Sound Abstractions for Security Protocol Verification

Doctoral Thesis

Author(s):
Nguyen, Binh Thanh

Publication date:
2015

Permanent link:
https://doi.org/https://doi.org/10.3929/ethz-a-010515033

DISS. ETH NO. 22674

Sound Abstractions for Security Protocol Verification

A thesis submitted to attain the degree of

DOCTOR OF SCIENCE of ETH ZÜRICH

(Dr. sc. ETH Zürich)

presented by

Binh Thanh Nguyen

M.Sc. in Computational Logic, Technische Universität Dresden
born on 25.03.1984
citizen of Vietnam

accepted on the recommendation of

Prof. Dr. David Basin, examiner
Dr. Christoph Sprenger, co-examiner
Prof. Dr. Srdjan Capkun, co-examiner
Prof. Dr. Sebastian Mödersheim, co-examiner

2015

Abstract

Security protocols (or cryptographic protocols) are programs that aim at establishing secure communication in hostile environments. Various applications rely on these protocols such as e-commerce, e-banking, mobile services, and sensor networks. The design of cryptographic protocols that are secure is notoriously hard and error-prone. This motivates research on methods for developing such protocols in a systematic and incremental manner and for their formal analysis. Although there exist a number of efficient security protocol analysis tools, the verification of the security of real-world protocols such as TLS and IPsec remains a challenging task due to their complexity and the undecidability of the verification problem.

In this thesis, we present a large class of security protocol abstractions with the principal aim of improving the scope and efficiency of verification tools. Our abstractions can also serve to check the correctness of refinement steps during the incremental development of protocols. We extend the state of the art in several directions. We propose typed abstractions, which transform a term's structure based on its type, and untyped abstractions, which remove atomic messages, variables, and redundant terms. Our theory improves on previous work by supporting a useful subclass of so-called *shallow* subterm-convergent rewrite theories, user-defined types, and untyped variables to cover type flaw attacks. We prove soundness results for an expressive property language that includes secrecy and various authentication properties. We design a heuristic-based algorithm that automatically abstracts security protocols before feeding them to security protocol verification tools. We have implemented this algorithm for the Scyther tool and validated our approach on realistic IETF protocol models. In most cases, we either achieve dramatic speedups or extend Scyther's scope by enabling verification tasks that previously exceeded time or memory limits. We have also (manually) applied our method to improve the performance of several other modern security protocol analyzers.

Zusammenfassung

Sicherheitsprotokolle (oder kryptographische Protokolle) sind Programme, deren Ziel es ist, sichere Kommunikation in feindlichen Umgebungen zu ermöglichen. Verschiedene Anwendungen stützen sich auf diese Protokolle, zum Beispiel e-Kommerz, e-Banking, mobile Dienste und Sensornetzwerke. Die Konzeption von sicheren kryptographischen Protokollen ist offenkundig schwierig und fehleranfällig. Dies motiviert die Erforschung von Methoden zur systematischen und inkrementellen Entwicklung solcher Protokolle sowie zu deren formalen Analyse. Obwohl es mehrere effiziente Werkzeuge zur Analyse von Sicherheitsprotokollen gibt, bleibt die Verifikation von realistischen Protokollen wie TLS und IPsec wegen deren Komplexität und der Unentscheidbarkeit des Verifikationsproblems eine herausfordernde Aufgabe.

In dieser Dissertation präsentieren wir eine grosse Klasse von Abstraktionen von Sicherheitsprotokollen mit dem Hauptziel der Verbesserung des Anwendungsbereichs und der Effizienz von Verifikationswerkzeugen. Unsere Abstraktionen können aber auch zur Überprüfung von Verfeinerungsschritten bei der inkrementellen Entwicklung von Protokollen dienen. Wir erweitern den aktuellen Stand der Forschung in mehrere Richtungen. Wir präsentieren getypte Abstraktionen, welche die Struktur einer Nachricht basierend auf ihrem Typ transformieren, und ungetypte Abstraktionen, welche atomische Nachrichten, Variablen und redundante Nachrichten entfernen. Unsere Theorie verbessert und erweitert vorherige Arbeiten indem sie eine nützliche Unterklasse von sogenannten *flachen* subterm-konvergenten Termersetzungstheorien sowie benutzerdefinierte Typen und ungetypte Variablen zur Abdeckung von Typfehler-Angriffen unterstützt. Wir beweisen die Korrektheit unserer Abstraktionen für eine ausdrucksstarke Sprache zur Spezifikation von Sicherheitseigenschaften einschliesslich Vertraulichkeit und verschiedener Authentisierungseigenschaften. Wir entwerfen einen heuristik-basierten Algorithmus, welcher Sicherheitsprotokolle automatisch abstrahiert bevor sie von Verifikationswerkzeugen überprüft werden. Wir haben diesen Algorithmus für das Scyther Werkzeug implementiert und unsere Methode anhand von realistischen IETF Protokollen validiert. In den meisten Fällen erreichen wir entweder ein dramatische Beschleunigung der Verifikation oder wir erweitern den Anwendungsbereich von Scyther, indem wir Verifikationsaufgaben ermöglichen, welche bisher an Zeit- oder Speichergrenzen gestossen sind. Wir haben unsere Abstraktionen auch (manuell) angewendet um die Leistung anderer moderner Verifikationswerkzeuge zu verbessern.

Acknowledgments

First and foremost, I would like to express my gratitude to everyone who has supported throughout my PhD. I express my special thanks to my supervisor David Basin for giving me the opportunity of pursuing research at ETH, and for his aspiring guidance and constructive criticism on my work. I am sincerely grateful to my supervisor Christoph Sprenger for encouraging my research and supporting me to get through my tough times. He has been tremendously patient with my poor English and provided me with many useful advices for improving my presentation skills. Without his endless support, I could not have finished up my degree. I would like to take this opportunity to thank Mathieu Turuani and Michael Rusinowitch for the productive and effective collaboration we had on the topic of my work. Many ideas for extending the work were coming out from our discussions. I would also like to thank Srdjan Capkun and Sebastian Mödersheim for their willingness to serve as my committee members.

Secondly, I would especially like to thank Ognjen Maric for his great help not only with my research questions, but also with my daily life problems. I would like to thank Cas Cremers for his help with Scyther. I thank Ralf Sasse and Son Thai Hoang for their careful proof-reading and valuable feedback on my work. I would like to thank Benedikt Schmidt, Simon Meier, Michèle Feltz, Bruno Conchinha Montalto, Sasa Radomirovic, Ralf Sasse, Jannik Dreier, Andreas Lochbihler, Grgur Petric Maretic, and my office mate Petar Tsankov for many insightful discussions and ideas. I would like to thank Barbara Pfändner for her help with administrative issues. I would also like thank Claudio Soriente for organizing weekly football game that has had given me a lot of fun and enjoyable moments after work.

Lastly, I would like to thank my friends in Zürich for helping me to get balance between academics and social life. Most importantly, I would like to express my deepest thanks to my parents and my brother, and in particular my beloved wife Voc and my son Bao Nam for their unconditional love and encouragement.

Contents

1. Introduction

Since the mid-1990s, the Internet has had an enormous impact on every aspect of human life. Together with the rise of the Internet, thousands of online services have been deployed, such as electronic banking. Such services often require certain security guarantees. For instance, an Internet banking system must ensure the privacy of sensitive information about its customers and transactions. This motivates the use of *security protocols* to protect communication among participants in an insecure environment.

Security protocols are computer programs that use cryptographic primitives such as digital signatures, hash functions, and encryptions, to achieve certain security goals such as confidentiality and authenticity, even in the presence of an adversary who controls the network. For instance, the TLS (Transport Layer Security) protocol [52] was designed to provide privacy and data integrity between two communicating participants. The design of security protocols is typically hard and error-prone. Historically, there are protocols that have been existing for many years before attacks were found. For instance, Lowe [86] found an attack on the Needham-Schroeder public-key protocol [101] using the model checker FDR [114, 23], 17 years after its publication. Various serious attacks have also been found on the well-known TLS protocol [96]. Despite this issue, security protocols are increasingly used in today's networked applications. This gives rise to many research topics on the design and analysis of cryptographic protocols. Since 1980s, there has been much progress on establishing solid and rigorous mathematical foundations for analyzing security protocols. Two main lines of research have evolved based on two distinct models, namely the *symbolic* (or the *formal*) and the *computational* (or the *cryptographic*) models.

The symbolic (or the formal) model introduced by Dolev and Yao [54] represents messages by symbolic terms from a term algebra. Cryptographic primitives are represented by function symbols and treated as black boxes. For example, symmetric encryption is modeled by the functions symbols $\{\!|\cdot|\!\}_{\cdot}^{-1}$ and $\{\!|\cdot|\!\}_{\cdot}$. The term $\{\!|m|\!\}_k$ represents the symmetric encryption of the message m with the key k and $\{\!|m|\!\}_k^{-1}$ represents the symmetric decryption of m with k. Algebraic properties of cryptographic operators can be modeled by equations. For example, the equation $\{\!|\{\!|m|\!\}_k|\!\}_k^{-1} = m$ can be used to model the inversion of symmetric encryption, i.e., the attacker learns the message m if he knows the encryption $\{\!|m|\!\}_k$ and the key k. Additionally, this model relies on *the perfect cryptography assumption*, i.e., all cryptographic primitives are perfectly secure. This assumption significantly restricts the adversary's capabilities. For example, the adversary does not learn anything from an encrypted message unless he knows the right decryption key. Due to the high level of abstraction, this approach is well-suited for automation. Many tools have been designed and developed for the formal analysis of security protocols such as ProVerif [19], OFMC [15], CL-Atse [122], Tamarin [94], and

Scyther [41].

The computational (or the cryptographic) model developed by Goldwasser, Micali, Rivest, Yao, and others [62, 63, 125], represents messages by bitstrings and the adversary is an arbitrary probabilistic polynomial-time Turing machine. Cryptographic primitives are probabilistic functions from bitstrings to bitstrings. The security of cryptographic protocols is defined as games played by the adversary. The protocol is considered secure if the adversary has only a negligible advantage of winning this game. A proof of a security statement is often achieved by reducing the security game to solving a computationally hard mathematical problem such as the factorization of large composite numbers or computing discrete logarithms. That is, one proves by contradiction that if the adversary has non-negligible probability of winning the game then he can solve the hard problem, which supposedly has no efficient solution. Cryptographic proofs such as those for the key exchange protocols in [84, 83, 95, 79] are usually complex, laborious, and highly error prone. Several tools have been developed to perform proofs directly in the computational model [13, 22]. These tools however offer very limited level of automation. Some of them allow users to interactively construct proofs, but require a considerable degree of expertise.

On the one hand, the computational model allows more precise modeling of cryptographic operations, but the lack of automation is a substantial obstacle. On the other hand, the symbolic model is considerably simpler and greatly benefits from machine support, but does not allow us to establish cryptographic guarantees. Recently, another line of research has been devoted to bridging the gap between these two models, i.e., establishing sufficient conditions under which security properties proved in the symbolic model also hold in the computational world. Research in this direction was initiated by Abadi and Rogaway [92]. In this work, they have shown that two messages that are symbolically equivalent are computationally indistinguishable. Their setting is restricted to symmetric (shared-key) encryption and a passive adversary who only eavesdrops on the network. Since this paper, various results have been established for different cryptographic primitives and stronger adversaries [44, 71, 45, 36, 113, 35, 76, 60, 12, 34] (see [123] for a detailed survey).

1.1. Motivation

As mentioned above, the development and verification of security protocols remains a remarkably difficult task. Past experience has amply shown that informal arguments justifying the security of such protocols are insufficient. This makes security protocols prime candidates for formal verification. In the last two decades, research in formal security protocol verification has made enormous progress, which is reflected in many state-of-the-art tools. These tools can verify small to medium-sized protocols in a few seconds or less, sometimes for an unbounded number of sessions. Despite this success, they can still be challenged when verifying real-world protocols such as those defined in standards and deployed on the internet (e.g., TLS [52], IKE [32], and ISO/IEC 9798 [73, 74, 75]). Such protocols typically have messages with numerous fields, support many alternatives (e.g., cryptographic setups), and may be composed from more basic protocols. Furthermore, termination cannot be achieved in general, as the symbolic verification of security protocols is undecidable [56, 121]. No tool is therefore capable of verifying all security protocols. To improve the efficiency of security protocol verification tools and possibly extend their scopes, *security protocol abstraction* has been introduced.

Abstraction [37] is a standard technique to over-approximate complex systems by simpler ones for verification. Sound abstractions preserve counterexamples (or attacks in security terms) from concrete to abstracted systems. This allows us to reduce the verification of concrete systems to that of abstracted ones, which is supposed to be much simpler. In the context of security protocols, abstractions are extensively used. Here, we only mention a few examples. First, the Dolev-Yao model is a standard (not necessarily sound) abstraction of cryptography. Second, many automated security protocol verification tools use abstractions to map the verification problem into the formalism of an efficient solver or reasoner. For example, ProVerif [19, 20] translates models in the applied pi calculus [1] to a set of Horn clauses. SATMC [9] reduces protocol verification to SAT solving. Paulson [105] models security protocols as inductively defined trace sets. He specifies security properties as predicates on protocol traces and uses the Isabelle/HOL theorem prover [104] to establish these properties. Finally, some abstractions aim at speeding up automated analysis by simplifying protocols within a given protocol model before feeding them to verifiers [72, 103].

Apart from the verification issues, the development of security protocols is an important concern in the security research community. The design of security protocols often goes through three main phases: (i) specifying security requirements, (ii) developing a concrete security protocol description based on the security requirements, and (iii) verifying whether the protocol actually satisfies the security goals. A specification of security requirements typically states what it means for a protocol to be correct. For example, a security protocol designer wishes to develop a protocol that allows two principals to communicate with each other over a hostile network. In order to secure the communication, the principals share a secret key generated by a mutually trusted third party. Security requirements of such protocols (also known as server-based key transport protocols) should include two crucial properties, i.e., the secret key remains unknown to the adversary and each principal knows whom he shares this key with. The later property is also known as *mutual authentication*.

Up to now, formal methods for security protocols have been mainly applied to the verification of completed security protocol designs. Methods for developing security protocols have received much less attention. Design methods in which security protocol designers go directly to concrete security protocol models from the security requirements, without going through intermediate protocol constructions, often leave a fundamental gap between security criteria and cryptographic implementations. There is no guarantee that the concrete protocol fulfils the security requirements. Furthermore, these design methods only allow one to detect flaws only when the protocol has been fully developed. Thus once a new attack has been found, the designers may have to adapt the whole protocol construction in order to eliminate the attack, which is extremely expensive and time-consuming.

