RBP}_e}$-normal, and $ndg = \left(\tilde{I}, \tilde{D}\right) \in ndgraphs(P)$ such that (1)–(4) hold. Let $ri \in ginsts_{ACC}(\lceil P \cup MD \rceil_{insts}^{\mathcal{RBP}_e} \cup \{\text{Fresh}\})$ arbitrary such that $dg' = (I \cdot ri, D \uplus D') \in dgraphs_{ACC}(\lceil P \cup MD \rceil_{insts}^{\mathcal{RBP}_e})$. Then we have to show that there is an $ndg' \in ndgraphs(P)$ that satisfies (1)–(4) with respect to dg'. We perform a case distinction on ri.

- $ri \in ginsts_{ACC}(\text{Fresh})$: Condition (3) for dg and ndg ensures that we can extend $\tilde{I}$ with ri to obtain ndg' without violating unique Fresh instances (**DG4**).
- $ri \in ginsts_{ACC}(\lceil P \rceil_{insts}^{\mathcal{RBP}_e})$: We append ri to $\tilde{I}$ and extend $\tilde{D}$ with the required dependencies to obtain ndg', which is possible because of condition (2) for dg and ndg.
- $ri \in ginsts_{ACC}(\lceil MD \rceil_{insts}^{\mathcal{RBP}_e})$: Let $ri = l \,{-}[a]{\rightarrow}\, r$. For all rules except for the adversary receive, send, fresh name, and public name rules, we have $l = [\mathsf{K}(m_1), \ldots, \mathsf{K}(m_n)]$, $a = []$, and $r = [\mathsf{K}(m)]$. If $m = f(m_1, \ldots, m_n)$, then we say the instance is a trivial variant of the rule. Otherwise it is nontrivial. We must show that there is a deduction extension ndg' of ndg such that $m \in_{ACC} known{\Updownarrow}(ndg')$. We can assume that $m \notin_{ACC} known{\Updownarrow}(ndg)$ and $m_i \in_{ACC} known{\Uparrow}(ndg)$ for $1 \leq i \leq n$ because of Lemma A.15. If m is a product, then we just have to show that m is accessible in one of the premise facts. Then, we can use Lemma A.14 to show that all *factors* or their inverses are known. Hence m is deducible by Lemma A.17.
  - $\mathsf{Out}(m) \,{-}[]{\rightarrow}\, \mathsf{K}(m)$: We can use Recv if m is not a product. If m is a product, then we can deduce m since it is accessible in $\mathsf{Out}(m)$.
  - $\mathsf{K}(m) \,{-}[\mathsf{K}(m)]{\rightarrow}\, \mathsf{In}(m)$: We can use Send.
  - $\mathsf{Fr}(n) \,{-}[]{\rightarrow}\, \mathsf{K}(m)$: We can use the construction rule for fresh names.
  - ${-}[]{\rightarrow} \mathsf{K}(c)$ for $c \in \mathsf{PN} \cup \{1\}$: We can use the corresponding construction rules.
  - Trivial variants $[\mathsf{K}(m_1), \ldots, \mathsf{K}(m_n)] \,{-}[]{\rightarrow}\, [\mathsf{K}(f(m_1, \ldots, m_n))]$ for $f \in \Sigma_{\mathcal{ST}} \cup \{inv\}$: We can use the the corresponding construction rule since $m_i \in_{ACC} known{\Uparrow}(ndg')$.

- $\mathsf{K}(m^{-1}) \dashv[]\mapsto \mathsf{K}(m)$: We can deduce m by Lemma A.16.
- $[\mathsf{K}(m_1), ..., \mathsf{K}(m_k)] \dashv[]\mapsto [\mathsf{K}(m)]$ for $f \in \Sigma_{\mathcal{ST}}$, $m = f(m_1, ..., m_k){\downarrow_{\mathcal{RBP}_e}}$, such that $m \neq_{ACC} f(m_1, ..., m_k)$. Then, there is $l \to r$ in $\mathcal{ST}$ such that $f(m_1, ..., m_k)$ is an instance of l and either $m = r \in \mathcal{T}_{\Sigma_{\mathcal{ST}}}(\emptyset)$ and in normal form or there are p and j such that $m = m_j|_p$. In the first case, we can use construction rules to build m. In the second case, $m_i \in_{ACC} \mathit{known}{\Updownarrow}(\mathit{ndg})$ and p satisfies all the conditions from Lemma A.19. Since we assume that $m \notin_{ACC} \mathit{known}{\Updownarrow}(\mathit{ndg})$, case a) must hold and we can use the corresponding rule from $\mathit{drules}(l,p)$ to deduce m.
- $[\mathsf{K}(m_1), \mathsf{K}(m_2)] \dashv[]\mapsto [\mathsf{K}((m_1{*}m_2){\downarrow_{\mathcal{RBP}_e}})]$: Since m_1 and m_2 are known, we can use Lemma A.18.
- trivial variant of *exp* : First note that the base cannot be an exponentiation since the result would not be in normal-form otherwise. We can therefore just use the construction rule for exponentiation.
- nontrivial variant of *exp* : The rule must be equal to $[\mathsf{K}(a\hat{}c), \mathsf{K}(d)] \dashv[]\mapsto [\mathsf{K}(m)]$ for $m = ((a\hat{}c)\hat{}d){\downarrow_{\mathcal{RBP}_e}}$. Since a cannot be an exponentiation and must be in normal-form, $m = a\hat{}((c{*}d){\downarrow_{\mathcal{RBP}_e}})$. The term $a\hat{}c$ must be known and d must be $\Uparrow$-known. If $a\hat{}c$ is only $\Uparrow$-known in ndg, then it must be the result of an exponentiation construction rule. Then a and c are also $\Uparrow$-known and we can deduce $(c{*}d){\downarrow_{\mathcal{RBP}_e}}$ by Lemma A.18. Therefore, we deduce m with $[\mathsf{K}^{\Uparrow}(a), \mathsf{K}^{\Uparrow}((c{*}d){\downarrow_{\mathcal{RBP}_e}})] \dashv[]\mapsto [\mathsf{K}^{\Uparrow}(m)]$. If $a\hat{}c$ is $\Downarrow^{\mathsf{d}}$-known in ndg, then we can use the corresponding exponentiation rule unless this would violate **N6** or **N10**.

  If this would violate **N6**, then a does not contain any fresh names and $\mathit{nifactors}((c{*}d){\downarrow_{\mathcal{RBP}_e}}) \subseteq_{ACC} \mathit{nifactors}(d)$. Hence $\mathit{factors}((c{*}d){\downarrow_{\mathcal{RBP}_e}}) \subseteq_{ACC} \mathit{factors}(d)$ and therefore $\mathit{factors}((c{*}d){\downarrow_{\mathcal{RBP}_e}})$ (or their inverses) known by Lemma A.14 and $(c{*}d){\downarrow_{\mathcal{RBP}_e}}$ deducible by Lemma A.17. Since a also deducible because it does not contain fresh names, we can use the construction rule for exponentiation as before.

  If the exponentiation rule would violate **N10**, then $a\hat{}c = \hat{e}(p,q)\hat{}c$ for some terms p and q that do not contain any fresh names, $\hat{e}(p,q)\hat{}c$ is the result of a deconstruction rule for $\hat{e}$ with input $\mathsf{K}^{\Downarrow^{\mathsf{d}}}([t_1]p)$ and $\mathsf{K}^{\Downarrow^{\mathsf{d}}}([t_2]q)$ such that $\mathit{nifactors}(t_i) \subseteq_{ACC} \mathit{nifactors}(d)$ for $i = 1$ or $i = 2$. Note that this implies that $c = (t_1{*}t_2){\downarrow_{\mathcal{RBP}_e}}$ and $m = \hat{e}(p,q)\hat{}((t_1{*}t_2{*}d){\downarrow_{\mathcal{RBP}_e}})$. We assume without loss of generality that $i = 2$. Then, we can replace the deconstruction rule for $\hat{e}$ with $\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}([t_1]p),\ \mathsf{K}^{\Uparrow}(q)\right] \dashv[]\mapsto \left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\hat{e}(p,q)\hat{}t_1)\right]$ since q does not contain fresh names. Since $\mathit{nifactors}(t_2) \subseteq_{ACC} \mathit{nifactors}(d)$ and d known, we can deduce $(t_2{*}d){\downarrow_{\mathcal{RBP}_e}}$ and then deduce m with the exponentiation rule

$$\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(\hat{e}(p,q)\hat{}t_1), \mathsf{K}^{\Uparrow}((t_2{*}d){\downarrow_{\mathcal{RBP}_e}})\right] \dashv[]\mapsto [\mathsf{K}^{\Downarrow^{\mathsf{e}}}(m)].$$