Recently, several works have concentrated on developing correct-by-construct frameworks to reason about properties of security protocols in a systematic and hierarchical manner [17, 16, 28, 48, 31, 107, 118, 65, 66, 119, 110]. These works propose different methodologies for developing security protocols based on stepwise refinement, which is a design technique for developing (low-level) concrete systems from (high-level) abstract specifications [3]. The key idea of this method is to progressively refine an abstract security protocol model by adding information, e.g., protocol principals, messages, and cryptographic primitives in order to obtain a more concrete one. The new protocol is as secure as its predecessor with respect to a relevant class of security properties, provided that every refinement step preserves these properties. As a consequence, establishing the security for the abstract protocol is sufficient to conclude the security of the concrete one. In general, proving the soundness of a refinement step involves specifying an abstraction invariant or a simulation relation that relates the abstract specification and the corresponding concrete implementation.

Unlike one-shot design, developing security protocols by refinement has a number of advantages. First, it allows us to gain more insights into the source of security issues by exploiting the failure of refinement steps. This is illustrated by Lowe's attack [86], in which he used the model-checker FDR to generate counterexamples from incorrect refinements. Second, it avoids repeating security proofs. In particular, only security properties that are not yet verified in the abstract model need to be analyzed in the concrete one. This significantly simplifies the verification of concrete models. Third, it enables the development of entire protocol families whose members share structure and properties. Lastly, it allows us to relate different security protocols, e.g., protocol variants in standards, and understand their differences.

It turns out the security protocol abstraction and security protocol refinement are closely related. They correspond to the bottom-up and top-down views on (the same) protocol transformations. Refinement must be property-preserving, or equivalently, abstraction must be attack-preserving. As a consequence, security protocol abstraction can be used not only for improving the performance of security protocol verifiers, but also for developing security protocols by refinement.

In this thesis, we develop a protocol abstraction framework that allows us to significantly simplify security protocols and improve the performance of automated security protocol verification tools. We have validated our approach on a large class of real-world case studies. Even though the thesis focuses on protocol abstractions, one can also justify the correctness of security protocol refinements using our results.

1.2. State of the Art

In this section, we present an overview of existing approaches to employing security protocol abstraction/refinement for the design and analysis of security protocols.

1.2.1. Security Protocol Abstraction

A prominent application of abstraction is to simplify security protocols for their verification. This has been the main motivation for various approaches in this area.

Burrows et al. [26] define a logic for the analysis of security protocols. They analyze several existing authentication protocols by transforming these protocols to idealized forms. In particular, each protocol message is transformed into a logical formula which expresses the essential logical meaning of the data contained in the message. The idealized protocol is then annotated with assumptions associated with the original protocol. Thereby, verifying a security protocol boils down to proving some logical formulas hold, which can be achieved by using a proof system. The authors have used this method to verify various authentication protocols. Unfortunately, their logic lacks a formal semantics and furthermore exhibits a significant flaw [102, 120]. In contrast to their work, we formally establish our soundness results for a large class of protocol abstractions. We show that if there is an attack in the original protocol then there is a corresponding one in the abstract protocol.

Backes et al. [11] present an approach to analyzing authentication protocols using security protocol abstractions. They formalize these protocols in the ρ-spi calculus [25], which is derived from the spi calculus [2] with additional authentication-specific features. The crucial ingredient in their abstraction framework is the introduction of *security labels* together with a partial order on these labels. Typically, security labels represent the capability of reading and writing messages based on the cryptographic primitives being used in these messages, e.g., digital signatures are readable by everyone, but writable only by the sender. To analyze authentication properties of an authentication protocol, they identify challenge-response messages in the protocol specification and abstract these messages into challenges or responses specified in the CR calculus with lower security labels. Roughly speaking, the CR calculus is similar to the ρ-spi calculus but contains specific constructs and primitives for reasoning about challenge-response mechanisms. Their soundness results are based on simulation of protocol executions, i.e., every concrete computation has an abstract counterpart. To analyze abstract CR protocols, they introduce an *effect system* that keeps track of the generation and reception of challenges and responses and the freshness of nonces. As their abstraction functions depend not only on messages to be transformed, but also on the semantics of these messages, automating the abstraction process becomes a non-trivial task. Furthermore, their abstraction framework is tailored to authentication protocols and authentication properties. It is unclear how the theory can be adapted to cover more protocols or security properties. Unlike their work, we prove our soundness results for a relevant class of security properties including secrecy and authentication. Our property language can also be used to specify various authentication properties such as non-injective agreement and synchronization [87, 43, 40].

Hui and Lowe [72] present a library of security protocol abstractions for improving the performance of the CASPER/FDR model checker. They specify security protocols and their abstractions as CSP processes [70]. The security properties of concern are tailored to fixed secrecy and authentication properties. Their abstraction framework provides several ways to modify protocol messages and structures, e.g., removing encryptions, hash functions, atomic or hashed fields, and redirecting messages. They have formalized sufficient conditions for a protocol abstraction to be sound. However, these conditions need to be individually established for each type of security protocol abstraction. They only consider atomic keys and establish their soundness results only for ground messages. In this work, they also present a tool support for protocol transformations. However, these transformations must be given as an input to the tool.

Another line of research in security protocol abstraction is to study the abstraction relationship between different protocol models. Blanchet [21] relates two models of security protocols, namely the linear logic model [56] or the multiset rewriting model [50, 29], and the classical logic, Horn clause representation of security protocols [124, 19]. He shows that the latter model is a sound abstraction of the former with respect to secrecy properties. More precisely, every attack against secrecy present in the classical logic model is also present in the linear logic model. However, no soundness result is established for authentication. The question that whether these models are equivalent is still left open.

Mödersheim [100] studies the relation between Paulson's message trace model and the set rewriting model. In Paulson's message trace model [105], security protocols are defined inductively as sets of traces. Proving the correctness of security protocols is carried out by induction. In the set rewriting model, security protocols are specified as a set of set rewrite rules. The initial state for the set rewriting model consists of facts about the initial knowledge of the intruder. The initialization of new protocol run with respect to a protocol role is modeled by a rule that has no left-hand side and creates a local state of an agent executing that role. This model also contains rules for modeling protocol steps and deduction rules of the intruder. The author shows that the message trace model is indeed an over-approximation of the set rewriting model, i.e., every reachable state in the set rewriting model of a protocol corresponds to a reachable trace in the message trace model. He also points out that this over-approximation is strict, i.e., there are traces in the message trace model that do not have counterpart in the set rewriting model. Based on these results, he proves that the over-approximation is sound with respect to secrecy properties, i.e., secrecy verified in the message trace model also holds in the set rewriting model. He introduces special events (or signals) inspired by Lowe's non-injective agreement [87] to specify authentication properties and prove the soundness results for these properties. However, as his formalism for security properties does not support reasoning about the order of protocol events, synchronization properties as defined in [40] cannot be specified.

Protocol abstraction has been used not only for improving the performance of security protocol verification tools, but also for improving the reliability of security proofs. In this line of research, Brucker and Mödersheim [24] present an integration of the Open-source Fixed-point Model-Checker OFMC [99] and the interactive theorem prover Isabelle [104]. On the automated side, they employ two types of protocol abstraction, namely data and control abstractions, to handle infinite search spaces. Data abstraction involves partitioning the set of freshly created data, e.g., nonces, into finitely many equivalent classes. In particular, the abstract protocol uses the same fresh value for all protocol sessions instead of a really fresh one. Control abstraction involves ignoring the trace structure, i.e., the order of protocol events, and concerns the set of events instead of traces. The abstract protocol is fed into OFMC for verification. If no attack is found then OFMC generates a proof of the correctness of the original protocol which is then checked by the Isabelle core. Otherwise, OFMC refines the abstraction and restarts the verification procedure. Unlike our approach, their abstractions do not guarantee soundness.

From the theoretical point of view, the works [21, 100, 24] are incomparable to ours since they are concerned with the relation of different protocol models while our abstractions involve transforming cryptographic protocols within the same protocol model. From the practical point of view, our approach covers a larger class of security properties than those supported in [21, 100, 24]. In particular, these approaches assume fixed signatures while we work on a more general setting, e.g., we support a subset of subterm-convergent rewrite theories to model cryptographic primitives.

Even though various methods have been proposed for simplifying security protocols, existing approaches either do not discuss a tool implementation that fully automates the abstraction process, or have not been sufficiently validated. Most of them do not provide experimental results. In [72], the authors present only few experimental results for a single tool using abstractions that are specified manually. Brucker and Mödersheim [24] implement their abstraction/refinement framework for the OFMC tool and validate their approach on a small number of security protocols. In this thesis, we have extended Scyther [41] with fully automated support for our abstraction methodology. We validated our approach on an extensive set of realistic case studies drawn from the IKEv1, IKEv2, ISO/IEC 9798, and PANA-AKA standard proposals. We cover a larger number of protocols than [24] does. The protocols we consider are also more complex than those in [24] in terms of the sizes of protocol messages and the number of protocol events.

1.2.2. Security Protocol Refinement

The design and analysis of security protocol by refinement has been mostly studied in the last decade. Several approaches have been taken with different formalisms for refinement.

Bieber et al. [18, 17] propose a formal development of authentication protocols by refinement using the B method [4, 5]. They model security protocols in terms of B abstract machines. Each B abstract machine consists of state variables which model states of the machine, and a set of operations which may modify those variables. Security properties are specified as invariants on state variables. The authors have applied refinement to develop an authentication protocol from the most abstract model which captures a standard authentication property. In the first refinement, they introduce protection mechanisms for enclosed clear-text messages against malicious agents, however no cryptography is used. The second refinement implements authenticated communication using cryptographic keys. They later refine the key distribution phase to obtain Needham-Schroeder [101] and Yahalom [26] style protocols. They however consider a limited adversary model. In particular, they only allow the adversary to learn a message by receiving, listening, and blocking that message, but not by exploiting the algebraic properties of cryptographic primitives.

Using a different formalism of refinement, Bella and Riccobene [16] present a specification of the Kerberos authentication system [97] based on a distributed version of Gurevich's abstract state machines [64]. In their framework, security protocols are specified as a sequential program which describes the operational behaviours of a security protocol in terms of state transitions. Security properties of the protocol are abstractly formalized as predicates and program rules. These elements are then enriched by refinement. They consider two models of adversaries who can intercept messages and steal a session key shared with the Ticket Granting Server (the first model) or with an end server (the second model). The authors apply their method to verify several authentication properties, i.e., an honest client can authenticate itself to the authenticated server and share a session key with the end server. Like [18, 17], their adversary models are rather restricted. For instance, these adversaries are not allowed to alter the messages sent on the network.

Lynch [90] proposes a simulation-based technique for verifying security protocols. Her modeling of security protocols relies on the I/O automaton model for distributed algorithms [91, 89]. In this work, she verifies a combination of a simple shared-key communication protocol and the Diffie-Hellman protocol [53] in the presence of a passive adversary. The correctness of each subprotocol is obtained by establishing a simulation relation between the protocol and its abstract specification. Later, she proves the correctness of the combination using general theorems about automaton composition.

Unlike state-based approaches [18, 17, 16, 90], Butler [27] takes an event-based view of systems specified in B, i.e., each operation with its input and output values define an event. He formalizes security protocol refinement based on B abstract machines to verify safety properties of security protocols. These properties are specified in terms of event traces. He uses his method to derive the fixed Needham-Schroeder protocol [86] from an abstract specification which models authenticated nonce sharing. He has also pointed out a hack on the original Needham-Schroeder protocol by inspecting unsound refinement steps. However, his adversary model is defined ad-hoc and has restricted behaviours, i.e., the intruder can only learn nonces that are encrypted with his own public key, and replay messages that are previously sent by honest agents.

Extending the prior work [27], Butler [28] combines the B method and Hoare's CSP [70] in his refinement framework. The use of CSP notation enables the modeling of the ordering of events in a more convenient way. In this work, he addresses a non-trivial issue of choosing refinement relation used to prove the correctness of refinement steps. In particular, he proposes an incremental refinement technique for deriving these invariants in a more structured and systematic way, while in many other approaches such as [18, 17, 90] abstraction invariants are derived in an ad-hoc manner. He slightly extends the adversary model in [27] by allowing the intruder to learn nonces that are not critical, i.e., those are currently used by the protocol. Nevertheless, the intruder is not allowed to decrypt messages sent by honest agents even if he has access to the keys.

None of the previous approaches [18, 17, 16, 90, 27, 28] considers a full Dolev-Yao intruder in their refinement frameworks. Overcoming this limitation, Sprenger and

Basin [119] propose a hierarchical development method for security protocols based on stepwise refinement that spans several levels of abstraction. Each development starts from abstract models of security properties and proceeds down to cryptographic protocols secure against a Dolev-Yao intruder. The development process traverses intermediate levels of abstraction based on message-less protocols and communication channels with authenticity and confidentiality properties. They have formalized the refinement framework and verified their results using the Isabelle/HOL theorem prover [104]. Using this method, they have developed families of authentication and key transport protocols. Their refinement proofs require establishing appropriate simulation relations which requires considerable expertise. In our work, we provide syntactic criteria for our soundness conditions. This allows us to justify the correctness of refinement steps once-and-for-all.

Datta et al. [48, 49] propose an abstraction and refinement methodology to develop security protocols formalized in Cords [106]. They extend the previous works [46, 47] with support for higher-order features, which enables the use of *protocol templates* with function variables representing cryptographic operations to abstractly specify classes of security protocols. Here abstraction means selecting an appropriate protocol template and refinement means replacing the template's function variables with cryptographic operations and discharging the associated assumptions. They employ a protocol logic and proof system [58, 46, 57, 47] to prove security properties of protocol templates. Security proof of a concrete protocol, i.e., an instance of a protocol template, follows from that of the template, provided that the instantiated cryptographic operations satisfy the assumed hypotheses. Their abstraction and refinement framework also supports combining two different protocol templates to derive security protocols that inherit properties from both. Using this methodology, they have derived a number of protocols from the ISO and IKE families. The correctness of these derivations are however only justified informally. Using our syntactic criteria, we can, for instance, justify the soundness of the message refinements in [49, Section 3.3] in our model.

Pavlovic et al. [31] develop a logic that supports reasoning about authentication properties of security protocols. The logic is closely related to [48], but much simpler. The authentication concepts are specified in terms of partial orders of actions, e.g., if a particular action occurs then some other actions must have been present before it and in a particular (partial) order. Their formalisms of authentication properties encapsulate secrecy requirements as assumptions. These assumptions are not established by the logic itself, but by a different verification method. This clean separation of authentication and secrecy proofs greatly simplifies their analysis. Unlike other approaches in which refinement is driven by properties of abstract protocols, their refinement is guided by properties of concrete protocols. In particular, they first formalize the properties of an abstract protocol based on its construction. Then, they identify missing components for the protocol to meet the expected properties. Finally, they apply an appropriate refinement that provides the required information. Their approach scales poorly as each refinement requires a separate proof for its justification. Applying our results, we can justify the soundness of the protocol derivation tree in [31, Figure 1] at the syntactic level.

Pavlovic and Meadows [107] extend [31] to support proving secrecy properties for key establishment protocols. Their approach is based on similar ideas as [48], i.e., they refine protocols by replacing function variables with cryptographic operations. Abstract protocols a specified with a set of axioms which express secrecy and authentication assumptions. These axioms then need to be discharged through refinement and concrete implementation of abstract specifications. This may require sufficient conditions on the replaced cryptographic operations and the algebraic structures being used. These conditions are typically hard to verify.

The refinement methods introduced in [48, 49, 31, 107] all involve transforming security proofs from abstract specifications to the concrete cryptographic implementations. Unfortunately, these approaches lacks a formal semantics. Furthermore, their results are only applied to particular protocols and refinements. This makes the proposed refinement methods unsuitable for automation. Our approach allows to efficiently verify the soundness of security protocol refinement at the syntactical level, and therefore paves the way for tool implementation that automatically checks the correctness of refinements.

Recently, Guttman [65, 66] studied the preservation of security properties for a rich class of protocol transformations in the strand space model [67]. A transformation is a mapping from nodes on roles of a source protocol to that of a target protocol. The verification of security protocols involves several analysis steps. Each step aims at finding possible protocol shapes called *skeletons* that are compatible with a given initial set of strands. These skeletons are automatically generated by the CPSA tool [112]. The soundness of protocol transformations is based on the simulation of protocol analysis steps instead of execution steps. More precisely, he shows that for each analysis step in the refined protocol, there is a corresponding counterpart in the abstract protocol. As this simulation relation can be used to simulate every attack of the concrete protocol in the abstract one, he is able to establish property preservation for his class of protocol transformations. However, he does not provide a syntactic soundness check as we do.

1.3. Contributions

Extending Hui and Lowe's work [72], we proposed in [103] a class of syntactic protocol abstractions and proved its soundness for a wide range of security properties including secrecy and authentication. We used the type system of Arapinis and Duflot [6] to achieve a fine-grained control over the message-based abstractions. We are able to justify the soundness of all protocol transformations defined in [72] in a more general setting. Nevertheless, our work has a number of limitations. For example, neither can we precisely model security protocols that employ algebraic properties of cryptographic primitives such as Diffie-Hellman key exchange protocols, nor cover type-flaw attacks. Furthermore, we only provide few experiments with a single protocol veficiation tool. Analogous to previous approaches, the work in [103] does not provide checkable conditions for the soundness results. The thesis extends our previous work [103] to address all these limitations. The results of the thesis have been published in:

B.T. Nguyen and C. Sprenger. *Abstractions for security protocol verification.* In R. Focardi and A. Myers (Eds.): POST 2015, LNCS 9036, pp. 196–215, 2015.

In this thesis, we make the following contributions. First, our work provides a sound formal underpinning for protocol transformations, which can serve as a foundation for rigorous security protocol development by refinement as well as for the abstraction of existing protocols. As an example of the latter, our approach helps to improve the performance of various security protocol verifiers, namely Scyther [41], CL-Atse [122], OFMC [15], SATMC [9], and ProVerif [19]. Second, we extend existing work including [103] with respect to the expressiveness of the protocol specifications, the protocol abstractions, and the preserved properties. We extend the state of the art in several ways:

1. We clarify and formally justify the application of abstractions to protocol specifications, which contain typed and untyped variables, not only ground terms as in [72] or only variables of strict (possibly structured) types as in [103]. This allows us to handle type-flaw attacks.
2. Unlike previous approaches which are limited to the free algebra over a fixed signature, we establish our soundness results for a useful subclass of *shallow* subterm-convergent rewrite theories modulo a set of axioms. This allows us to precisely model security protocols that employ algebraic properties of cryptographic primitives, e.g., specifying Diffie-Hellman style protocols as well as cryptographic protocols with XOR.
3. We extend [72, 103] with additional protocol abstractions (e.g., splitting encryptions or hashes) including many of those in [31, 49, 48]. We also introduce untyped protocol abstractions that can be combined with typed abstractions in various ways.
4. We provide syntactic criteria for our soundness conditions and a heuristic-based algorithm for automatically simplifying security protocols.
5. We have implemented this algorithm for the Scyther tool and validated its usefulness on a wide range of real-world security protocols mostly from the IKE and ISO/IEC 9798 families. Our abstractions result in very substantial performance gains. We have also obtained positive results for several other state-of-the-art verifiers (ProVerif, CL-Atse, OFMC, and SATMC) with manually produced abstractions.

1.4. Outline

The content of the thesis is organized as follows. In Chapter 2, we provide background on mathematical notation and basic definitions for term rewriting and equational theories.

In Part I, we introduce our security protocol abstractions and establish their soundness results. This part consists of two chapters. In Chapter 3, we present our security protocol model. First, we define a generic term algebra to model cryptographic messages. We introduce a type system that gives types to variables and terms, and a subtyping relation that defines a partial order over types. We define a subclass of subterm-convergent rewrite theories that is used to model the algebraic properties of cryptographic primitives. Then, we present a role-based protocol model to specify security protocols. Next, we define our adversary model and explain how security protocols are executed in the presence of an active intruder. Finally, we introduce a predicate-based property language that we use to specify security properties. In Chapter 4, we introduce three main types of protocol abstractions, namely *typed abstractions*, *untyped abstractions*, and *splitting abstractions*. A typed abstraction transforms a term's structure, e.g., splitting or removing cryptographic operations; an untyped abstraction removes redundant or unprotected terms; and a splitting abstraction transforms a protocol event into a sequence of events. By an example at the end of this chapter, we show how to combine different protocol abstractions to simplify a variant of the Kerberos protocol.

In Part II, we discuss an implementation of our abstraction mechanism and present the experimental results obtained using our protocol abstractions. In Chapter 5, we first give a description of our heuristic for automatically abstracting security protocols. Then, we explain how we have implemented our abstraction module for the Scyther tool. In Chapter 6, we present our experimental results using the extended Scyther with automatically generated abstractions and for other security protocol verification tools with manually produced abstractions on a wide range of case studies.

In Part III, we further discuss related work and conclude the thesis. In Chapter 7, we provide a comparison of our work with existing approaches on a technical level. In Chapter 8, we give a summary of the thesis and envision several directions for future work.

2. Background

In this chapter, we present some mathematical notation and the standard terminology for term rewriting and equational theories from [51, 10, 59].

2.1. Mathematical Notation

Sets We denote by $\mathbb{N}$ the set of natural numbers. For $n \in \mathbb{N}$, we write $\widetilde{n}$ to denote the set $\{1, \ldots, n\}$.

Ordered pairs We denote by (a, b) the ordered pairs of a and b, where a is the first entry and b is the second component. We also define the projections $\pi_i((a_1, a_2)) = a_i$ for $i \in \widetilde{2}$.

Sequences For a set S, we denote by S^* the set of finite sequences of elements in S. For two sequences s and s', we denote by $s \cdot s'$ the concatenation of s and s'. For a sequence s, we use $|s|$ to denote the length of s. We use ϵ to denote the empty sequence and $[s_1, \ldots, s_k]$ to denote the sequence of elements $s_1, \ldots, s_k$.

Relations For a set S, a relation $\rightarrow$ over S is a subset of the Cartesian product $S \times S$. We denote by $\rightarrow^+$ and $\rightarrow^*$ its transitive closure and reflexive transitive closure, respectively.

Functions We write $f \colon A \rightarrow B$ to denote a total function f that maps each element of A to an element of B. We write $f : X \rightharpoonup Y$ to denote a partial function f that maps some elements of X to elements of Y. In this case, the domain of f is the set of elements of X on which f is defined. We denote by $dom(f)$ and $ran(f)$ the domain and range of f, respectively. For a function f, we use $f[a \mapsto b]$ to denote the function that maps a to b and any other $c \neq a$ to $f(c)$.

2.2. Term Rewriting

2.2.1. Basic notions

Term algebra A *signature* Σ is a set of function symbols. Each function symbol $f \in \Sigma$ is associated with a number $n \geq 0$, called the *arity* of f. We denote by Σ^n the set of n-arity function symbols from Σ. We call the elements of Σ^0 *atoms* and write $\Sigma^{\geq 1}$ for the set of proper function symbols. Given a signature Σ and a set of *variables* V such that $\Sigma \cap V = \emptyset$, the set of terms $\mathcal{T}_\Sigma(V)$ over Σ and V is the smallest set such that $V \subseteq \mathcal{T}_\Sigma(V)$

and for all $n \geq 0$, all $f \in \Sigma^n$, and all $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(V)$, we have $f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma(V)$. For a term $t \in \mathcal{T}_\Sigma(V)$, we use $vars(t)$ to denote the set of variables occurring in t. A term $t \in \mathcal{T}_\Sigma(V)$ is *ground* if $vars(t) = \emptyset$ and *open* otherwise. We denote the top-level symbol of a (non-variable) term t by $top(t)$ and the set of its symbols in $\Sigma^{\geq 1}$ by $ct(t)$.

Positions, subterms, sizes, and replacements A position is a sequence of natural numbers representing the path from the top-level symbol to the function symbol at that position. For a term t, the subterm of t at position p, written $t|_p$, is defined inductively as

$$t|_p = \begin{cases} t & \text{if } p = \epsilon \\ t_i|_{p'} & \text{if } p = [i] \cdot p',\ t = f(t_1, \ldots, t_n), \text{ and } 1 \leq i \leq n \\ \text{undefined} & \text{otherwise} \end{cases}$$

For a term t, we denote the set of positions in t by $\mathcal{P}os(t)$. We write $subs(t)$ for the set of subterms of t and define the size of t by $|t| = |subs(t)|$. A subterm t' of t is *proper* if $t' \neq t$. For positions p and q, we write $p \sqsubseteq q$ if p is a prefix of q. We also write $p \sqsubset q$ if p is a strict prefix of q. We say that p and q are comparable if $p \sqsubseteq q$ or $q \sqsubseteq p$. The term obtained from t by replacing $t|_p$ with u, written $t[u]_p$, is defined as

$$t[u]_p = \begin{cases} u & \text{if } p = \epsilon \\ f(t_1, \ldots, t_i[u]_{p'}, \ldots, s_n) & \text{if } p = [i] \cdot p',\ t = f(t_1, \ldots, t_n), \text{ and } 1 \leq i \leq n \\ \text{undefined} & \text{otherwise} \end{cases}$$

A *replacement* ρ is a function from a finite set of terms to $\mathcal{T}_\Sigma(V)$ such that $dom(\rho) \cap subs(ran(\rho)) = \emptyset$. We denote by $t\rho$ the term obtained from t by replacing all occurrences of terms $t|_p \in dom(\rho)$ in t with $\rho(t|_p)$ such that no position p' above p in t with $t|_{p'} \in dom(\rho)$ and $t|_{p'}$ is replaced by $\rho(t|_{p'})$. We use $[t/u]$ to denote the single replacement of u by t.

Substitutions, composition, unification, and matching A *substitution* over $\mathcal{T}_\Sigma(V)$ is a function $\sigma\colon V \to \mathcal{T}_\Sigma(V)$ such that σ is the identity function except for a finite subset of V, which we denote by $dom(\sigma)$. We also write $ran(\sigma)$ to denote the image of $dom(\sigma)$ under σ. Substitutions are homomorphically extended to all terms in $\mathcal{T}_\Sigma(V)$. For a term t and a substitution σ, we denote by $t\sigma$ the application of σ to t. We use $\{t_1/x_1, \ldots, t_n/x_n\}$ to denote the substitution σ with $dom(\sigma) = \{x_1, \ldots, x_n\}$ and $x_i\sigma = t_i$ for all i, $1 \leq i \leq n$. We write $\sigma|_V$ for $V \subseteq dom(\sigma)$ to denote the substitution obtained from σ by restricting its domain to V. For substitutions σ, θ, the composition of σ and θ, written $\sigma\theta$, is defined as $x(\sigma\theta) = (x\theta)\sigma$ for all $x \in V$.

For terms t and u, we say that t and u are *unifiable* if there is a substitution σ such that $t\sigma = u\sigma$. We also say that t *matches* u (or u is an *instance of* t) if there is a substitution σ such that $t\sigma = u$.

Equations and equational theories An *equation* over a signature Σ is an unordered pair $\{s, t\}$, written $s \simeq t$, where $s, t \in \mathcal{T}_\Sigma(\mathcal{V})$. An *equational presentation* $\mathcal{E} = (\Sigma, E)$

consists of a signature Σ and a set E of equations over Σ. The *equational theory* induced by $\mathcal{E}$ is the smallest Σ-congruence, written $=_E$, containing all instances of equations in E. We often identify $\mathcal{E}$ with the induced equational theory.

Equational unification For a set of equations E and two terms t, u, we say that t and u are *E-unifiable* if and only if there is a substitution σ such that $t\sigma =_E u\sigma$. We also call σ an *E-unifier* of t and u.

Rewrite Systems A *rewrite rule* over a signature Σ is an oriented pair $l \to r$, where $l, r \in \mathcal{T}_\Sigma(V)$ and $vars(r) \subseteq vars(l)$. A *rewrite system* R is a set of rewrite rules. For a rewrite system R, the rewriting relation $\to_R$ on $\mathcal{T}_\Sigma(V)$ is defined by $t \to_R t'$, if and only if there exists a non-variable position $p \in \mathcal{P}os(t)$, a rule $l \to r \in R$, and a substitution σ such that $t|_p = l\sigma$ and $t' = t[r\sigma]_p$. The term $t|_p$ is called a *redex*. We say that R is

- *terminating* if and only if there is no infinite chain $t_1 \to_R t_2 \to_R \cdots$,
- *confluent* if and only if for all terms t, t_1, and t_2 such that $t \to_R^* t_1$ and $t \to_R^* t_2$, there is a term t' such that $t_1 \to_R^* t'$ and $t_2 \to_R^* t'$, and
- *convergent* if and only if it is terminating and confluent. In this case, for each term t, there is a unique term t' such that $t \to_R^* t'$ and there is no term t'' such that $t' \to_R t''$. We call t' the *R-normal form* of t and denote it by $t{\downarrow_R}$. We also say that t' is in R-normal form (or R-normal).