  If $a\hat{}c$ is only $\Downarrow^{\mathsf{e}}$-known, it must be the conclusion of an exponentiation rule and there must be e and f such that $a\hat{}c = a\hat{}((e{*}f){\downarrow_{\mathcal{RBP}_e}})$, $a\hat{}e$ is $\Downarrow$-known, and f is known. Then $m = a\hat{}((e{*}f{*}c){\downarrow_{\mathcal{RBP}_e}})$ and we can use $\left[\mathsf{K}^{\Downarrow^{\mathsf{d}}}(a\hat{}e), \mathsf{K}^{\Uparrow}((f{*}c){\downarrow_{\mathcal{RBP}_e}})\right] \dashv[]\mapsto [\mathsf{K}^{\Downarrow^{\mathsf{e}}}(m)]$ unless this violates **N6** or **N10** which can be handled like in the previous cases.
- variants of *smult*: These case are analogous to *exp*
- trivial variant of $\hat{e}$: We can use the construction rule for $\hat{e}$. The restriction **N11** is only concerned with deconstruction rules for $\hat{e}$.

  - non-trivial variant of $\hat{e}$: The rule must be equal to $[\mathsf{K}([a]p), \mathsf{K}([b]q)] \dashv[]\!\!\rightarrow [\mathsf{K}(m)]$ for $m = \hat{e}(p,q)\hat{}((a{*}b)\downarrow_{\mathcal{RBP}_e})$ or equal to $[\mathsf{K}([a]p), \mathsf{K}(q)] \dashv[]\!\!\rightarrow [\mathsf{K}(\hat{e}(p,q)\hat{}a)]$. We begin by considering the first case. Note that if the variant used matches the one in **N11**, then the premises have to be ordered in the right way. The bilinear pairing rules require both messages to be $\Downarrow$-known. If one of the messages is not $\Downarrow$-known, e.g., $[a]p$, then it must be the result of the construction rule for scalar multiplication and both a and p must be known. We can then use a bilinear pairing rule with input p and modify the exponent to include a by applying an exponentiation rule to the result. If the rule is equal to $[\mathsf{K}([a]p), \mathsf{K}(q)] \dashv[]\!\!\rightarrow [\mathsf{K}(\hat{e}(p,q)\hat{}a)]$, where q is not a scalar multiplication, and $[a]p$ is not $\Downarrow$-known, we can apply the construction rule for $\hat{e}$ to p and q followed by applying the construction rule for exponentiation to $\hat{e}(p,q)$ and a.
  - $[\mathsf{K}(u_1\sharp\ldots\sharp u_k), \mathsf{K}(w_1\sharp\ldots\sharp w_l)] \dashv[]\!\!\rightarrow [\mathsf{K}(u_1\sharp\ldots\sharp u_k\sharp w_1\sharp\ldots\sharp w_l)]$: We can deduce u_i and w_i using the deconstruction rule for $\sharp$ (if required) and use the following rule to deduce the desired term: $[\mathsf{K}^{\Uparrow}(u_1), \ldots, \mathsf{K}^{\Uparrow}(u_l), \mathsf{K}^{\Uparrow}(w_l), \ldots, \mathsf{K}^{\Uparrow}(w_l)] \dashv[]\!\!\rightarrow [\mathsf{K}(u_1\sharp\ldots\sharp u_k\sharp w_1\sharp\ldots\sharp w_l)]$.
  - $[\mathsf{K}(u_1\sharp\ldots\sharp u_k)] \dashv[]\!\!\rightarrow [\mathsf{K}(u_{i_1}\sharp\ldots\sharp u_{i_l})]$: We can first deduce the u_{i_j} and then use the construction rule for $\sharp$ to deduce the desired term.

$\supseteq_{ACC}$: We show by induction over dependency graphs that for all $ndg \in ndgraphs(P)$, there is $dg \in dgraphs_{ACC}(\lceil P \cup MD\rceil^{\mathcal{RBP}_e}_{insts})$ with dg $\downarrow_{\mathcal{RBP}_e}$-normal such that

$$\begin{aligned} known{\Updownarrow}(ndg) &\subseteq_{ACC} known(dg) && (1)\\ stfacts(ndg) &\subseteq^{\sharp}_{ACC} stfacts(dg) && (2)\\ created(ndg) &=_{ACC} created(dg) && (3)\\ \overline{trace(ndg)} &=_{ACC} \overline{trace(dg)}. && (4) \end{aligned}$$

This clearly holds for $ndg = ([], \emptyset)$. Let $ndg = (I, D) \in ndgraphs(P)$ and $dg = \big(\tilde{I}, \tilde{D}\big) \in dgraphs_{ACC}(\lceil P \cup MD\rceil^{\mathcal{RBP}_e}_{insts})$ with dg $\downarrow_{\mathcal{RBP}_e}$-normal such that (1)–(4) hold. Let $ri \in ginsts_{ACC}(\lceil P\rceil^{\mathcal{RBP}_e}_{insts} \cup \{\textsc{Fresh}\} \cup ND)$ arbitrary such that $ndg' = (I \cdot ri, D \uplus D') \in ndgraphs(P)$. Then we have to show that there is an $dg' \in dgraphs_{ACC}(\lceil P \cup MD\rceil^{\mathcal{RBP}_e}_{insts})$ that satisfies (1)–(4) with respect to ndg'. We perform a case distinction on ri.

- $ri \in ginsts_{ACC}(\textsc{Fresh})$: Analogous to other inclusion.
- $ri \in ginsts_{ACC}(\lceil P\rceil^{\mathcal{RBP}_e}_{insts})$: Analogous to other inclusion.
- $ri \in ginsts_{ACC}(ND)$: Except for some deconstruction rules for $\Sigma_{\mathcal{ST}}$ and the n-ary rules for multiplication and $\sharp$, there is a corresponding rule in $\lceil \mathrm{MD}\rceil^{\mathcal{RBP}_e}_{insts}$. For a rewriting rule $f(t_1,\ldots,t_k) \dashv[]\!\!\rightarrow t_i|_p$ from $\mathcal{ST}$, all corresponding deconstruction rules can be simulated with trivial variants of rules for function symbols that occur in t_i and the variant $[\mathsf{K}(t_1),\ldots,\mathsf{K}(t_k)] \dashv[]\!\!\rightarrow [\mathsf{K}(t_i|_p)]$. The n-ary rules for multiplication and $\sharp$ can be simulated by the variants of the binary rules for $*$ and $\sharp$.

$\square$

A.1.3 Proofs for Formulas

We prove Lemma 3.26 directly for the equational theory with bilinear pairings and AC operators. Remember that we have adapted the definition of guarded trace property as follows. A guarded trace formula φ is a *guarded trace property* if it is closed and for all subterms t of φ, $root(t)$ is a variable, a public name, equal to $\sharp$, or an irreducible function symbol from $\Sigma_{\mathcal{ST}}$. We adapt Lemma 3.26 as follows.

Lemma A.20. *For all $\downarrow_{\mathcal{RBP}_e}$-normal traces tr and guarded trace properties φ,*

$$tr \vDash_{\mathcal{BP}_e} \varphi \textit{ if and only if } tr \vDash_{ACC} \varphi.$$

For the following proofs, we define the *allowed function symbols* as

$$\Sigma_{allowed} = \{f \in \Sigma_{\mathcal{ST}} | f \text{ irreducible}\} \cup \{\sharp\} \cup \mathsf{PN}.$$

We first prove the following lemma.

Lemma A.21. *For all formulas φ, traces tr, tr' with $tr =_{\mathcal{E}} tr'$, and valuations θ, θ' with $\theta(x) =_{\mathcal{E}} \theta'(x)$ for all $x \in fvars(\phi)$,*

$$(tr, \theta) \vDash_{\mathcal{E}} \varphi \textit{ if and only if } (tr', \theta') \vDash_{\mathcal{E}} \varphi.$$

Proof. The proof proceeds by structural induction over formulas. □

To prove Lemma A.20, we require the following definition.