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

2.2.2. Rewriting Modulo Equational Theories

It is well-known that many cryptographic operators, such as those that are commutative, cannot be oriented to obtain a terminating rewrite system.

Example 1. Consider $\Sigma = \{\oplus\}$, $E = \{X \oplus Y \simeq Y \oplus X\}$, and a term $t = a \oplus b$. Suppose that the equation $X \oplus Y \simeq Y \oplus X$ can be oriented from left to right to obtain the rewrite system $R = \{X \oplus Y \to Y \oplus X\}$. But R is not terminating since we obtain an infinite rewrite sequence $t \to_R b \oplus a \to_R a \oplus b \to_R b \oplus a \to_R \cdots$. ♠

In order to cope with this issue, rewrite systems are often considered modulo equational theories. This leads us to the notion of *rewrite theories*. Formally, a *rewrite theory* is a triple $\mathcal{R} = (\Sigma, Ax, R)$ where Σ is a signature, Ax a set of Σ-equations, and R a set of rewrite rules. We say that a rewrite theory $\mathcal{R} = (\Sigma, Ax, R)$ is subterm-convergent whenever R is.

The rewrite relation $\to_{R/Ax}$ is defined (on equivalence classes of terms) such that for all terms s and t, we have $s \to_{R/Ax} t$ if and only if there are terms s' and t' such that $s =_{Ax} s'$, $t =_{Ax} t'$, and $s' \to_R t'$. We say that $\to_{R/Ax}$ is *confluent* if and only if for all terms t, t_1, and t_2 such that $t \to_{R/Ax}^* t_1$ and $t \to_{R/Ax}^* t_2$, there is a term t' such that $t_1 \to_{R/Ax}^* t'$ and $t_2 \to_{R/Ax}^* t'$.

Decomposition of equational theories A prominent application of rewrite theories is to decide equality modulo an equational theory $\mathcal{E} = (\Sigma, E)$. Suppose that some equations in E can be oriented to obtain a set of rewrite rules R and the remaining equations in E constitute a set of axioms Ax. If $\rightarrow_{R/Ax}$ is convergent then we can use $\rightarrow_{R/Ax}$ to decide equality modulo $\mathcal{E}$, since for all terms s and t, it holds that $s =_E t$ if and only if there are terms s' and t' such that $s =_{Ax} s'$, $t =_{Ax} t'$, and $s'\downarrow_{R/Ax} = t'\downarrow_{R/Ax}$. However, finding reducible terms in Ax-equivalence classes may be undecidable or inefficient as these classes may be infinite. A weaker relation $\rightarrow_{R,Ax}$ has therefore been proposed in [109, 77, 78]. The rewriting relation $\rightarrow_{R,Ax}$ on $\mathcal{T}_\Sigma(V)$ is defined by $t \rightarrow_{R,Ax} t'$ if and only if there exists a non-variable position p in t, a rule $l \rightarrow r \in R$, and a substitution σ such that $t|_p =_{Ax} l\sigma$ and $t' = t[r\sigma]_p$. We say that $\rightarrow_{R,Ax}$ is *confluent* if and only if for all terms t, t_1, and t_2 such that $t \rightarrow^*_{R,Ax} t_1$ and $t \rightarrow^*_{R,Ax} t_2$, there are terms t'_1 and t'_2 such that $t_1 \rightarrow^*_{R,Ax} t'$ and $t_2 \rightarrow^*_{R,Ax} t'$, and $t'_1 =_{Ax} t'_2$. The following example shows that $\rightarrow_{R,Ax}$ is strictly weaker than $\rightarrow_{R/Ax}$.

Example 2. Consider $Ax_+ = \{(X{+}Y){+}Z \simeq X{+}(Y{+}Z)\}$ and $R_+ = \{0{+}X \rightarrow X\}$. Then we have $(a{+}0){+}b \rightarrow_{R_+/Ax_+} a{+}b$, because $(a{+}0){+}b =_{Ax_+} a{+}(0{+}b) \rightarrow_{R_+} a{+}b$. However, the term $(a{+}0){+}b$ is already in R_+, Ax_+-normal form. ♠

In order to bridge the gap between the relations $\rightarrow_{R,Ax}$ and $\rightarrow_{R/Ax}$, it is sufficient to require that $\rightarrow_{R/Ax}$ is convergent and *coherent*. The latter condition says that for all terms t_1, t_2, and t_3 such that $t_1 \rightarrow_{R,Ax} t_2$ and $t_1 =_{Ax} t_3$, there are t_4 and t_5 such that $t_2 \rightarrow^*_{R,Ax} t_4$, $t_3 \rightarrow^+_{R,Ax} t_5$, and $t_4 =_{Ax} t_5$. Under these conditions, it holds that for all terms t and u, $t =_E u$ if and only if $t\downarrow_{R,Ax} =_{Ax} u\downarrow_{R,Ax}$. The proof can be found in [78].

An equational presentation $\mathcal{E} = (\Sigma, E)$ is *decomposable* if E can be partitioned into a set of axioms Ax and a set of rewrite rules R such that $\rightarrow_{R/Ax}$ is convergent and coherent. Note that there are equational presentations that are not decomposable. Let us consider a model of the XOR operator in the following example.

Example 3. Let $\mathcal{E}_\oplus = (\Sigma_\oplus, E_\oplus)$ be an equational presentation, where $\Sigma_\oplus = \{\oplus\}$ and $E_\oplus = \{X \oplus 0 \simeq X,\ X \oplus X \simeq 0,\ (X \oplus Y) \oplus Z \simeq X \oplus (Y \oplus Z),\ X \oplus Y \simeq Y \oplus X\}$. We orient the first two equations from left to right to obtain the set of rewrite rules $R_\oplus$ and the set of axioms $Ax_\oplus$, where

$$\begin{aligned} R_\oplus &= \{X \oplus 0 \rightarrow X,\ X \oplus X \rightarrow 0\} \\ Ax_\oplus &= \{(X \oplus Y) \oplus Z \simeq X \oplus (Y \oplus Z),\ X \oplus Y \simeq Y \oplus X\} \end{aligned}$$

We consider $t_1 = (x \oplus x) \oplus y$, $t_2 = 0 \oplus y$, and $t_3 = x \oplus (x \oplus y)$. Then we have $t_1 \rightarrow_{R_\oplus,Ax_\oplus} t_2$ and $t_1 =_{Ax_\oplus} t_3$. But there is no term t_5 such that $t_3 \rightarrow^+_{R_\oplus,Ax_\oplus} t_5$ since t_3 is already $R_\oplus, Ax_\oplus$-normal form. Therefore, $R_\oplus/Ax_\oplus$ is not coherent. By adding the missing rewrite rule $X \oplus (X \oplus Y) \rightarrow Y$ to $R_\oplus$, we obtain $R'_\oplus$ and $(\Sigma_\oplus, R'_\oplus, Ax_\oplus)$ is a decomposition of $\mathcal{E}_\oplus$. ♠

Part I.

Security Protocol Abstractions

3. Security Protocol Model

In this chapter, we define our modeling of security protocols. We organise the presentation as follows. First, we present our message theory. Then, we describe how security protocols are formulated in our framework. Next, we present our protocol specification and provide an operational semantics for protocol executions based on the book [40]. Finally, we introduce a property language for specifying security goals.

3.1. Cryptographic Message Model

We model cryptographic messages as terms from a term algebra $\mathcal{T}_\Sigma(V)$ over a signature Σ and a set of variables V. For a fixed $\Sigma^{\geq 1}$, we will vary Σ^0 to generate different sets of terms, denoted by $\mathcal{T}(V, \Sigma^0)$, including terms in protocol roles, network messages, and types. We also partition Σ into sets of public and private symbols, denoted by Σ_{pub} and Σ_{pri}. We assume Σ_{pub} includes (ordered) pairing $\langle \cdot, \cdot \rangle$ which associates to the right, e.g., $\langle t, u, v \rangle = \langle t, \langle u, v \rangle \rangle$. We usually omit the tuple brackets under other symbols, e.g., we write $\{\!|t, u, v|\!\}_k$ rather than $\{\!|\langle t, u, v \rangle|\!\}_k$. We define the *splitting* function by $\mathit{split}(\langle t, u \rangle) = \mathit{split}(t) \cup \mathit{split}(u)$ on pairs and $\mathit{split}(t) = \{t\}$ on other terms t. We call the elements of $\mathit{split}(t)$ the *fields* of t.

We denote by $\mathcal{V}$, $\mathcal{A}$, $\mathcal{F}$, and $\mathcal{C}$ infinite and pairwise disjoint sets of variables, agents, fresh values, and constants, respectively. We use $\mathcal{A}$ to model agent identities, $\mathcal{F}$ to model freshly generated random numbers such as nonces and timestamps, and $\mathcal{C}$ to model non-varying values such as the identity element 0 of the XOR operator. We define the set of *message terms* as $\mathcal{M} = \mathcal{T}(\mathcal{V}, \mathcal{A} \cup \mathcal{F} \cup \mathcal{C})$. We partition $\mathcal{A}$ into sets of honest and compromised agents: $\mathcal{A} = \mathcal{A}_H \cup \mathcal{A}_C$.

3.1.1. Type System

It is well-known that the security of cryptographic protocols is undecidable [56, 121]. One of the sources of undecidability is the unboundedness of the message length. To give a bound on the message length and hence obtain decidability results for relevant subclasses of protocols, *type systems* have been used [88, 111, 6, 98]. This approach relies on the assumption that all agents can distinguish different message types, which is often achieved by proper formatting or tagging mechanisms [69, 85]. Therefore, it is often sufficient to restrict attacks to *well-typed* ones. Well-typedness allows us to bound the length of messages that we need to consider. Many leading security protocol verifiers, e.g., Scyther [41], ProVerif [19], and Tamarin [94] employ type systems among other optimization strategies to reduce the search space in the verification of cryptographic protocols. These tools are guaranteed to terminate on a vast number of case studies.

In the context of protocol abstraction, type systems are also used to enable fine-grained control over abstractions, i.e., simplifying different terms of the same shape in different ways. This allows us to define pattern-based abstractions which are not possible without types. We will later discuss the usefulness of types in Example 10.

We introduce a type system akin to [6, 98] and extend it with subtyping. We define the set of atomic types by $\mathcal{Y}_{at} = \mathcal{Y}_0 \cup \{\alpha, \mathsf{msg}\} \cup \{\beta_n \mid n \in \mathcal{F}\} \cup \{\gamma_c \mid c \in \mathcal{C}\}$, where α, β_n, and γ_c are the types of agents, the fresh value n, and the constant c, respectively. In this system, all agent names have the same type α. Each fresh value $n \in \mathcal{F}$ and constant $c \in \mathcal{C}$ has its own type: β_n and γ_c. This will enable a fine-grained control in our typed abstractions. Moreover, msg is the type of all messages and $\mathcal{Y}_0$ is a disjoint set of user-defined types. The set of all types is then defined by $\mathcal{Y} = \mathcal{T}(\emptyset, \mathcal{Y}_{at})$.

We assume that all variables have an atomic type, i.e., $\mathcal{V} = \{\mathcal{V}_\tau\}_{\tau \in \mathcal{Y}_{at}}$ is a family of disjoint infinite sets of variables. Let $\Gamma\colon \mathcal{V} \to \mathcal{Y}_{at}$ be such that $\Gamma(X) = \tau$ if and only if $X \in \mathcal{V}_\tau$. We extend Γ to atoms by defining $\Gamma(a) = \alpha$, $\Gamma(n) = \beta_n$, and $\Gamma(c) = \gamma_c$ for $a \in \mathcal{A}$, $n \in \mathcal{F}$, and $c \in \mathcal{C}$, and then homomorphically to all terms $t \in \mathcal{M}$. We call $\tau = \Gamma(t)$ the *type of* t and sometimes also write $t : \tau$. We also call a variable X with $X : \mathsf{msg}$ a *message variable*.

The subtyping relation $\preccurlyeq$ on types is defined by the following inference rules and by two additional rules (not shown) defining its reflexivity and transitivity.

$$\frac{\tau \in \mathcal{Y}}{\tau \preccurlyeq \mathsf{msg}}\ \mathrm{S}(\mathsf{msg}) \qquad \frac{\tau_1 \preccurlyeq_0 \tau_2}{\tau_1 \preccurlyeq \tau_2}\ \mathrm{S}(\preccurlyeq_0) \qquad \frac{\tau_1 \preccurlyeq \tau_1' \quad \cdots \quad \tau_n \preccurlyeq \tau_n'}{c(\tau_1, \ldots, \tau_n) \preccurlyeq c(\tau_1', \ldots, \tau_n')}\ \mathrm{S}(c \in \Sigma^n)$$

Every type is a subtype of msg by the first rule. The second rule embeds a user-defined *atomic subtyping relation* $\preccurlyeq_0 \subseteq (\mathcal{Y}_{at} \setminus \{\mathsf{msg}\}) \times \mathcal{Y}_0$, which relates atomic types (except msg) to user-defined atomic types in $\mathcal{Y}_0$. For simplicity, we require that $\preccurlyeq_0$ is a partial function. The third rule ensures that subtyping is preserved by all symbols. The set of subtypes of τ is $\tau{\downarrow} = \{\tau' \in \mathcal{Y} \mid \tau' \preccurlyeq \tau\}$.

3.1.2. Shallow Equational Theories

In this subsection, we characterize a class of equational theories that we use to model cryptographic operations. We first provide the formal definition of this class. Then, we illustrate its expressiveness on several examples.

Let $\mathcal{E} = (\Sigma, E)$ be an equational theory over Σ such that for all pair $\{s, t\} \in E$, we have $s, t \in \mathcal{T}_\Sigma(\mathcal{V}_{\mathsf{msg}})$. We assume that $\mathcal{E}$ can be decomposed into a rewrite theory (Σ, Ax, R) where $Ax \subseteq E$ and, for all terms $t, u \in \mathcal{T}_\Sigma(V)$, we have $t =_E u$ if and only if $t{\downarrow_{R,Ax}} =_{Ax} u{\downarrow_{R,Ax}}$. As mentioned in Section 2.2, decomposability can be achieved under suitable termination, confluence, and coherence conditions. In this thesis, we work with decomposable equational theories. We have used the AProVE termination tool [61] and Maude Church-Rosser and Coherence Checker [55] to verify the termination, confluence, and coherence properties for our case studies. To establish our soundness result, we consider the subclass $\mathcal{S}$ of subterm-convergent rewrite theories where each rule in R has one of the following forms.

- (R1): $d(c(x_1, \ldots, x_n, t), u) \to x_j$, where $c, d \in \Sigma_{\mathsf{pub}}$, t, u are terms, $j \in \widetilde{n}$, and $x_1, \ldots, x_n$ are pairwise distinct variables with $x_i \notin vars(t, u)$ for all $i \in \widetilde{n}$.
- (R2): $d(c(x_1, \ldots, x_n)) \to x_j$, where $c, d \in \Sigma_{\mathsf{pub}}$, $j \in \widetilde{n}$, and $x_1, \ldots, x_n$ are pairwise distinct variables.
- (R3): $c(x_1, \ldots, x_n) \to x_j$ where $c \in \Sigma_{\mathsf{pub}}$, x_j for some $j \in \widetilde{n}$ is a variable, and x_i is a variable or an atom for all $i \in \widetilde{n}$.
- (R4): $l \to a$ for a composed term l and a constant a.

Intuitively, the first three forms enable different types of projection of a term's arguments. Rules R1 and R2 apply a function symbol d to extract one of c's arguments. Rule R3 uses no destructor. Finally, R4 models rewriting a term to a constant. Since the rules (R1-R3) have limited depth, we call the class $\mathcal{S}$ of rewrite theories *shallow subterm-convergent.*

We also introduce a condition on the equations Ax of the rewrite theory.

Definition 1. A rewrite theory (Σ, Ax, R) is *well-formed* if for all $\{s, t\} \in Ax$ and all rewrite rules $l \to r \in R$, we have (i) s, t, and l are composed, (ii) neither s, t, nor l is a pair, (iii) $top(s) = top(t)$, and (iv) $vars(s) = vars(t)$.

We only consider equational theories that can be decomposed into a shallow subterm-convergent, well-formed rewrite theory. These are adequate to model many well-known cryptographic primitives as illustrated by the examples below.

Example 4. We model the protocols of our case studies (see Section 4.5 and Chapter 6) in the rewrite theory $\mathcal{R}_{cs} = (\Sigma_{cs}, Ax_{cs}, R_{cs})$ where $\Sigma_{cs} = \Sigma^0_{cs} \cup \Sigma^1_{cs} \cup \Sigma^2_{cs}$ and

$$\begin{aligned}
\Sigma^1_{cs} &= \{\mathsf{pk}, \mathsf{pri}, \mathsf{prf}, \mathsf{kdf}, \mathsf{mac}, \mathsf{fst}, \mathsf{snd}\} \\
\Sigma^2_{cs} &= \{\mathsf{sh}, \langle \cdot, \cdot \rangle, \{\!|\cdot|\!\}_\cdot, \{\!|\cdot|\!\}^{-1}_\cdot, \{\cdot\}_\cdot, \{\cdot\}^{-1}_\cdot, [\cdot]_\cdot, \mathsf{ver}\} \\
\Sigma_{\mathsf{pri}} &= \{\mathsf{sh}, \mathsf{pri}\}
\end{aligned}$$

In the first line, the function symbols pk and pri model public keys and private keys, respectively. The symbols prf, kdf, and mac are hash functions. Note that mac models a *message authentication code* which involves a cryptographic hash function in combination with a symmetric key. At the end of this line, fst and snd denote projections on pairs. In the second line, sh models pre-shared symmetric keys and $\langle \cdot, \cdot \rangle$ models pairs of terms. We use $\{\!|\cdot|\!\}_\cdot$ and $\{\!|\cdot|\!\}^{-1}_\cdot$ to model symmetric encryption and decryption. Likewise, the symbols $\{\cdot\}_\cdot$ and $\{\cdot\}^{-1}_\cdot$ denote asymmetric encryption and decryption. The symbols $[\cdot]_\cdot$ and ver are used to model signing and verification, respectively. In the last line, we require that sh and pri are private function symbols.

The set of atoms Σ^0_{cs} is specified later in Example 7. The set R_{cs} consists of rewrite rules for projections (type R2) and for decryption and signature verification (type R1):

$$\begin{array}{lll}
\mathsf{fst}(\langle X, Y \rangle) \to X & \{\!|\{\!|X|\!\}_Y|\!\}^{-1}_Y \to X & \mathsf{ver}([X]_{\mathsf{pri}(Y)}, \mathsf{pk}(Y)) \to X \\
\mathsf{snd}(\langle X, Y \rangle) \to Y & \{\{X\}_{\mathsf{pk}(Y)}\}^{-1}_{\mathsf{pri}(Y)} \to X &
\end{array}$$

where X and Y are variables. We have two equations in Ax_{cs}, namely $\mathsf{exp}(\mathsf{exp}(g, X), Y) \simeq \mathsf{exp}(\mathsf{exp}(g, Y), X)$ and $\mathsf{sh}(X, Y) \simeq \mathsf{sh}(Y, X)$. The first equation models commutativity of Diffie-Hellman exponentiation. Pre-shared symmetric keys can be used in two different ways. If two parties use the same long-term symmetric key for their communication, then such keys are called *bidirectional*. Otherwise, they share two different keys, one for each direction. In this case, these keys are called *unidirectional*. The latter models bidirectional pre-shared symmetric keys. ♠

Example 5. The theory of XOR is given by the following rewrite system where the rewrite rules are of types R1, R3, and R4. The rightmost rule ensures coherence [78].

$$\begin{array}{rclll} X \oplus Y & \simeq & Y \oplus X & X \oplus 0 \to X & Y \oplus X \oplus X \to Y \\ (X \oplus Y) \oplus Z & \simeq & X \oplus (Y \oplus Z) & X \oplus X \to 0 & \end{array}$$

♠

For our theoretical development, we consider an arbitrary but fixed shallow subterm-convergent and well-formed rewrite theory (Σ, Ax, R) that includes the function symbols and rewrite rules for pairing and projections.

We now define well-typed substitutions.

Definition 2 (Well-typed substitutions)**.** A substitution θ is *well-typed* if it holds that $\Gamma((X\theta)\downarrow_{R,Ax}) \preccurlyeq \Gamma(X)$ for all $X \in dom(\theta)$.

3.2. Protocol Specification

We specify a security protocol as a partial function from agent variables to *roles*. A role is a sequence of events. We distinguish three types of events: *send events*, *receive events*, and *signal events*. For a term t, a send event $\mathsf{send}(t)$ indicates the transmission of a message that is an instance of t. Likewise, a receive event $\mathsf{rcv}(t)$ indicates the reception of a message that matches t. Finally, a signal event marks a progressive stage of an agent playing a role, i.e., it tells how far the agent has been executing. We use signal events to specify security properties. Past research has also employed signal events to express various authentication specifications [116, 117] for security protocols.

We assume a fixed set Sig of signal events that is disjoint from the normal protocol events, i.e., $Sig \cap \{\mathsf{send}, \mathsf{rcv}\} = \emptyset$. For a set of terms T, we define the set of events $Evt(T) = \{\mathsf{send}(t), \mathsf{rcv}(t) \mid t \in T\} \cup Sig$ and $term(ev(t)) = t$ for event $ev(t)$. A *role* is a sequence of events from $Evt(\mathcal{M})$.

Definition 3 (Protocol)**.** A *protocol* is a function $P : \mathcal{V}_\alpha \rightharpoonup Evt(\mathcal{M})^+$ mapping agent variables to roles. Let $\mathcal{M}_P = term(ran(P))$ be the set of *protocol terms* appearing in the roles of P, and let $\mathcal{V}_P$, $\mathcal{A}_P$, $\mathcal{F}_P$, and $\mathcal{C}_P$ denote the sets of variables, agents, fresh values, and constants in $\mathcal{M}_P$.

Let us consider the Needham-Schroeder public-key protocol [101] whose message sequence chart (MSC) is depicted in Figure 3.2. The protocol works as follows. First, the

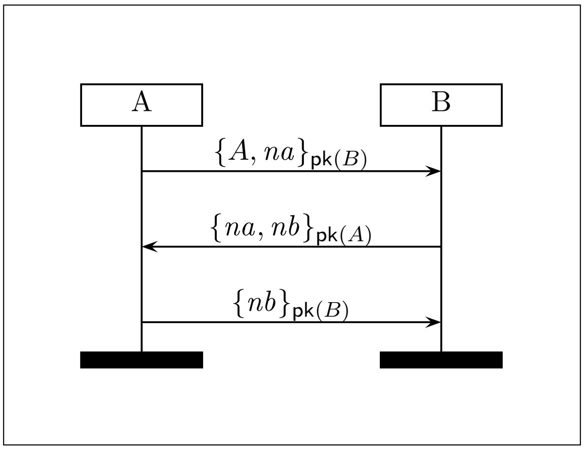

Figure 3.1.: The Needham-Schroeder public-key protocol.

initiator in role A generates a nonce na, encrypts this nonce together with his identity using the public key of the responder in role B, and sends the resulting message to the responder. Then, the responder confirms his reception of the initiator's nonce by sending the encryption of na and his own generated nonce nb with the initiator's public key. Finally, the initiator returns to the responder nonce nb encrypted with the responder's public key.

In the following example, we explain how to formalize this protocol in our specification language. For the formal specifications of security protocols, we introduce some conventions. All identifiers for variables (atoms) start with an upper-case (a lower-case) letter. We also assume a user-defined type Nonce to model fresh values generated by protocol participants, agent variables $I, R \in \mathcal{V}_\alpha$, and a set $F \subseteq \mathcal{F}$. We require that $\beta_n \preccurlyeq_0 \mathsf{Nonce}$ for all $n \in F$. We will instantiate F in different ways for our examples.

Example 6. To model the protocol depicted in Figure 3.2, we assume a set of fresh values $F = \{na, nb\}$, and a signature $\Sigma = \{\mathsf{pk}, \mathsf{pri}\}$. Let $Na, Nb \in \mathcal{V}_{\mathsf{Nonce}}$. We specify this protocol by the function $NS : \mathcal{V}_\alpha \rightharpoonup Evt(\mathcal{M})^+$ where

$$\begin{array}{rcl} NS(A) & = & \mathsf{send}(\{A, na\}_{\mathsf{pk}(B)}) \cdot \mathsf{rcv}(\{na, Nb\}_{\mathsf{pk}(A)}) \cdot \mathsf{send}(\{Nb\}_{\mathsf{pk}(B)}) \\ NS(B) & = & \mathsf{rcv}(\{A, Na\}_{\mathsf{pk}(B)}) \cdot \mathsf{send}(\{Na, nb\}_{\mathsf{pk}(A)}) \cdot \mathsf{rcv}(\{nb\}_{\mathsf{pk}(B)}) \end{array}$$

and NS is undefined otherwise. ♠

We now consider an Internet Key Exchange (IKE) protocol which also serves as our running case study. The IKE family of protocols are part of the IPsec [81] protocol suite for securing Internet Protocol (IP) communication. It provides confidentiality, integrity, and data source authentication for IP packets. IKE establishes a shared key, which is later used for securing IP packets, realizes mutual authentication, and offers identity protection as an option. Its first version (IKEv1) dates back to 1998 [68]. The second version (IKEv2) [80] significantly simplifies the first one. However, the protocols in this family are still complex and contain a large number of fields. As our

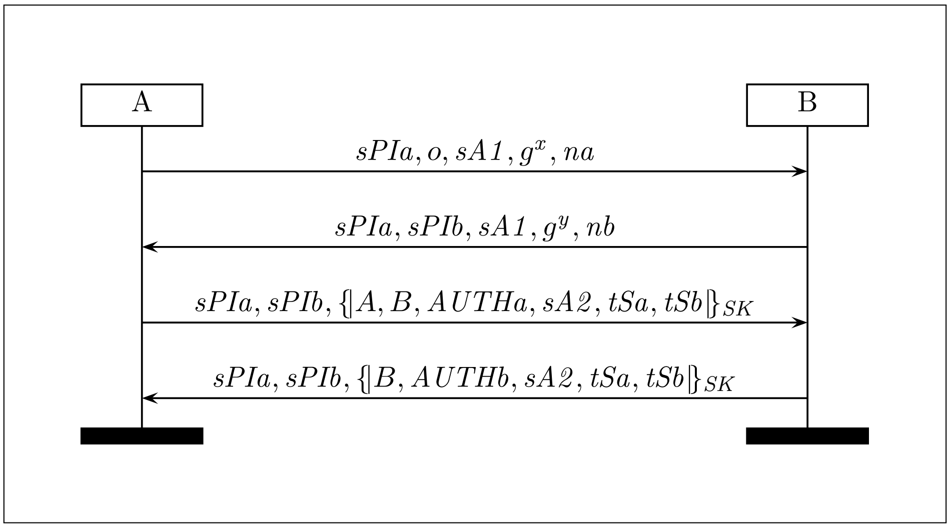

Figure 3.2.: The $\mathsf{IKE_m}$ protocol.

running example, we present a member of the IKEv2 family, called IKEv2-mac (or $\mathsf{IKE_m}$ for short), which sets up a session key using a Diffie-Hellman (DH) key exchange, provides mutual authentication based on MACs, and also offers identity protection. We use Cremers' models of IKE [42] as a basis for our presentation and experiments (see Chapter 6). We depict the $\mathsf{IKE_m}$ protocol's MSC in Figure 3.2.

Here, *SPIa* and *SPIb* denote the *Security Parameter Indices* that determine cryptographic algorithms, o is a constant number, *sA1* and *sA2* are *Security Associations*, g is the DH group generator, x and y are secret DH exponents, *Na* and *Nb* are nonces, and *tSa* and *tSb* denote *Traffic Selectors* specifying certain IP parameters. $AUTHa$ and $AUTHb$ denote the authenticators of A and B, and *SK* denotes the session key derived from the DH key g^{xy}. These are defined as follows.

$$
\begin{aligned}
SK &= \mathsf{kdf}(na, nb, g^{xy}, sPIa, sPIb),\\
AUTHa &= \mathsf{mac}(\mathsf{sh}(A,B), sPIa, o, sA1, g^x, na, nb, \mathsf{prf}(SKa, A))\\
AUTHb &= \mathsf{mac}(\mathsf{sh}(B,A), sPIa, sPIb, sA1, g^y, nb, na, \mathsf{prf}(SKb, B))
\end{aligned}
$$

The symbols mac, kdf, and prf denote a message authentication code, a key derivation function, and a pseudorandom function, respectively.