Definition A.22. *Let tr be an arbitrary trace and X a set of variables. Two valuations θ_1 and θ_2 are equivalent for tr and $X \subseteq \mathcal{V}_{msg} \cup \mathcal{V}_{temp}$, written $\theta_1 \equiv_{tr,X} \theta_2$ if the following conditions hold:*

1. *There is a substitution σ such that $\theta_1|_X$ and $\theta_2|_X$ are instances of σ, $dom(\sigma) = X$, and $range(\sigma) \subseteq \mathcal{T}_{\Sigma_{allowed}}(vrange(\sigma))$.*
2. *For the unique substitutions τ_1 and τ_2 with $\theta_i(x) = x(\sigma \circ \xi_i)$ for $i \in \{1, 2\}$ and all $x \in X$, the following holds for all $y', y \in vrange(\sigma)$ and $i \in \{1, 2\}$.*
   a) *$root(\xi_i(y)) \in (\Sigma_{\mathcal{BP}_e} \setminus \Sigma_{allowed}) \cup \mathsf{FN}$.*
   b) *$\xi_i(y) \neq_{ACC} \xi_i(y')$ for $y \neq y'$.*
   c) *$\xi_1(y) \in_{ACC} St(tr)$ if and only if $\xi_2(y) \in_{ACC} St(tr)$.*
   d) *$\xi_1(y) \in_{ACC} St(tr)$ implies $\xi_1(y) = \xi_2(y)$.*
3. *Both valuations agree on temporal variables, i.e., it holds that $\theta_1|_{\mathcal{V}_{temp}} = \theta_2|_{\mathcal{V}_{temp}}$.*

Note that the previous definition allows us to modify valuations by *consistently* replacing subterms that do not occur in the trace and whose root symbols are not in $\Sigma_{allowed}$.

Lemma A.23. *Let φ be an arbitrary formula such that all subterms only contain root symbols from $\mathcal{V}_{msg} \cup \mathcal{V}_{temp} \cup \Sigma_{allowed}$. Then, for all traces tr and all valuations θ_1, θ_2 such that $\theta_1 \equiv_{tr,fvars(\varphi)} \theta_2$, it holds that $(tr, \theta_1) \vDash_{ACC} \varphi$ if and only if $(tr, \theta_2) \vDash_{ACC} \varphi$.*

Proof. We prove this by induction over formulas where, without loss of generality, all actions are of the form $F(x_1, \ldots, x_k)@i$. This can be achieved by replacing actions $F(t_1, \ldots, t_k)@i$ with the formula $\exists x_1 \ldots x_k. F(x_1, \ldots, x_k)@i \wedge x_1 \approx t_1 \wedge \ldots \wedge x_k \approx t_k$. Let tr be arbitrary and θ_1, θ_2 arbitrary such that $\theta_1 \equiv_{tr,fvars(\varphi)} \theta_2$. Let σ and ξ_1, ξ_2 denote the substitutions that witness that θ_1 and θ_2 are equivalent.

$i \prec j$, $i \approx j$. Holds because of 3.

$F(x_1, \ldots, x_k)@i$. $(tr, \theta_1) \vDash_{ACC} F(x_1, \ldots, x_k)@i$ implies that $F(x_1, \ldots, x_k)\theta_1 \in_{ACC} tr_{\theta_1(i)}$. Hence $x_i\theta_1 \in_{ACC} St(tr)$ and by 2 c) and d), we have $x_i\theta_1 = x_i\theta_2$. Together with 3., we therefore have $F(x_1, \ldots, x_k)\theta_2 \in_{ACC} tr_{\theta_2(i)}$ and $(tr, \theta_2) \vDash_{ACC} F(x_1, \ldots, x_k)@i$.

$\neg\psi$. By the induction hypothesis, we have that $(tr,\theta_1)\vDash_{ACC}\psi$ if and only if $(tr,\theta_2)\vDash_{ACC}\psi$ since $\theta_1\equiv_{tr,fvars(\psi)}\theta_2$ because $fvars(\psi)=fvars(\neg\psi)=fvars(\varphi)$. We therefore have $(tr,\theta_1)\vDash_{ACC}\neg\psi$ if and only if $(tr,\theta_2)\vDash_{ACC}\neg\psi$.

$\psi_1\wedge\psi_2$. We can use the induction hypothesis on ψ_1 and ψ_2 and combine the results.

$s\approx t$. We must show that

$$s\theta_1=(s\sigma)\xi_1=_{ACC}(t\sigma)\xi_1=t\theta_1 \qquad (1)$$

if and only if

$$s\theta_2=(s\sigma)\xi_2=_{ACC}(t\sigma)\xi_2=t\theta_2. \qquad (2)$$

To achieve this, we prove that (1) implies that $s\sigma=_{ACC}t\sigma$, which implies (2). The other direction can then be proved analogously. More precisely, we prove that for all terms $u,w\in\mathcal{T}_{\Sigma_{allowed}}(\mathcal{V})$, it holds that $u\xi_1=_{ACC}w\xi_1$ implies $u=_{ACC}w$. The proof proceeds by induction over u.

For the base case of the induction, let $u=y$ for a variable y. Then $root(\xi_1(y))=root(w\xi_1)$ and w must therefore also be a variable z because of 2 a). Furthermore, $\xi_1(y)=_{ACC}\xi_1(z)$ implies $y=z$ because of 2 b), i.e., $u=_{ACC}w$. In the first step case, we assume that $u=f(r_1,\ldots,r_k)$ for $f\in\Sigma_{allowed}$. Hence $t=f(q_1,\ldots,q_k)$ for some terms q_i because of 2 a). Therefore, it holds that $r_i\xi_1=_{ACC}q_i\xi_1$ and by the induction hypothesis, it holds that $r_i=_{ACC}q_i$ and therefore $u=_{ACC}w$. In the next step case, we assume that $u=r_1\sharp\ldots\sharp r_k$ and $root(r_i)\neq\sharp$. Then $w=q_1\sharp\ldots\sharp q_k$ for some q_i such that $root(q_i)\neq\sharp$ and $r_i\xi_1=_{ACC}q_i\xi_1$ by 2 a). We can use the induction hypothesis on the r_i to obtain $r_i=_{ACC}q_i$, which yields $u=_{ACC}w$.

$\exists x{:}s.\,\psi$. From $(tr,\theta_1)\vDash_{ACC}\exists x{:}s.\,\psi$, it follows that there is u of sort s such that $(tr,\theta_1[x\mapsto u])\vDash_{ACC}\psi$. To use the induction hypothesis, we have to find u' such that $\theta_1[x\mapsto u]\equiv_{tr,fvars(\varphi)\cup\{x\}}\theta_2[x\mapsto u']$. Then, $(tr,\theta_2[x\mapsto u'])\vDash_{ACC}\psi$, which implies $(tr,\theta_2)\vDash_{ACC}\exists x{:}s.\,\psi$. If x is a temporal variable, then we can set $u'=u$ and it is not hard to see that $\theta_1[x\mapsto u]\equiv_{tr,fvars(\varphi)\cup\{x\}}\theta_2[x\mapsto u]$.

Otherwise, we proceed as follows. Let $\theta_1':=\theta_1[x\mapsto u]$ and σ' and ξ_1' such that $dom(\sigma')=fvars(\varphi)\cup\{x\}$, $range(\sigma')\subseteq\mathcal{T}_{\Sigma_{allowed}}(vrange(\sigma'))$, $\theta_1'(z)=z(\sigma'\circ\xi_1')$ for all $z\in dom(\sigma')$, and ξ_1' satisfies 2. a) and 2. b). We can choose σ' such that $\sigma'=\sigma\cup\{w/x\}$ and we can choose ξ_1' such that $\xi_1'=\xi_1\cup\{\vec{q}\,/\vec{y}\,\}$ for $\vec{y}=vars(w)\setminus vrange(\sigma)$ and terms $\vec{q}$. To complete the proof, we have to choose $\vec{r}$ such that σ', ξ_1' and $\xi_2'{:}=\xi_2\cup\{\vec{r}\,/\vec{y}\,\}$ witness that $\theta_1'\equiv_{tr,fvars(\varphi)\cup\{x\}}\theta_2[x\mapsto w\xi_2']$, i.e., we set $u'=w\xi_2'$. Note that conditions 1. and 3. are satisfied independently of the choice of $\vec{r}$. To satisfy 2. a)–d), we distinguish two cases for each $1\le i\le|\vec{q}\,|$. If $q_i\in St(tr)$, we set $r_i=q_i$ because of c) and d). Otherwise, we set q_i to a new fresh name because of a) and b).