Example 7. We formalize the $\mathsf{IKE_m}$ protocol in the rewrite theory of Example 4 as follows. The atoms Σ^0_{cs} are composed of constants $C = \{g, o, sA1, sA2, tSa, tSb\}$ and fresh values $F = \{na, nb, x, y, sPIa, sPIb\}$. The variables and their types are

$$A, B\colon \alpha, \quad Ga, Gb\colon \mathsf{msg}, \quad SPIa, SPIb, Na, Nb\colon \mathsf{Nonce}.$$

We model mac, kdf, and prf as hash functions. We also assume a set of signal events

$$Sig = \{\mathsf{Running}, \mathsf{Commit}, \mathsf{Secret}\}.$$

We later use Running and Commit to specify authentication properties and Secret to specify secrecy properties (see Example 8). We formalize the roles of the initiator A and the responder B as follows.

$$\begin{array}{rcl}\mathsf{IKE_m}(A) & = & \mathsf{send}(sPIa, o, sA1, \mathsf{exp}(g,x), na)\cdot\mathsf{rcv}(sPIa, SPIb, sA1, Gb, Nb)\cdot \\ & & \mathsf{Running}\cdot\mathsf{send}(sPIa, SPIb, \{\!|A, B, AUTHaa, sA2, tSa, tSb|\!\}_{SKa})\cdot \\ & & \mathsf{rcv}(sPIa, SPIb, \{\!|B, AUTHba, sA2, tSa, tSb|\!\}_{SKa})\cdot\mathsf{Secret}\cdot\mathsf{Commit}\end{array}$$

$$\begin{array}{rcl}\mathsf{IKE_m}(B) & = & \mathsf{rcv}(SPIa, o, sA1, Ga, na)\cdot\mathsf{send}(SPIa, sPIb, sA1, \mathsf{exp}(g,y), nb)\cdot \\ & & \mathsf{rcv}(SPIa, sPIb, \{\!|A, B, AUTHab, sA2, tSa, tSb|\!\}_{SKb})\cdot\mathsf{Running}\cdot \\ & & \mathsf{send}(SPIa, sPIb, \{\!|B, AUTHbb, sA2, tSa, tSb|\!\}_{SKb})\cdot\mathsf{Secret}\cdot\mathsf{Commit}\end{array}$$

where the authenticators and session keys are defined as follows.

$$\begin{array}{rcl}SKa & = & \mathsf{kdf}(na, Nb, \mathsf{exp}(Gb, x), sPIa, SPIb), \\ SKb & = & \mathsf{kdf}(Na, nb, \mathsf{exp}(Ga, y), SPIa, sPIb), \\ AUTHaa & = & \mathsf{mac}(\mathsf{sh}(A,B), sPIa, o, sA1, \mathsf{exp}(g,x), na, Nb, \mathsf{prf}(SKa, A)), \\ AUTHab & = & \mathsf{mac}(\mathsf{sh}(B,A), SPIa, o, sA1, Ga, Na, nb, \mathsf{prf}(SKb, A)), \\ AUTHba & = & \mathsf{mac}(\mathsf{sh}(A,B), sPIa, SPIb, sA1, Gb, Nb, na, \mathsf{prf}(SKa, B)), \\ AUTHbb & = & \mathsf{mac}(\mathsf{sh}(B,A), SPIa, sPIb, sA1, \mathsf{exp}(g,y), nb, Na, \mathsf{prf}(SKb, B)).\end{array}$$

♠

3.3. Adversary Model and Operational Semantics

In this section, we introduce our adversary model and define an operational semantics for security protocol executions. Intuitively, protocol executions capture dynamic behaviours. When a protocol is executed, each of its roles can be executed an arbitrary number of times by possibly different agents in parallel. Such single execution of a role is called a *thread* or *run*. We distinguish between different threads by associating each thread with a unique *thread identifier*. When a role is instantiated into a thread for execution, we mark its variables and fresh values with the thread identifier i to distinguish them from those of other threads.

We assume a countably infinite set TID of thread identifiers. We define the instantiation $t^{\sharp i}$ of a term t for $tid \in TID$ as the homomorphic extension of the following definition to all messages:

$$\begin{array}{rcll}w^{\sharp tid} & = & w^{tid} & \text{if } w \in \mathcal{V}\cup\mathcal{F} \\ c^{\sharp tid} & = & c & \text{if } c \in \mathcal{A}\cup\mathcal{C}\end{array}$$

Intuitively, this function maps every variable or fresh value u to u^i, and keeps constants and agents unchanged. It transform terms occurring in role descriptions to terms occurring in threads. Furthermore, such transformations do not affect the type of a term, i.e., $\Gamma(t^{\sharp i}) = \Gamma(t)$ for all terms t.

We define by $T^\sharp = \{t^{\sharp i} \mid t \in T \wedge i \in TID\}$ the set of instantiations of terms in a set T and abbreviate $T^\flat = T \cup T^\sharp$. For example, $\mathcal{M}^\sharp$ is the set of instantiated message

$$\frac{u \in T}{T \vdash_E u}\ \mathsf{Ax} \qquad \frac{T \vdash_E t' \quad t' =_E t}{T \vdash_E t}\ \mathsf{Eq} \qquad \frac{T \vdash_E t_1 \quad \cdots \quad T \vdash_E t_n}{T \vdash_E f(t_1, \ldots, t_n)}\ \mathsf{Comp}\ (f \in \Sigma^{\geq 1}_{\mathsf{pub}})$$

Figure 3.3.: Intruder deduction rules (where $\Sigma^{\geq 1}_{\mathsf{pub}} = \Sigma^{\geq 1} \cap \Sigma_{\mathsf{pub}}$)

terms, which we will use to instantiate roles into threads. We define the set of *network messages* exchanged during protocol execution by $\mathcal{N} = \mathcal{T}(\mathcal{V}^\sharp, \mathcal{A} \cup \mathcal{F}^\sharp \cup \mathcal{F}^\bullet \cup \mathcal{C})$, where $\mathcal{F}^\bullet = \{n^\bullet_k \mid n \in \mathcal{F} \wedge k \in \mathbb{N}\}$ is the set of attacker-generated fresh values. Note that $\mathcal{M}^\sharp \subseteq \mathcal{N}$. We abbreviate $\mathcal{T} = \mathcal{M} \cup \mathcal{N}$.

3.3.1. Adversary Model

We use a Dolev-Yao attacker model parametrized by an equational theory E. Its judgements are of the form $T \vdash_E t$ meaning that the intruder can derive term t from the set of terms T. The derivable judgements are defined in a standard way by the three deduction rules in Figure 3.3.

3.3.2. Operational Semantics

We define a transition system with states (tr, th, σ), where

- tr is a *trace* consisting of a sequence of pairs of thread identifiers and events,
- $th : \mathit{TID} \rightharpoonup \mathit{dom}(P) \times \mathit{Evt}(\mathcal{M}^\sharp_P)^*$ are *threads* executing role instances, and
- $\sigma : \mathcal{V}^\sharp \rightharpoonup \mathcal{N}$ is a well-typed ground substitution from instantiated protocol variables to network messages such that $\mathcal{V}^\sharp_P \subseteq \mathit{dom}(\sigma)$.

The trace tr as well as the executing role instance are symbolic (with terms in $\mathcal{M}^\sharp$). The separate substitution σ instantiates these messages to (ground) network messages. The ground trace associated with such a state is $tr\sigma$.

The set Init_P of initial states of protocol P contains all (ϵ, th, σ) satisfying

$$\forall i \in \mathit{dom}(th).\ \exists R \in \mathit{dom}(P).\ th(i) = (R, P(R)^{\sharp i})$$

where all terms in the respective protocol roles are instantiated. The substitution σ is chosen non-deterministically in the initial state.

The rules in Figure 3.4 define the transitions. In these rules, the first premise states that an event heads thread i's role. This event is removed and added together with the thread identifier i to the trace tr. The substitution σ remains unchanged. The second premise of *RECV* requires that the network message $t\sigma$ matching the term t in the receive event is derivable from the intruder's (ground) knowledge $\mathit{IK}(tr)\sigma \cup \mathit{IK}_0$. Here, $\mathit{IK}(tr)$ denotes the (symbolic) intruder knowledge derived from a trace tr as the set of

$$\frac{th(i) = (R, \mathsf{send}(t) \cdot tl)}{(tr, th, \sigma) \to (tr \cdot (i, \mathsf{send}(t)), th[i \mapsto (R, tl)], \sigma)} \; SEND$$

$$\frac{th(i) = (R, \mathsf{rcv}(t) \cdot tl) \quad IK(tr)\sigma \cup IK_0 \vdash_E t\sigma}{(tr, th, \sigma) \to (tr \cdot (i, \mathsf{rcv}(t)), th[i \mapsto (R, tl)], \sigma)} \; RECV$$

$$\frac{th(i) = (R, s \cdot tl) \quad s \in Sig}{(tr, th, \sigma) \to (tr \cdot (i, s), th[i \mapsto (R, tl)], \sigma)} \; SIGNAL$$

Figure 3.4.: Operational semantics

terms in the send events on tr, i.e., $IK(tr) = \{t \mid \exists i.\ (i, \mathsf{send}(t)) \in tr\}$ and IK_0 denotes the intruder's (ground) initial knowledge. We assume that $\mathcal{A} \cup \mathcal{C} \cup \mathcal{F}^\bullet \subseteq IK_0$ and IK_0 is R,Ax-normal. Note that the *SEND* rule implicitly updates this intruder knowledge.

3.4. Security Property Language

Meier et al. [93] define a first-order security property language. In this language, many security properties such as those from [87, 43, 40] can be specified. In this section, we introduce a specification language for security properties based on [93].

Syntax Our property specification language is an instance of first-order logic with formulas in negation normal form (i.e., only atomic formulas can be negated) with variables for terms and for thread identifiers. Quantification is allowed only over thread identifier variables. The language consists of the following formulas.

$$\phi ::= A \mid \neg A \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \forall i.\, \phi' \mid \exists i.\, \phi'$$

Here, A are atomic predicates and the quantified variables i represent thread identifiers. An atomic predicate or negated atomic predicate is called a *literal*. For a trace tr, we define a total ordering $\prec_{tr}$ over events occurring in tr such that $a \prec_{tr} b$ if $tr = tr_1 \cdot a \cdot tr_2 \cdot b \cdot tr_3$ for some tr_1, tr_2, and tr_3. The relation $\prec_{tr}$ is crucial to express security properties that exploit fine-grained control over events, such as *injective synchronization* [43].

The atomic predicates and their meaning are as follows, where $m, m' \in \mathcal{M}^\sharp$ are messages, e, e' are events, i, j are thread-id variables, and R is a role name.

A	$::=$	$i = j$	thread i and thread j are equal
	$\mid$	$t = u$	messages t and u are equal
	$\mid$	$role(i, R)$	thread i executes role R
	$\mid$	$honest(i, R)$	the agent playing role R in thread i's view is honest
	$\mid$	$steps(i, e)$	thread i has executed event e
	$\mid$	$(i, e) \prec (j, e')$	thread i has executed e before thread j has executed e'
	$\mid$	$secret(t)$	the intruder does not know message t

In order to achieve attack preservation, we focus on the fragment of this logic where the predicate $secret(m)$ only occurs positively. We call this language $\mathcal{L}_P$. A *property* is a closed formula of $\mathcal{L}_P$. In examples, we freely use standard abbreviations (e.g., for implication) in formulas if there is an equivalent negation normal form in $\mathcal{L}_P$.

Semantics We define the semantics of our language $\mathcal{L}_P$. Recall that $\mathcal{A}_H$ denotes the set of honest agents. Let (tr, th, σ) be a state of the protocol P and let ϑ be a substitution such that $ran(\vartheta) \subseteq dom(th)$. For an equational theory E, we define formula satisfaction, $(tr, th, \sigma, \vartheta) \vDash_E \phi$, as follows:

$$
\begin{array}{lll}
(tr, th, \sigma, \vartheta) \vDash_E i = j & \text{iff} & \vartheta(i) = \vartheta(j) \\
(tr, th, \sigma, \vartheta) \vDash_E t = u & \text{iff} & t\sigma =_E u\sigma \\
(tr, th, \sigma, \vartheta) \vDash_E role(i, R) & \text{iff} & \pi_1(th(\vartheta(i))) = R \\
(tr, th, \sigma, \vartheta) \vDash_E honest(i, R) & \text{iff} & R^{\vartheta(i)}\sigma \in \mathcal{A}_H \\
(tr, th, \sigma, \vartheta) \vDash_E steps(i, e) & \text{iff} & (\vartheta(i), e^{\sharp\vartheta(i)}) \in tr \\
(tr, th, \sigma, \vartheta) \vDash_E (i, e) \prec (j, e') & \text{iff} & (\vartheta(i), e^{\sharp\vartheta(i)}) \prec_{tr} (\vartheta(j), e'^{\sharp\vartheta(i)}) \\
(tr, th, \sigma, \vartheta) \vDash_E secret(t) & \text{iff} & IK(tr)\sigma \cup IK_0 \vdash_E t\sigma \text{ is not derivable} \\
(tr, th, \sigma, \vartheta) \vDash_E \neg A & \text{iff} & \text{not } (tr, th, \sigma, \vartheta) \vDash_E A \\
(tr, th, \sigma, \vartheta) \vDash_E \phi_1 \wedge \phi_2 & \text{iff} & (tr, th, \sigma, \vartheta) \vDash_E \phi_1 \text{ and } (tr, th, \sigma, \vartheta) \vDash_E \phi_2 \\
(tr, th, \sigma, \vartheta) \vDash_E \phi_1 \vee \phi_2 & \text{iff} & (tr, th, \sigma, \vartheta) \vDash_E \phi_1 \text{ or } (tr, th, \sigma, \vartheta) \vDash_E \phi_2 \\
(tr, th, \sigma, \vartheta) \vDash_E \forall i.\, \phi' & \text{iff} & (tr, th, \sigma, \vartheta[i \mapsto tid]) \vDash_E \phi' \text{ for all } tid \in dom(th) \\
(tr, th, \sigma, \vartheta) \vDash_E \exists i.\, \phi' & \text{iff} & (tr, th, \sigma, \vartheta[i \mapsto tid]) \vDash_E \phi' \text{ for some } tid \in dom(th)
\end{array}
$$

We write $(tr, th, \sigma, \vartheta) \nvDash_E \phi$ if $(tr, th, \sigma, \vartheta) \vDash_E \phi$ does not hold. If ϕ is a closed formula, we write $(tr, th, \sigma) \vDash_E \phi$ instead of $(tr, th, \sigma, \vartheta) \vDash_E \phi$. We call a protocol state $s = (tr, th, \sigma)$ an *attack on* ϕ if $s \nvDash_E \phi$.

In the following example, we present our formalizations of secrecy and authentication properties for $\mathsf{IKE_m}$.

Example 8 (Properties of $\mathsf{IKE_m}$)**.** We express the secrecy of the Diffie-Hellman key $\mathsf{exp}(Gb, x)$ for role A of the protocol $\mathsf{IKE_m}$ of Example 7 as follows.

$$\phi_s \;=\; \forall j.\ (role(j, A) \wedge honest(j, [A, B]) \wedge steps(j, \mathsf{Secret})) \Rightarrow secret(\mathsf{exp}(Gb^j, x^j)).$$

where $honest(j, [A, B])$ is an abbreviation of the conjunction $honest(j, A) \wedge honest(j, B)$. Intuitively, ϕ_s states that whenever an agent a playing role A completes his run with another agent b playing role B and both a and b are honest, the key $\mathsf{exp}(Gb, x)$ is secret. The completion of the run is expressed by the presence of a Secret signal event in a trace.

We formalize non-injective agreement of A with B [87] on the nonces na and nb and the Diffie-Hellman half-keys $\mathsf{exp}(g, x)$ and $\mathsf{exp}(g, y)$ by

$$
\begin{aligned}
\phi_a \;=\; & \forall j.\ (role(j, A) \wedge honest(j, [A, B]) \wedge steps(j, \mathsf{Commit})) \\
& \Rightarrow (\exists k.\, role(k, B) \wedge steps(k, \mathsf{Running}) \wedge \\
& \quad \langle A^j, B^j, na^j, Nb^j, \mathsf{exp}(g, x^j), Gb^j \rangle = \langle A^k, B^k, Na^k, nb^k, Ga^k, \mathsf{exp}(g, y^k) \rangle).
\end{aligned}
$$

The formula ϕ_a states that whenever an agent a playing role A completes a run of the protocol with another agent b playing role B and both agents are honest, then b has previously been running the protocol with a. Moreover, a and b agree on na and nb and the Diffie-Hellman half-keys $\mathsf{exp}(g, x)$ and $\mathsf{exp}(g, y)$. The authentication on these values is formulated by the equality in the formula. This equality also expresses the agreement on their respective communication partners. ♠

We show in the following proposition that there is no distinction between attacks that stem from the same equivalence class.

Proposition 1. *Let $\phi \in \mathcal{L}_P$, (tr, th, σ) be a reachable state of P, and σ' be a substitution such that $dom(\sigma') = dom(\sigma)$ and $X\sigma =_E X\sigma'$ for all $X \in dom(\sigma)$. Then the following holds.*

(i) (tr, th, σ') *is a reachable state in P, and*

(ii) if $(tr, th, \sigma) \not\vDash \phi$ *then* $(tr, th, \sigma') \not\vDash \phi$.

Proof Sketch: To show point (i), we proceed by induction on the number n of transitions leading to the state (tr, th, σ) and perform a case distinction on the last rule that has been applied. The non-trivial case is when the *RECV* rule is applied. In this case, as the intruder deduction does not distinguish equivalent terms, it follows that (tr, th, σ') is reachable in P. To show point (ii), we proceed by a routine induction on ϕ.

4. Security Protocols Abstractions

In this chapter, we will present three types of security protocol abstractions:

Typed abstractions transform a term's structure by reordering or removing fields and by splitting or removing cryptographic operations. The same transformations are applied to all terms of a given type and its subtypes.

Untyped abstractions complement typed ones with additional simplifications: the removal of unprotected atoms and variables and of redundant subterms.

Splitting abstractions change the structures of security protocols by splitting protocol events into several ones.

Our main results are soundness theorems for these abstractions. They ensure that any attack on a given property of the original protocol translates to an attack on a related property in the abstracted protocol. As we will explain, these results only hold under certain conditions on the protocol and the property. For our running example, we focus on the first two classes of protocol abstractions. In Section 4.3, we provide a separate discussion of splitting abstractions.

4.1. Typed Protocol Abstractions

Our typed abstractions are specified by a list of recursive equations subject to some conditions on their shape. We define their semantics in terms of a simple Haskell-style functional program. We use both pattern matching on terms and subtyping on types to select the equation to be applied to a given term. This ensures that terms of similar types are transformed in a uniform manner. We also extend typed protocol abstractions to protocol specifications, traces, substitutions, and security properties. At the end of this section, we apply our typed protocol abstractions to simplify the $\mathsf{IKE_m}$ protocol.

4.1.1. Specification of Typed Protocol Abstractions

Syntax

Let $\mathcal{W} = \{\mathcal{W}_\tau\}_{\tau \in \mathcal{Y}}$ be a family of *pattern variables* disjoint from $\mathcal{V}$. We define the set of *patterns* by $\mathcal{P} = \mathcal{T}(\mathcal{W}, \emptyset)$. A pattern $p \in \mathcal{P}$ is called *linear* if each (pattern) variable occurs at most once in p. We extend the typing function Γ to patterns by setting $\Gamma(X) = \tau$ if and only if $X \in \mathcal{W}_\tau$ and then lifting it homomorphically to all patterns. Our typed protocol abstractions are instances of the following recursive function specifications.

Definition 4. A *function specification* $F_f = (f, E_f)$ consists of an unary function symbol $f \notin \Sigma^1$ and a list of equations

$$E_f = [f(p_1) = u_1, \dots, f(p_n) = u_n],$$

where each $p_i \in \mathcal{P}$ is a linear pattern such that $u_i \in \mathcal{T}_{\Sigma^{\geq 1} \cup \{f\}}(vars(p_i))$ for all $i \in \widetilde{n}$, i.e., u_i consists of variables from p_i and function symbols from $\Sigma^{\geq 1} \cup \{f\}$.

Intuitively, each clause $f(p_i) = u_i \in E_f$ describes how to abstract a term whose type is a subtype of p_i's. The pattern u_i represents the shape of abstracted terms. For instance, the clause $f(\{\!|X|\!\}_K) = f(X)$ with $X, K : \mathsf{msg}$ describes an abstraction that removes symmetric encryptions. Note that E_f is a list, not a set. The reason is that the order of clauses in E_f affects the transformation of a given term (see Program 1).

We say f *is homomorphic for* c where $c \in \Sigma^n$ and $n \geq 1$ if either c does not occur in E_f or there are a clause $(f(p) = q) \in E_f$ and variables $X_1, \dots, X_n$ of type msg such that

$$\begin{array}{rcl} p & = & c(X_1, \dots, X_n), \\ q & = & c(f(X_1), \dots, f(X_n)). \end{array}$$

We use vectors (lists) of terms $\bar{t} = [t_1, \dots, t_n]$ for $n > 0$. We define $set(\bar{t}) = \{t_1, \dots, t_n\}$ and $\widehat{f}(\bar{t}) = \langle f(t_1), \dots, f(t_n) \rangle$, the elementwise application of a function f to a vector where the result is converted to a tuple (with the convention $\langle t \rangle = t$). We extend *split* to vectors by $split(\bar{t}) = split(set(\bar{t}))$. We define three sets of function symbols occurring in R and Ax as follows.

$$\begin{array}{rcl} \mathsf{C}_{\mathsf{Key}} & = & \bigcup \{ct(t) \cup ct(u) \mid d(c(x_1, \dots, x_n, t), u) \to x_j \in R\} \\ \mathsf{C}_{\mathsf{R}} & = & \{c \mid d(c(x_1, \dots, x_n, t), u) \to x_j \in R\} \\ \mathsf{C}_{\mathsf{Ax}} & = & \bigcup \{ct(s) \cup ct(t) \mid \{s, t\} \in Ax\} \end{array}$$

Intuitively, $\mathsf{C}_{\mathsf{Key}}$ is the set of all non-constant function symbols occurring in a matching key pair of an $\mathsf{R1}$ rule. C_{R} is the set of function symbols occurring in proper non-key subterms of an $\mathsf{R1}$ rule's left-hand side. Finally, C_{Ax} is the set of all non-constant function symbols occurring in an axiom. Below, we specify typed protocol abstractions based on these sets and the following function:

$$pp(c) = \{j \mid d(c(x_1, \dots, x_n, t), u) \to x_j \in R \text{ or } d(c(x_1, \dots, x_n)) \to x_j \in R\}.$$

Intuitively, the function $pp(c)$ returns the set of extractable indices of a function symbol c occurring in an $\mathsf{R1}$ or $\mathsf{R2}$ rewrite rule. We also define

$$\begin{array}{rcl} top(Ax) & = & \{top(s) \mid \{s, s'\} \in Ax\}, \\ \mathsf{C}_{\mathsf{R,Ax}} & = & \bigcup \{ct(l) \mid l \in R \wedge top(l) \in top(Ax)\}. \end{array}$$

Intuitively, $top(Ax)$ is the set of top constructors of axioms in Ax, and $\mathsf{C}_{\mathsf{R,Ax}}$ is the set of all function symbols occurring in the left-hand side l of a rewrite rule such that l shares the same top constructor with an axiom.

We now define our class of typed protocol abstractions.

Definition 5 (Typed protocol abstractions)**.** A function specification $F_f = (f, E_f)$ is a *typed abstraction* if each equation in E_f has the form

$$f(c(p_1, \dots, p_n)) = \langle e_1, \dots, e_d \rangle$$

where for each $i \in \widetilde{d}$ we have either

(a) $e_i = f(q)$ such that $q \in \mathit{split}(p_j)$ for some $j \in \widetilde{n}$, or

(b) $e_i = c(\widehat{f}(\overline{q_1}), \dots, \widehat{f}(\overline{q_n}))$ such that $\mathit{set}(\overline{q_j}) \subseteq \mathit{split}(p_j)$ for all $j \in \widetilde{n}$, $c \neq \langle \cdot, \cdot \rangle$, and $c \in \mathsf{C_R}$ implies $\overline{q_n} = [p_n]$, i.e., $\widehat{f}(\overline{q_n}) = f(p_n)$.

Moreover, we require

(i) for all $j \in \mathit{pp}(c)$ we have $\mathit{split}(p_j) \subseteq Q_j$ where

$$Q_j \quad = \quad \bigcup \{ \mathit{set}(\overline{q_j}) \mid \exists i.\ e_i = c(\widehat{f}(\overline{q_1}), \dots, \widehat{f}(\overline{q_n})) \} \cup \{ q \mid \exists i.\ e_i = f(q) \}.$$

(ii) if $c \in \mathsf{C_{Ax}} \cup \mathsf{C_{Key}} \cup \mathsf{C_{R,Ax}}$ then f is homomorphic for c. Note that in this case, we have $d = 1$ and $e_1 = c(f(X_1), \dots, f(X_n))$ is an instance of (b).

Intuitively, the abstractions can only weaken the cryptographic protection of terms, but never strengthen it. Each defining equation maps a term with top-level symbol c to a tuple whose components have the form (a) or (b). Form (a) allows us to pull fields out of the scope of c, hence removing c's protection. Using form (b) we can reorder or remove fields in each argument of c. Form (b) is subject to two conditions. First, we disallow this form for pairs to obtain the simple shape $f(\langle p_1, p_2 \rangle) = \widehat{f}(\overline{q})$ by (a). Second, we do not permit the reordering or removal of fields in key positions, i.e., in the last argument of $c \in \mathsf{C_R}$. Moreover, by point (i), all fields of extractable arguments, i.e., elements of $\mathit{split}(p_j)$ for $j \in \mathit{pp}(c)$, must be present in some e_i and point (ii) requires that the abstraction is homomorphic for function symbols c that occur in axioms or in keys ($c \in \mathsf{C_{Ax}} \cup \mathsf{C_{Key}}$), or that are common top constructors of the left-hand side of a rewrite rule and an axiom ($c \in \mathsf{C_{R,Ax}}$). We say that f is homomorphic for a set of function symbols S if f is homomorphic for all $c \in S$. In Subsection 4.1.1, we will discuss the necessity of these conditions in more detail. The following example, we present a typed abstraction that specifies different message transformations captured by our theory.

Example 9. Let $F_f = (f, E_f)$ be the typed abstraction where E_f consists of the following equations with variables $X : \gamma_c$, $Y : \mathsf{Nonce}$, and $Z, U, V : \mathsf{msg}$:

$$\begin{aligned} f(\langle X, Y, Z \rangle) &= \langle f(Y), f(X), f(Z) \rangle \\ f(\mathsf{kdf}(X, Y, U, V)) &= \langle \mathsf{kdf}(f(X), f(Y)), \mathsf{kdf}(f(U)) \rangle \\ f(\{\!| X, Y, Z |\!\}_U) &= \langle \{\!| f(X), f(Y) |\!\}_{f(U)}, f(Y), \{\!| f(Z) |\!\}_{f(U)} \rangle \end{aligned}$$

Note that X and Y only match the constant c and a nonce, respectively. The first clause swaps the first two fields in n-tuples for $n \geq 3$. The second one splits a kdf hash into two, dropping the field V. The last clause splits an encryption: the pair

$$\begin{array}{l}\textbf{fun } f(t) = \textbf{case } t \textbf{ of} \\ \quad \|_{(f(p)=u)\in E_f^+} \ \ p \mid \Gamma(t) \preccurlyeq \Gamma(p) \Rightarrow u\end{array}$$

Program 1: Functional program f resulting from $F_f = (f, E_f)$.