□

Proof. (of Lemma A.20) We prove for all trace formulas φ such that all subterms only contain root symbols from $\mathcal{V}_{temp}\cup\mathcal{V}_{msg}\cup\Sigma_{allowed}$ and for all for all $\downarrow_{\mathcal{RBP}_e}$-normal traces tr and valuations θ, it holds that $(tr,\theta)\vDash_{\mathcal{BP}_e}\varphi$ if and only if $(tr,\theta)\vDash_{ACC}\varphi$. Our proof proceeds by induction over formulas. Let tr be an arbitrary $\downarrow_{\mathcal{RBP}_e}$-normal trace and θ be an arbitrary $\downarrow_{\mathcal{RBP}_e}$-normal valuation.

$i\prec j$, $i\approx j$. Same semantics in $\vDash_{\mathcal{BP}_e}$ and $\vDash_{ACC}$.

$F(t_1, ..., t_k)@i$. $F(t_1,\ ...,\ t_k)\theta \in_{\mathcal{BP}_e} tr_{\theta(i)}$ iff $F(t_1,\ ...,\ t_k)\theta \in_{ACC} tr_{\theta(i)}$ since tr and $F(t_1, ..., t_k)\theta$ are $\downarrow_{\mathcal{RBP}_e}$-normal because t_i only contains allowed function symbols and θ is $\downarrow_{\mathcal{RBP}_e}$-normal.

$\neg\psi$. Follows immediately from induction hypothesis.

$\psi_1 \wedge \psi_2$. Follows immediately from induction hypothesis.

$t_1 \approx t_2$. $t_1\theta =_{\mathcal{BP}_e} t_2\theta$ iff $t_1\theta =_{ACC} t_2\theta$ since $t_i\theta$ is $\downarrow_{\mathcal{RBP}_e}$–normal for $i = 1$ and $i = 2$ because t_i only contains allowed function symbols and θ is $\downarrow_{\mathcal{RBP}_e}$-normal.

$\exists x. \psi$. We prove both directions separately.

$$\begin{array}{ll} & (tr, \theta) \vDash_{\mathcal{BP}_e} \exists x. \psi \\ \text{implies} & \text{exists } u \text{ such that } (tr, \theta[x \mapsto u]) \vDash_{\mathcal{BP}_e} \psi \\ \text{implies} & \text{exists } u' \text{ such that } (tr, \theta[x \mapsto u']) \vDash_{\mathcal{BP}_e} \psi \text{ and } u' \downarrow_{\mathcal{RBP}_e}\text{-normal} \quad [u' = u\downarrow_{\mathcal{RBP}_e}] \\ \text{implies} & \text{exists } u' \text{ such that } (tr, \theta[x \mapsto u']) \vDash_{ACC} \psi \text{ and } u' \downarrow_{\mathcal{RBP}_e}\text{-normal} \quad [\text{IH}] \\ \text{implies} & (tr, \theta) \vDash_{ACC} \exists x. \psi \end{array}$$

$$\begin{array}{ll} & (tr, \theta) \vDash_{ACC} \exists x. \psi \\ \text{implies} & \text{exists } u \text{ such that } (tr, \theta[x \mapsto u]) \vDash_{ACC} \psi \\ \text{implies} & \text{exists } u' \text{ such that } (tr, \theta[x \mapsto u']) \vDash_{ACC} \psi \text{ and } u' \downarrow_{\mathcal{RBP}_e}\text{-normal} \quad [(*)] \\ \text{implies} & \text{exists } u' \text{ such that } (tr, \theta[x \mapsto u]) \vDash_{\mathcal{BP}_e} \psi \text{ and } u' \downarrow_{\mathcal{RBP}_e}\text{-normal} \quad [\text{IH}] \\ \text{implies} & (tr, \theta) \vDash_{\mathcal{BP}_e} \exists x. \psi \end{array}$$

$(*)$: Here, we use Lemma A.23 and show that there is a term u' such that $\theta[x \mapsto u] \equiv_{tr, fvars(\psi)} \theta[x \mapsto u']$ and u' is $\downarrow_{\mathcal{RBP}_e}$-normal. The proof that such an u' exists that is also $\downarrow_{\mathcal{RBP}_e}$-normal is analogous to the last case in the proof of Lemma A.23.

□

A.2 Proofs for Constraint Solving

We prove the soundness and completeness of all constraint solving rules including those for bilinear pairings and AC operators.

Proof. (of Theorem 3.33) We first consider the completeness of rules. If $\Gamma \rightsquigarrow_P \{\Gamma_1, ..., \Gamma_k\}$ for the given rule and $(dg, \theta) \Vdash \Gamma$ for an arbitrary dependency graph dg and valuation θ, then there must be θ' and i such that $(dg, \theta') \Vdash \Gamma_i$.

$\mathcal{S}_@$. Since $(dg, \theta) \Vdash f@i$, it holds that $f\theta \in_{ACC} trace(dg)_{\theta(i)}$. Therefore, $\theta(i)$ must be labeled with some non-silent rule ri and $f\theta$ must be equal to one of the actions of ri. Hence $\theta(i)$ must be an instance of a variant of a rule in P or an instance of the Recv rule. Hence for one of the cases $i: ru_j, g_j \approx f$, where ru_j contains only freshly chosen variables, θ can be extended to θ' such that $(dg, \theta') \Vdash i: ru_j, g_j \approx f, \Gamma$.

$\mathcal{S}_\approx$. Since $(dg,\ \theta) \Vdash s \approx t$, it holds that $s\theta =_{ACC} t\theta$. Hence, the valuation θ (restricted to $vars(s,t)$) is an AC-unifier of s and t and therefore an instance of a $\sigma \in unif_{ACC}^{fvars(\Gamma)}(s,t)$, i.e., there is a valuation θ' such that $\theta|_{fvars(\Gamma)} = (\theta' \circ \sigma)|_{fvars(\Gamma)}$. We therefore have $(dg, \theta') \Vdash \Gamma\sigma$.

$\boldsymbol{\mathcal{S}}_{\exists}$. By the $\vDash_{ACC}$-semantics of $\exists$, there is a sequence of terms and timepoints $\vec{u}$ such that $(trace(dg), \theta[\vec{y} \mapsto \vec{u}]) \vDash_{ACC} \varphi\{\vec{y}/x\}$. Since we use freshly chosen variables $\vec{y}$, $(dg, \theta[\vec{y} \mapsto \vec{u}])$ still satisfies the other constraints.

$\boldsymbol{\mathcal{S}}_{\forall,@}$. Since $(dg,\ \theta) \Vdash \forall\vec{x}.\neg(f@i) \vee \varphi$ and $(dg,\ \theta) \Vdash (f@i)\sigma$, where σ instantiates all variables from $\vec{x}$ with terms built using variables from $fvars(\Gamma)$, $(dg,\ \theta) \Vdash \varphi\sigma$ since $(\neg(f@i) \vee \varphi)\sigma \wedge (f@i)\sigma$ is logically equivalent to $\varphi\sigma$.

$\boldsymbol{\mathcal{S}}_{\forall,\approx}$. This case is analogous to the previous one.

$\boldsymbol{\mathcal{S}}_{\vee}$, $\boldsymbol{\mathcal{S}}_{\wedge}$, $\boldsymbol{\mathcal{S}}_{\bot}$. Directly follow from $\vDash_{ACC}$-semantics of $\vee$, $\wedge$, and $\bot$.

$\boldsymbol{\mathcal{S}}_{\not\approx}$. Since $(dg, \theta) \Vdash \neg(t \approx t)$, it holds that $t\theta \neq_{ACC} t\theta$, which is a contradiction. This means $models_P(\Gamma) = \emptyset = models_P(\bot)$.

$\boldsymbol{\mathcal{S}}_{\neg @}$. Since $(dg, \theta) \Vdash \neg(f@i)$, it holds that $f\theta \notin_{ACC} trace(dg)_{\theta(i)}$. But some other constraint (from $as(\Gamma)$) implies that $f\theta \in_{ACC} trace(dg)_{\theta(i)}$, which is a contradiction. This means $models_P(\Gamma) = \emptyset = models_P(\bot)$.

$\boldsymbol{\mathcal{S}}_{\mathbf{Prem}}$. The node $\theta(i)$ is labeled with $ri\theta$ in dg. It has a non-K^d and non-Fr premise with index u. Hence, there must be an edge starting at some node and some conclusion index v_j of this node because of **DG2**. This node must be labeled with an instance of a variant of a protocol rule or an instance of Send since all other rules only have K^d and Fr conclusions.

$\boldsymbol{\mathcal{S}}_{\rightarrowtail}$. The source and targets must be syntactically equal by **DG1**. Since constraints are understood modulo ACC, we can use $\approx$ here.

$\boldsymbol{\mathcal{S}}_{\mathbf{USrc}}$. There is exactly one incoming edge for each premise by **DG2**. Hence, the source of both edge constraints with the same target must be equal.

$\boldsymbol{\mathcal{S}}_{\mathbf{UTgt}}$. There is at most one outgoing edge for each linear conclusion because of **DG3**. Hence, the target of both edge constraints with the same source must be equal.

$\boldsymbol{\mathcal{S}}_{\mathbf{ULabel}}$. The label of each node must be unique.

$\boldsymbol{\mathcal{S}}_{\mathbf{Acyc}}$. For each $i \prec_\Gamma j$, it must hold that $\theta(i) \prec \theta(j)$. Hence, it cannot hold that $i \prec_\Gamma i$ and $\theta(i) \prec \theta(i)$.

$\boldsymbol{\mathcal{S}}_{\mathbf{UFresh}}$. By **DG4**, Fresh instances are unique in a dependency graph. Note that Fr-facts are only provided by Fresh instances and Fr-facts are linear and can therefore have at most one outgoing edge. Hence, Fr-premises are also unique.

$\boldsymbol{\mathcal{S}}_{\downarrow}$. In a normal dependency graph, labels ri must be $\downarrow_{\mathcal{RBP}_e}$-normal. Since ri not $\downarrow_{\mathcal{RBP}_e}$-normal implies that $ri\theta$ not $\downarrow_{\mathcal{RBP}_e}$-normal for all valuations θ, there is no normal dependency graph that can satisfy a node constraint $i\colon ri$ for an ri that is not $\downarrow_{\mathcal{RBP}_e}$-normal.

$\boldsymbol{\mathcal{S}}_{\mathbf{Prem},\mathsf{K}^{\Uparrow}}$. If $\theta(i)$ has a $\mathsf{K}^{\Uparrow}$-premise, then there must be some earlier node that provides this premise.

$\boldsymbol{\mathcal{S}}_{\rhd,\mathsf{K}^{\Uparrow}}$. If a node $\theta(i)$ provides a premise $\mathsf{K}^{\Uparrow}(m\theta)$ such that $root(m\theta)$ is a function symbol that is not equal to $*$ or invertible, then $\theta(i)$ must be an instance of Coerce or the construction rule for f.

$\boldsymbol{\mathcal{S}}_{\rhd,fr}$. If a node $\theta(i)$ provides a premise $\mathsf{K}^{\Uparrow}(m\theta)$ such that $m\theta \in \mathsf{FN}$, then $\theta(i)$ must be an instance of Coerce or the construction rule for fresh names.

$\boldsymbol{\mathcal{S}}_{\rhd,inv}$. If a node $\theta(i)$ provides a premise $\mathsf{K}^{\Uparrow}(m\theta)$ such that $root(m\theta)$ is an invertible function symbol, then $\theta(i)$ must be an instance of the construction rule for f by **N4**.

$\boldsymbol{\mathcal{S}}_{\triangleright,*}$. If a node $\theta(i)$ provides a premise $\mathsf{K}^{\Uparrow}(m\theta)$ such that $root(m\theta) = *$ and $a \in nifactors(m\theta)$, then there must be an earlier rule that provides $\mathsf{K}^{\Uparrow}(a)$ because $\theta(i)$ is an instance of a multiplication rule whose premises are $nifactors(m\theta)$. We know that if $t \in nifactors(m)$ and t is not a message variable, then $t\theta \in nifactors(m\theta)$.

$\boldsymbol{\mathcal{S}}_{\mathbf{Prem},\mathbf{K}^{\Downarrow y}}$. If $\theta(i)$ has a $\mathsf{K}^{\Downarrow y}$-premise, then there must be a chain that starts at an instance of a Recv node that provides this premises by Lemma 3.23.

$\boldsymbol{\mathcal{S}}_{\twoheadrightarrow}$. If there is a chain from $\theta(i)$ to $(\theta(k), w)$, then there is either a direct edge (first case) or there is an intermediate deconstruction node, an edge from $(\theta(i), 1)$ to the first premise of this node, and a chain from this node to $(\theta(k), w)$ (remaining cases).

$\boldsymbol{\mathcal{S}}_{\mathbf{N3},\Downarrow}$, $\boldsymbol{\mathcal{S}}_{\mathbf{N3},\Uparrow}$. These rules enforce **N3** by identifying nodes with the same knowledge conclusion.

$\boldsymbol{\mathcal{S}}_{\mathbf{N5},\mathbf{1}}$, $\boldsymbol{\mathcal{S}}_{\mathbf{N5},\mathbf{2}}$. These rules enforce **N3**.

$\boldsymbol{\mathcal{S}}_{\mathbf{N6}}$. This rule detects exponentiation rule instance that are forbidden by **N6**.

$\boldsymbol{\mathcal{S}}_{\triangleright,\sharp}$. The reasoning for this rule is similar to the one for $\mathcal{S}_{\triangleright,*}$.

$\boldsymbol{\mathcal{S}}_{\twoheadrightarrow,\sharp}$. This rule specializes the solving of $\twoheadrightarrow$-constraint to chains that end in multisets. Because of **N8**, we do not have to consider cases where a message that is already $\mathsf{K}^{\Uparrow}$-known is extracted or where the extracted message has the form $a\sharp b$.

$\boldsymbol{\mathcal{S}}_{\mathbf{N9}}$. This rule detects scalar multiplication rule instance that are forbidden by **N9**.

$\boldsymbol{\mathcal{S}}_{\mathbf{N10}}$. This rule detects exponentation rule instances that are forbidden by **N10**.

$\boldsymbol{\mathcal{S}}_{\mathbf{N11}}$. This rule detects $\hat{e}$ rule instances where the order of premises is wrong according to **N11**.

We now consider the soundness of rules. For all $\mathcal{I}$-rules, the soundness is immediate since we only add new constraints without modifying or removing any old constraints. For the remaining rules, we have to show the following. If $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_k\}$ for the given rule and $(dg, \theta) \Vdash \Gamma_i$ for an arbitrary dependency graph dg and valuation θ and index $1 \leq i \leq k$, then there must be θ' such that $(dg, \theta') \Vdash \Gamma$.

$\boldsymbol{\mathcal{S}}_{\approx}$. The constraints Γ are instantiated by σ and the equality $(t \approx s)\sigma =_{ACC} (t\sigma \approx t\sigma)$ is remove. Therefore, $\theta' := \theta \circ \sigma$ is a valuation such that $(dg, \theta') \Vdash \Gamma$.

$\boldsymbol{\mathcal{S}}_{\bot}$. For this rule, k is always equal to 0 and there is nothing to show.

$\boldsymbol{\mathcal{S}}_{\neg @}$. There is no (dg, θ) that satisfies $\bot$.

$\boldsymbol{\mathcal{S}}_{\twoheadrightarrow}$, $\boldsymbol{\mathcal{S}}_{\twoheadrightarrow,\sharp}$. Both rules only remove the chain constraint, which is implied by the newly added constraints in all cases. □

Bibliography

[1] M. Abadi, B. Blanchet and C. Fournet. Just fast keying in the pi calculus. *Programming Languages and Systems*, :340–354, 2004.

[2] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 367(1-2):2–32, 2006.

[3] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on principles of programming languages*, , 2001.

[4] G. Acs, L. Buttyan and I. Vajda. Provably Secure On-Demand Source Routing in Mobile Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 5(11):1533–1546, 2006.

[5] S. Al-Riyami and K. Paterson. Tripartite authenticated key agreement protocols from pairings. *Cryptography and Coding*, :332–359, 2003.

[6] R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[7] H. Andréka, I. Németi and J. van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.

[8] M. Arapinis and M. Duflot. Bounding messages for free in security protocols. *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, :376–387, 2007.

[9] M. Arapinis, E. Ritter and M.D. Ryan. Statverif: verification of stateful processes. In *Computer Security Foundations Symposium (CSF), 2011 IEEE 24th*, pages 33–47. IEEE, 2011.

[10] A. Armando, D. Basin, Y. Boichut., Y. Chevalier, Compagna. L., J. Cuellar, P. H. Drielsma, P. C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proceedings of CAV'2005*, LNCS 3576, pages 281–285. Springer-Verlag, 2005.

[11] A. Armando, R. Carbone and L. Compagna. LTL model checking for security protocols. In *csf2010*, pages 385–396. 2007.

[12] A. Armando, R. Carbone, L. Compagna, J. Cuellar, G. Pellegrino and A. Sorniotti. An authentication flaw in browser-based single sign-on protocols: impact and remediations. *Computers & Security*, , 2012.

[13] M. Arnaud, V. Cortier and S. Delaune. Modeling and verifying ad hoc routing protocols. *Computer Security Foundations Symposium, IEEE*, 0:59–74, 2010.

[14] G. Avoine, M. A. Bingöl, S. Kardaş, C. Lauradoux and B. Martin. A framework for analyzing RFID distance bounding protocols. *Journal of Computer Security – Special Issue on RFID System Security*, , 2010.

[15] F. Baader and K. Schulz. Unification in the union of disjoint equational theories: combining decision procedures. *Journal of Symbolic Computation*, , 1996.

[16] M. Backes and B. Pfitzmann. Limits of the brsim/uc soundness of dolev–yao-style xor. *International Journal of Information Security*, 7(1):33–54, 2008.

[17] M. Backes, B. Pfitzmann and M. Waidner. Limits of the brsim/uc soundness of dolev-yao models with hashes. Pages 404–423. Springer, 2006.

[18] C. Bansal, K. Bhargavan and S. Maffeis. Discovering concrete attacks on website authorization by formal analysis. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 247–262. IEEE, 2012.

[19] 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.

[20] D. Basin, S. Capkun, P. Schaller and B. Schmidt. Let's get physical: models and methods for real-world security protocols. In *TPHOLs*, pages 1–22. 2009.

[21] D. Basin, S. Capkun, P. Schaller and B. Schmidt. Formal reasoning about physical properties of security protocols. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):16, 2011.

[22] 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. Sv, 2010.

[23] D. Basin, C. Cremers and S. Meier. Provably repairing the iso/iec 9798 standard for entity authentication. *Principles of Security and Trust*, :129–148, 2012.

[24] M. Baudet, V. Cortier and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. *Information and Computation*, 207(4):496–520, 2009.

[25] K. Bhargavan, C. Fournet, A.D. Gordon and S. Tse. Verified interoperable implementations of security protocols. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31(1):5, 2008.

[26] S. Blake-Wilson, D. Johnson and A. Menezes. Key agreement protocols and their security analysis. *Crytography and Coding*, :30–45, 1997.

[27] 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. Sv, 1999.

[28] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW*, pages 82–96. 2001.

[29] B. Blanchet. A computationally sound mechanized prover for security protocols. In *Security and Privacy, 2006 IEEE Symposium on*, page 15. IEEE, 2006.

[30] B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[31] 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.

[32] B. Blanchet and A. Podelski. Verification of cryptographic protocols: tagging enforces termination. In *Foundations of Software Science and Computation Structures*, pages 136–152. Springer, 2003.

[33] C. Boyd. Security architectures using formal methods. *Selected Areas in Communications, IEEE Journal on*, 11(5):694–701, Jun 1993.

[34] S. Brands and D. Chaum. Distance-bounding protocols. In *EUROCRYPT '93: Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, pages 344–359. Springer-Verlag New York, Inc., 1994.

[35] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, 1999.

[36] R. Canetti and M. Fischlin. Universally composable commitments. In *Advances in Cryptology-Crypto 2001*, pages 19–40. Springer, 2001.

[37] 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. Sv, 2001.

[38] S. Capkun, L. Buttyan and J.-P. Hubaux. SECTOR: secure tracking of node encounters in multi-hop wireless networks. In *SASN '03: Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks*, pages 21–32. New York, NY, USA, 2003. ACM Press.

[39] S. Capkun and M. Cagalj. Integrity regions: authentication through presence in wireless networks. In *WiSe '06: Proceedings of the 5th ACM Workshop on Wireless Security*, pages 1–10. New York, NY, USA, 2006. ACM Press.

[40] S. Capkun and J.P. Hubaux. Secure positioning in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(2):221–232, 2006.

[41] I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of CSFW 1999*, pages 55–69. IEEE, 1999.

[42] S. Chatterjee, A. Menezes and B. Ustaoglu. Combined security analysis of the one-and three-pass unified model key agreement protocols. *Progress in Cryptology-INDOCRYPT 2010*, :49–68, 2010.

[43] 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. Sv, 2011.

[44] L. Chen and C. Kudla. Identity based authenticated key agreement protocols from pairings. In *Computer Security Foundations Workshop, 2003. Proceedings. 16th IEEE*, pages 219–233. IEEE, 2003.

[45] Y. Chevalier, R. Küsters, M. Rusinowitch and M. Turuani. Complexity results for security protocols with diffie-hellman exponentiation and commuting public key encryption. *ACM Transactions on Computational Logic (TOCL)*, 9(4):24, 2008.

[46] Y. Chevalier and M. Rusinowitch. Combining intruder theories. In *ICALP*, pages 639–651. Springer, 2005.

[47] J. Chomicki and D. Toman. Temporal databases. *Foundations of Artificial Intelligence*, 1:429–467, 2005.

[48] E.M. Clarke, S. Jha and W. Marrero. Verifying security protocols with Brutus. *ACM Trans. Softw. Eng. Methodol.*, 9(4):443–487, 2000.

[49] J. Clulow, G. P. Hancke, M. G. Kuhn and T. Moore. So near and yet so far: distance-bounding attacks in wireless networks. In *Security and Privacy in Ad-hoc and Sensor Networks*, pages 83–97. Springer, 2006.

[50] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *Proceedings of the 12th European conference on Programming*, ESOP'03, pages 99–113. Berlin, Heidelberg, 2003. Sv.

[51] H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. *Term Rewriting and Applications*, :294–307, 2005.

[52] H. Comon-Lundh, S. Delaune and J. Millen. Constraint solving techniques and enriching the model with equational theories. *Formal Models and Techniques for Analyzing Security Protocols*, 5:35–61, 2010.

[53] R. Corin, S. Etalle, PH Hartel and A. Mader. Timed analysis of security protocols. *Journal of Computer Security*, 15(6):619–645, 2007.

[54] R. Corin, A. Saptawijaya and Sandro Etalle. A logic for constraint-based security protocol analysis. In *Proceedings of IEEE Symposium on Security and Privacy, S&P 2006*, pages 155–168. 2006.

[55] V. Cortier, J. Degrieck and S. Delaune. Analysing routing protocols: four nodes topologies are sufficient. *Principles of Security and Trust*, :30–50, 2012.

[56] V. Cortier and S. Delaune. Decidability and combination results for two notions of knowledge in security protocols. Pages 1–47. Springer, 2012.

[57] 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.

[58] 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.

[59] C. Cremers and M. Feltz. Beyond eck: perfect forward secrecy under actor compromise and ephemeral-key reveal. *Computer Security–ESORICS 2012*, :734–751, 2012.

[60] C. Cremers, K. B. Rasmussen, B. Schmidt and S. Čapkun. Distance Hijacking Attacks on Distance Bounding Protocols. In *33rd IEEE Symposium on Security and Privacy, S&P 2012*, pages 113–127. IEEE Computer Society, may 2012.

[61] C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Ph.D. dissertation, Eindhoven University of Technology, 2006.

[62] C.J.F. 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.

[63] S. Delaune, S. Kremer, M.D. Ryan and G. Steel. Formal analysis of protocols based on tpm state registers. In *Computer Security Foundations Symposium (CSF), 2011 IEEE 24th*, pages 66–80. IEEE, 2011.

[64] G. Delzanno and P. Ganty. Automatic Verification of Time Sensitive Cryptographic Protocols. *TACAS '04: Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, , 2004.

[65] N. Dershowitz and J.P. Jouannaud. Rewrite systems, Handbook of theoretical computer science (vol. B): formal models and semantics. 1991.

[66] Y. Desmedt, C. Goutier and S. Bengio. Special uses and abuses of the fiat-shamir passport protocol. In *Advances in Cryptology-CRYPTO'87*, pages 21–39. Springer, 2006.

[67] Y. Desmedt, C. Goutier and S. Bengio. Special uses and abuses of the fiat-shamir passport protocol. In *Advances in Cryptology-CRYPTO'87*, pages 21–39. Springer, 2006.

[68] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[69] D. Dolev and A. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[70] D.J. Dougherty and J.D. Guttman. Symbolic protocol analysis for diffie-hellman. *ArXiv preprint arXiv:1202.2168*, , 2012.

[71] S. Drimer and S. J. Murdoch. Keep your enemies close: distance bounding against smartcard relay attacks. In *Usenix '07: Proceedings of 16th USENIX Security Symposium*, pages 1–16. Berkeley, CA, USA, 2007. USENIX Association.

[72] N. Durgin, P. Lincoln, J. Mitchell and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

[73] F. Durán and J. Meseguer. A church-rosser checker tool for conditional order-sorted equational maude specifications. *Rewriting Logic and Its Applications*, :69–85, 2010.

[74] U. Dürholz, M. Fischlin, M. Kasper and C. Onete. A formal approach to distance-bounding rfid protocols. *Information Security*, :47–62, 2011.

[75] S. Erbatur, S. Escobar, D. Kapur, Z. Liu, C. Lynch, C. Meadows, J. Meseguer, P. Narendran, S. Santiago and R. Sasse. Effective symbolic protocol analysis via equational irreducibility conditions. *Computer Security–ESORICS 2012*, :73–90, 2012.

[76] S. Escobar, J. Hendrix, C. Meadows and J. Meseguer. Diffie-Hellman cryptographic reasoning in the Maude-NRL protocol analyzer. 2007. Informal proceedings.

[77] S. Escobar, C. Meadows and J. Meseguer. A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. *Theor. Comput. Sci.*, 367:162–202, 2006.

[78] S. Escobar, C. Meadows and J. Meseguer. State space reduction in the Maude-NRL protocol analyzer. In *Computer Security - ESORICS 2008*, volume 5283 of *lncs*, pages 548–562. Sv, 2008.

[79] S. Escobar, R. Sasse and J. Meseguer. Folding variant narrowing and optimal variant termination. *Journal of Logic and Algebraic Programming*, , 2012.

[80] N. Evans and S. Schneider. Analysing Time Dependent Security Properties in CSP Using PVS. In *ESORICS '00: Proceedings of the 6th European Symposium on Research in Computer Security*, pages 222–237. London, UK, 2000. Springer-Verlag.

[81] M. Fischlin and C. Onete. Provably secure distance-bounding: an analysis of prominent protocols. Technical Report, Cryptology ePrint Archive, Report 2012/128, 2012.

[82] A. Francillion, B. Danev and S. Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Cryptology ePrint Archive: Report 2010/332*. 2010.

[83] F.J.T. Fábrega. Strand spaces: proving security protocols correct. *Journal of Computer Security*, 7(2):191–230, 1999.

[84] S. Ganeriwal, C. Pöpper, S. Capkun and M.B. Srivastava. Secure time synchronization in sensor networks. *ACM Transactions on Information and System Security*, 11(4):23, 2008.

[85] J. Giesl, P. Schneider-Kamp and R. Thiemann. Aprove 1.2: automatic termination proofs in the dependency pair framework. *Automated Reasoning*, :281–286, 2006.

[86] Joseph A. Goguen. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *TCS*, 105:217–273, 1992.

[87] M. González Muñiz and P. Laud. On the (im) possibility of perennial message recognition protocols without public-key cryptography. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 1510–1515. ACM, 2011.

[88] R. Gorrieri, F. Martinelli, M. Petrocchi and A. Vaccarelli. Formal analysis of some timed security properties in wireless protocols. In *FMOODS '03: Proceedings of the 6th IFIP Workshop on Formal Methods for Open Object-based Distributed Systems*, pages 139–154. 2003.

[89] J. Goubault-Larrecq and F. Parrennes. Cryptographic protocol analysis on real c code. In *Verification, Model Checking, and Abstract Interpretation*, pages 363–379. Springer, 2005.

[90] J. Goubault-Larrecq, M. Roger and K.N. 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.

[91] J.D. Guttman. State and progress in strand spaces: proving fair exchange. *Journal of Automated Reasoning*, :1–37, 2011.

[92] G. P. Hancke and M. G. Kuhn. An RFID distance bounding protocol. In *SECURECOMM '05: Proceedings of the 1st International Conference on Security and Privacy for Emerging Areas in Communications Networks*, pages 67–73. Washington, DC, USA, 2005. IEEE Computer Society.

[93] American National Standards Institute. ANSI X9.42. Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography. , , 2003.

[94] P. Jouannaud J and H. Kirchner. Completion of a set of rules modulo a set of equations. 1986.

[95] 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.

[96] 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/`.

[97] A. Joux. A one round protocol for tripartite diffie–hellman. *Algorithmic number theory*, :385–393, 2000.

[98] B.S. Kaliski Jr. An unknown key-share attack on the mqv key agreement protocol. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):275–288, 2001.

[99] 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. Sv, 2003.

[100] D. Kapur, P. Narendran and L. Wang. Undecidability of unification over two theories of modular exponentiation. In *Seventeenth International Workshop on Unification (UNIF-2003), Valencia, Spain*. 2003.

[101] D. Kapur, P. Narendran and L. Wang. A unification algorithm for analysis of protocols with blinded signatures. Pages 433–451. Springer, 2005.

[102] National Institute of Standards and Technology. *SP 800-56B, Special Publication 800-56B, Recommendation for Pair-Wise Key Establishment Schemes using Integer Factorization Cryptography*. National Institute of Standards and Technology, August 2009.

[103] H. Krawczyk. HMQV: a high-performance secure Diffie-Hellman protocol. In *Advances in Cryptology–CRYPTO 2005*, volume 3621 of *lncs*, pages 546–566. Sv, 2005.

[104] S. Kremer, A. Mercier and R. Treinen. Reducing equational theories for the decision of static equivalence. *Advances in Computer Science-ASIAN 2009. Information Security and Privacy*, :94–108, 2009.

[105] M.G. Kuhn. An Asymmetric Security Mechanism for Navigation Signals. *IH 2004: 6th International Workshop on Information Hiding, Revised Selected Papers*, , 2004.

[106] J. Katz and Y. Lindell. *Introduction to modern cryptography*. Chapman Hall , 2008.

[107] R. Künnemann and G. Steel. YubiSecure? formal security analysis results for the Yubikey and YubiHSM. In Audun Jøsang and Pierangela Samarati, editors, *Preliminary Proceedings of the 8th Workshop on Security and Trust Management (STM'12)*. Pisa, Italy, sep 2012.

[108] R. Küsters and T. Truderung. Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In *csf2009*, pages 157–171. IEEE Computer Society, 2009.

[109] R. Küsters and T. Truderung. Reducing protocol analysis with xor to the xor-free case in the horn theory based approach. *Journal of Automated Reasoning*, 46(3):325–352, 2011.

[110] B.A. LaMacchia, K. Lauter and A. Mityagin. Stronger security of authenticated key exchange (eprint). *IACR eprint*, 073, 2006.

[111] B.A. LaMacchia, K. Lauter and A. Mityagin. Stronger security of authenticated key exchange. In *Provable Security*, volume 4784 of *lncs*, pages 1–16. Sv, 2007.

[112] K. Lauter and A. Mityagin. Security analysis of KEA authenticated key exchange protocol. In *PKC 2006*, volume 3958 of *lncs*, pages 378–394. Sv, 2006.

[113] L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28:119–134, 2003.

[114] L. Lazos, R. Poovendran and S. Capkun. ROPE: robust position estimation in wireless sensor networks. *IPSN 2005: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, 2005*, :324–331, 2005.

[115] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. *Tools and Algorithms for the Construction and Analysis of Systems*, :147–166, 1996.

[116] G. Lowe. A hierarchy of authentication specifications. In *CSFW '97: Proceedings of the 10th Computer Security Foundations Workshop (CSFW '97)*, page 31. Washington, DC, USA, 1997. IEEE Computer Society.

[117] 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. Sv, 2004.

[118] S. Malladi, B. Bruhadeshwar and K. Kothapalli. Automatic analysis of distance bounding protocols. *ArXiv preprint arXiv:1003.5383*, , 2010.

[119] T. Matsumoto and Y. Takashima. On seeking smart public-key-distribution systems. *IEICE TRANSACTIONS (1976-1990)*, 69(2):99–106, 1986.

[120] U. Maurer and P. Schmid. A calculus for security bootstrapping in distributed systems. *Journal of Computer Security*, 4(1):55–80, 1996.

[121] D.A. McAllester. Automatic recognition of tractability in inference relations. *Journal of the ACM (JACM)*, 40(2):284–303, 1993.

[122] C. Meadows and P. Narendran. A unification algorithm for the Group Diffie-Hellman protocol. In *Proc. of WITS 2002*. 2002.

[123] C. Meadows, R. Poovendran, D. Pavlovic, L. Chang and P. Syverson. Distance bounding protocols: authentication logic analysis and collusion attacks. *Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks*, :279–298, 2006.

[124] S. Meier. *Advancing Automated Security Protocol Verification*. Ph.D. dissertation, ETH Zurich, 2012. Available `http://www.infsec.ethz.ch/research/software#TAMARIN`.

[125] S. Meier, C. Cremers and D. Basin. Strong invariants for the efficient construction of machine-checked protocol security proofs. In *csf2010*, pages 231–245. IEEE Computer Society, 2010.

[126] S. Meier and B. Schmidt. The Tamarin Prover: source code and case studies. April 2012. Available `http://hackage.haskell.org/package/tamarin-prover-0.4.0.0`.

[127] A. Menezes. Another look at hmqv. *Mathematical Cryptology JMC*, 1(1):47–64, 2007.

[128] A. Menezes and B. Ustaoglu. Security arguments for the um key agreement protocol in the nist sp 800-56a standard. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 261–270. ACM, 2008.

[129] A. Menezes and B. Ustaoglu. On reusing ephemeral keys in diffie-hellman key agreement protocols. *International Journal of Applied Cryptography*, 2(2):154–158, 2010.

[130] D. Micciancio and S. Panjwani. Optimal communication complexity of generic multicast key distribution. *IEEE/ACM Transactions on Networking (TON)*, 16(4):803–813, 2008.

[131] J. Millen and V. Shmatikov. Symbolic protocol analysis with an abelian group operator or diffie-hellman exponentiation. *Journal of Computer Security*, 13(3):515–564, 2005.

[132] S. Mödersheim. Diffie-Hellman without difficulty. In *Proceedings of the 8th International Workshop on Formal Aspects of Security & Trust (FAST)*. 2011.

[133] S. Nanz and C. Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1):203–227, 2006.

[134] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[135] W. Nutt. Unification in monoidal theories. In *Proceedings of the 10th international conference on automated deduction*, volume 449, pages 618–632. Springer, 1990.

[136] T. Nipkow, L.C. Paulson and M. Wenzel. *Isabelle/Hol: A Proof Assistant for Higher-Order Logic*. Springer, 2002.

[137] National Institute of Standards and Technology. SP 800-56A. Special Publication 800-56A, Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. 2006.

[138] A. Pankova and P. Laud. Symbolic analysis of cryptographic protocols containing bilinear pairings. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 63–77. IEEE, 2012.

[139] P. Papadimitratos, M. Poturalski, P. Schaller, P. Lafourcade, D. Basin, S. Capkun and J.P. Hubaux. Secure neighborhood discovery: a fundamental element for mobile ad hoc networking. *Communications Magazine, IEEE*, 46(2):132–139, 2008.

[140] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[141] O. Pereira and J.J. Quisquater. On the impossibility of building secure cliques-type authenticated group key agreement protocols. *Journal of Computer Security*, 14(2):197–246, 2006.

[142] G.E. Peterson and M.E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM (JACM)*, 28(2):233–264, 1981.

[143] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over gf (p) and its cryptographic significance. *IEEE Transactions on information Theory*, , Jan 1978.

[144] M. Poturalski, P. Papadimitratos and J.-P. Hubaux. Secure neighbor discovery in wireless networks: formal investigation of possibility. In *ASIACCS '08: Proceedings of the 3rd ACM Symposium on Information, Computer, and Communications Security*, pages 189–200. ACM, 2008.

[145] Adrian Perrig and J. D. Tygar. *Secure Broadcast Communication in Wired and Wireless Networks*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[146] D. Prawitz. *Natural deduction*. Almqvist & Wiksell, 1965.

[147] C. Pöpper, N. Tippenhauer, B. Danev and S. Capkun. Investigation of signal and message manipulations on the wireless channel. *Computer Security–ESORICS 2011*, :40–59, 2011.

[148] M. Rabi and A.T. Sherman. Associative one-way functions: a new paradigm for secret-key agreement and digital signatures. *University of Maryland at College Park, MD, USA*, :13, 1993.

[149] R. Ramanujam and S. Suresh. Tagging makes secrecy decidable with unbounded nonces as well. *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science*, :363–374, 2003.

[150] J.D. Ramsdell and J.D. Guttman. Cpsa: a cryptographic protocol shapes analyzer. *Hackage. The MITRE Corporation*, 2(009), 2009.

[151] K. B. Rasmussen and S. Capkun. Realization of RF distance bounding. In *Proceedings of the USENIX Security Symposium*. 2010.

[152] K. B. Rasmussen, C. Castelluccia, T. S. Heydt-Benjamin and S. Capkun. Proximity-based access control for implantable medical devices. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009.

[153] J. Reid, J. M. Gonzalez Nieto, T. Tang and B. Senadji. Detecting relay attacks with timing-based protocols. In *ASIACCS '07: Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*, pages 204–213. ACM, 2007.

[154] R. Rivest, A. Shamir and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, , Jan 1978.

[155] E.K. Ryu, E.J. Yoon and K.Y. Yoo. An efficient id-based authenticated key agreement protocol from pairings. Pages 1458–1463. Springer, 2004.

[156] N. Sastry, U. Shankar and D. Wagner. Secure verification of location claims. In *WiSe '03: Proceedings of the 2003 ACM workshop on Wireless security*, pages 1–10. New York, NY, USA, 2003. ACM Press.

[157] B. Schmidt, S. Meier, C. Cremers and D. Basin. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties (Extended Version). April 2012. Available `http://www.infsec.ethz.ch/research/software#TAMARIN`.

[158] B. Schmidt and P. Schaller. Isabelle Theory Files: Formal Reasoning about Physical Properties of Security Protocols. 2010. Available `http://www.infsec.ethz.ch/research/software/protoveriphy`.

[159] M. Scott. Authenticated id-based key exchange and remote log-in with simple token and pin number. *IACR eprint*, 164, 2002.

[160] S. Sedihpour, S. Capkun, S. Ganeriwal and M. Srivastava. Implementation of attacks on ultrasonic ranging systems (demo). *Proceedings of the ACM Conference on Networked Sensor Systems (SenSys)*, , 2005.

[161] R. Sharp and M.R. Hansen. Timed Traces and Strand Spaces. *Proceedings of the 16th Nordic Workshop on Programming Theory*, :96–98, 2004.

[162] 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.

[163] C. Staub. A user interface for interactive protocol design. 2011. Available `http://www.infsec.ethz.ch/research/software#TAMARIN`.

[164] K. Sun, P. Ning and C. Wang. TinySeRSync: secure and resilient time synchronization in wireless sensor networks. *CCS '06: Proceedings of the 13th ACM conference on Computer and Communications Security*, :264–277, 2006.

[165] A. Tiu and R. Goré. A proof theoretic analysis of intruder theories. In *Rewriting Techniques and Applications*, pages 103–117. Springer, 2009.

[166] B. Ustaoglu. Comparing session state reveal and ephemeral key reveal for Diffie-Hellman protocols. *Provable Security*, :183–197, 2009.

[167] R. Wang, S. Chen, X.F. Wang and S. Qadeer. How to shop for free online–security analysis of cashier-as-a-service based web stores. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 465–480. IEEE, 2011.

[168] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. *Automated Deduction CADE-16*, :70–70, 1999.

[169] S. Yang and J.S. Baras. Modeling vulnerabilities of ad hoc routing protocols. In *SASN '03: Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks*, pages 12–20. New York, NY, USA, 2003. ACM.

[170] Yubikey USB-key. Http://www.yubico.com/yubikey.