$\langle f(X), f(Y)\rangle$ and $f(Z)$ are encrypted separately with the key $f(U)$ and $f(Y)$ is pulled out of the encryption. Note that by condition (i) of Definition 5, we cannot directly remove plaintext fields from encryptions. To achieve this, we pull such fields out of encryptions to the top-level. This may require a combination of several abstractions if there are multiple layers of cryptographic protection. At the top-level, the fields are no longer protected and can be removed using untyped abstractions. In Section 5.1, we will discuss our heuristics to determine sequences of abstractions automatically. ♠

Semantics

The semantics of a typed abstraction F_f is given by the Haskell-style functional program f (Program 1). We often identify this program with the function symbol f when no confusion arises. Note that F_f itself may not define a total function, i.e., there are terms whose types are not subtypes of any patterns' types in E_f. For instance, the function specification F_f in Example 9 is not total, because the first clause in E_f only abstracts tuples of at least three components, and therefore tuples of two components, e.g., $\langle X, Y\rangle$ cannot be transformed. To ensure totality, we use the extended function specification $(f, E_f^+) = (f, E_f \cdot E_f^0)$, where $f(g(Z_1, \ldots, Z_n)) = g(f(Z_1), \ldots, f(Z_n)) \in E_f^0$ for each $g \in \Sigma^n$ with $n \geq 1$ such that $Z_i : \mathsf{msg}$ for all $i \in \widetilde{n}$, and $f(Z) = Z$ with $Z : \mathsf{msg}$ is the last clause in E_f^0. Intuitively, each non-constant function symbol c is transformed homomorphically by exactly one clause in E_f^0. Moreover, we require that all variables in E_f^0 are of the type msg to cover all possible types, hence all terms. Note that the last clause $f(Z) = Z$ in E_f^0 is used to transform (only) atoms and variables. We assume E_f and E_f^0 do not share variables. The **case** statement in Program 1 has a clause

$$p \mid \Gamma(t) \preccurlyeq \Gamma(p) \Rightarrow u$$

for each equation $f(p) = u$ of E_f^+. Such a clause is enabled if (1) the term t matches the pattern p, i.e., $t = p\theta$ for some substitution θ, and (2) θ is well-typed, i.e., $\Gamma(t) \preccurlyeq \Gamma(p)$. The first enabled clause is executed. Hence, the equations E_f^0 serve as fall-back clauses, which cover the terms not handled by E_f.

Note that in Program 1, we use types to define how a given term is transformed. Types allow us to abstract terms of the same shape in different ways, as illustrated in the following example.

Example 10. Let us consider two terms $\{\!|na|\!\}_k$ and $\{\!|nb|\!\}_k$ for some nonces na, nb, and k. The property we are interested in is the secrecy of na. We want to define an abstraction that removes symmetric encryption in $\{\!|nb|\!\}_k$ but keeps $\{\!|na|\!\}_k$ unchanged.

Since the types of $\{\!|na|\!\}_k$ and $\{\!|nb|\!\}_k$ are not in a subtyping relation, it is straightforward to define this abstraction by two separate clauses, i.e.,

$$\begin{array}{rcl} f(\{\!|X|\!\}_K) & = & \{\!|f(X)|\!\}_{f(K)} \\ f(\{\!|Y|\!\}_Z) & = & f(Y) \end{array}$$

where $X\colon \beta_{na}$, $Y\colon \beta_{nb}$, and $K, Z\colon \mathsf{msg}$. ♠

To clarify the restrictions in Definition 5, we motivate them in the following paragraphs.

Motivation for conditions on typed protocol abstractions

In order to achieve soundness, and in particular, to preserve attacks on secrecy properties, our typed protocol abstractions need to preserve deducibility, i.e., whenever a term t is deducible from a set of terms T, we also have that $f(t)$ is deducible from $f(T)$. In this subsection, we further explain the necessity of the different conditions in Definition 5 to achieve deducibility preservation. For our examples here, we work with a set of rewrite rules that consists of projections and an empty set of axioms. Additional rewrite rules and axioms or alternative settings are specified explicitly.

In the specification of typed protocol abstractions, form (a) allows us to remove cryptographic protections. However, it disallows introducing fields that are not part of original terms, as these fields may not be deducible from the corresponding abstracted terms. We present this situation in the following example.

Example 11. Consider a function specification $F_f = (f, E_f)$ where E_f consists of the homomorphic clauses for the hash function symbols h and g, and the equation $f(\langle g(X), h(Y)\rangle) = \langle f(g(Y)), f(h(X))\rangle$ with variables $X, Y\colon \mathsf{msg}$. Intuitively, this equation swaps the plaintexts in the first and the second components of the abstracted pair. Note that $h(X) \notin \mathit{split}(\langle g(X), h(Y)\rangle)$. Let $t = \langle g(a), h(b)\rangle$ and $u = h(b)$ where $a, b \in \mathcal{C}$. Then we have $t \vdash_E u$. But $f(u) = h(b)$ is not deducible from $f(t) = \langle g(b), h(a)\rangle$. Therefore, deducibility is not preserved under f. ♠

For a similar reason, form (b) does not allow occurrences of new fields, which is ensured by the constraint $\mathit{set}(\overline{q_j}) \subseteq \mathit{split}(p_j)$ for all $j \in \widetilde{n}$. In addition, this form forbids duplicating keys for $c \in \mathsf{C_R}$. This together with condition (ii) ensures that abstracted keys remains proper to decrypt the corresponding abstracted encryptions, which is required for deducibility preservation. The following example shows that duplicating fields in key positions may break deducibility preservation.

Example 12. Consider $\Sigma = \{\langle\cdot,\cdot\rangle, d, c, k\}$ and the following additional rewrite rule

$$d(c(x_1, x_2), k(x_2)) \to x_1.$$

Let $F_f = (f, E_f)$ be such that E_f consists of the homomorphic clauses for the function symbols $\langle\cdot,\cdot\rangle$, d, and k, and the equation $f(c(X, Y)) = c(f(X), \langle f(Y), f(Y)\rangle)$ with variables X, Y : msg. Note that this equation abstracts a term $c(u_1, u_2)$ by duplicating its second argument. We assume Let $t = c(a_1, a_2)$ and $u = k(a_2)$ where $a_i \in \mathcal{F}$ for $i \in \widetilde{3}$. Then we have

$$\begin{array}{rcl} f(t) & = & c(a_1, \langle a_2, a_2 \rangle), \\ f(u) & = & k(a_2), \\ f(a_1) & = & a_1. \end{array}$$

Clearly, it holds that $t, u \vdash_E a_1$ because the term $d(c(a_1, a_2), k(a_2))$ is deducible from t, u and $d(c(a_1, a_2), k(a_2)) \to a_1$. But $f(a_1) = a_1$ is not deducible from $f(t)$ and $f(u)$, as the correct decryption key $k(\langle a_2, a_2 \rangle)$ cannot be derived from $f(t)$ and $f(u)$. ♠

Next, condition (i) requires that all fields of extractable arguments must occur in some component e_i. Without this condition, the adversary may lose necessary information to deduce certain terms and thus deducibility may not be preserved. The following example presents such a case.

Example 13. Consider a function specification $F_f = (f, E_f)$ where E_f consists of the equation $f(\langle X, Y \rangle) = f(X)$ with variables $X, Y \colon \mathsf{msg}$, $Z \colon \mathsf{Nonce}$. Intuitively, this clause removes the second field of a pair and the second clause keeps fresh values unchanged. We assume the set of rewrite rules is $R = R_{cs}$ and we do not have any axioms. Let $t = \langle a, b \rangle$ where $a, b \in \mathcal{F}$. Then we have $f(t) = f(a) = a$, $f(b) = b$, and $t \vdash_E b$. But $f(b) = b$ is not deducible from a. ♠

We have shown in Example 12 that duplicating terms in key positions may turn a matching key pair into an unmatched one and therefore lead to a violation of deducibility preservation. The introduction of (b) is not sufficient to prevent manipulating these terms. In particular, non-homomorphic modifications of terms occurring in keys may result in a similar issue as shown in the following example.

Example 14. Let us consider $\Sigma = \{\langle \cdot, \cdot \rangle, d, c, h, k_1, k_2\}$ where $d, c, h \in \Sigma^2$, $k_1, k_2 \in \Sigma^1$ and the following additional rewrite rule

$$d(c(x_1, k_1(\langle x_2, x_3 \rangle)), k_2(h(x_2, x_3))) \to x_1.$$

Let $F_f = (f, E_f)$ be such that E_f consists of the homomorphic clauses for d, c, h, k_1, k_2, and the equation $f(\langle X, Y \rangle) = \langle f(Y), f(X) \rangle$ with variables $X, Y : \mathsf{msg}$. Note that this clause reorders fields in a pair. Let $t = c(a_1, k_1(\langle a_2, a_3 \rangle))$ and $u = k_2(h(a_2, a_3))$ where $a_i \in \mathcal{F}$ for $i \in \widetilde{3}$. Then we have

$$\begin{array}{rcl} f(t) & = & c(a_1, k_1(\langle a_3, a_2 \rangle)), \\ f(u) & = & k_2(h(a_2, a_3)), \\ f(a_1) & = & a_1. \end{array}$$

Clearly, the keys $k_1(\langle a_3, a_2 \rangle)$ and $k_2(h(a_2, a_3))$ no longer match. It is not hard to see that a_1 is deducible from t and u, but $f(a_1) = a_1$ is not deducible from $f(t)$ and $f(u)$. The reason is that f is homomorphic for h, but not for $\langle \cdot, \cdot \rangle$. As a consequence, f abstracts the keys $k_1(\langle a_2, a_3 \rangle)$ and $k_2(h(a_2, a_3))$ in two different ways, resulting in an unmatched abstracted key pairs. ♠

In order to address the issue described in Example 14, the first part of condition (ii) in Definition 5 requires that f is homomorphic for all function symbols in $\mathsf{C}_{\mathsf{Key}}$. This fixes the issue in Example 14. The second part of condition (ii) requires that f is homomorphic for all functions symbols in C_{Ax}. This is to ensure that whenever the adversary deduces a term using rule Eq, he can also use the same rule to deduce the corresponding abstracted term. We provide a simple example to illustrate the necessity of this restriction.

Example 15. We assume the additional axiom $\mathsf{exp}(\mathsf{exp}(g, X), Y) = \mathsf{exp}(\mathsf{exp}(g, Y), X)$ that models Diffie-Hellman exponentiation. Let $F_f = (f, E_f)$ be a function specification, where $E_f = [f(\mathsf{exp}(X, Y)) = \mathsf{exp}(f(Y), f(X))]$. We also assume that $X, Y : \mathsf{msg}$. Note that the first clause is not homomorphic for the function symbol exp. Let $t = \mathsf{exp}(\mathsf{exp}(g, a), b)$ and $u = \mathsf{exp}(\mathsf{exp}(g, b), a)$, where $a, b \in \mathcal{F}$. Then we have $t \vdash_E u$ because $t =_{Ax} u$. But $f(u) = \mathsf{exp}(a, \mathsf{exp}(b, g)) \neq_E f(t) = \mathsf{exp}(b, \mathsf{exp}(a, g))$. Hence, $f(u)$ is not deducible from $f(t)$. ♠

Finally, the third part of condition (ii) ensures that f is homomorphic for all function symbols occurring in a left-hand side l of a rewrite rule $l \to r$ if l and s have the same top constructor for some pair $\{s, s'\} \in Ax$. This restricts the interference of f with an original term t when an axiom is applicable to t at the root. We show in the following example that deducibility is not preserved if this restrictions is not satisfied.

Example 16. We assume the additional axiom $c(\mathsf{h}_1(X) = c(\mathsf{h}_2(X))$ and rewrite rule $c(\mathsf{h}_2(X)) \to X$. Let $F_f = (f, E_f)$ be a function specification, where $E_f = [f(\mathsf{h}_1(X)) = f(X)]$ and $X : \mathsf{msg}$. Let us consider a term $t = c(\mathsf{h}_1(a))$ for $a \in \mathcal{C}$. Then, we have $t \to_{R,Ax} a$ and therefore $t \vdash_E a$. But $f(t) = f(c(\mathsf{h}_1(a))) = c(a)$ and thus $a = f(a)$ is not deducible from $f(t)$. To achieve deducibility, it is sufficient to ensure that f is homomorphic for h_1 (note that in this example we already have that f is homomorphic for c). ♠

Of course, weaker conditions than the ones in Definition 5 may exist. However, our conditions are relatively simple and can be checked efficiently.

Transforming protocols, substitutions, and security properties

We extend f to events, event sequences, and traces by applying f to the terms they contain and to substitutions and protocols by applying f to the terms in their range.

Let $F_f = (f, E_f)$ be a typed abstraction. We define

(i) $f(ev(t)) = ev(f(t))$ for events $ev(t) \in Evt(\mathcal{T}) \setminus Sig$ and $f(sig) = sig$ for $sig \in Sig$.

(ii) For event sequences, $f(\epsilon) = \epsilon$ and $f(e \cdot tl) = f(e) \cdot f(tl)$; the lifting to traces is defined analogously.

(iii) $f(\rho) = \{(X, f(X\rho)) \mid X \in dom(\rho)\}$ for maps $\rho : \mathcal{V} \rightharpoonup D$ such as substitutions and protocols.

Similarly, we extend f to formulas ϕ of our property language by applying f to all terms occurring in ϕ.

$$\begin{array}{rclrcl}
f(i=i') & = & i=i' & f(\mathit{secret}(m)) & = & \mathit{secret}(f(m)) \\
f(m=m') & = & f(m)=f(m') & f(\neg A) & = & \neg f(A) \\
f(\mathit{role}(i,R)) & = & \mathit{role}(i,R) & f(\phi_1 \wedge \phi_2) & = & f(\phi_1) \wedge f(\phi_2) \\
f(\mathit{honest}(i,R)) & = & \mathit{honest}(i,R) & f(\phi_1 \vee \phi_2) & = & f(\phi_1) \vee f(\phi_2) \\
f(\mathit{steps}(i,e)) & = & \mathit{steps}(i,f(e)) & f(\forall i.\, \phi') & = & \forall i.\, f(\phi') \\
f((i,e) \prec (j,e')) & = & (i,f(e)) \prec (j,f(e')) & f(\exists i.\, \phi') & = & \exists i.\, f(\phi')
\end{array}$$

The following proposition states that whenever P is a protocol, $f(P)$ is also a protocol. This allows us to create a complex protocol abstraction by composing several simpler ones.

Proposition 2. *Let $F_f = (f, E_f)$ be a function specification and P be a protocol. Then $f(P)$ is also a protocol.*

Proof. Note that f maps roles to roles and is the identity on variables. Hence by Definition 3, it is clear that $f(P)$ is a protocol. □

4.1.2. Application of Typed Protocol Abstractions

We use our typed protocol abstractions to transform the original $\mathsf{IKE_m}$ protocol. Finding abstractions is fully automated by our tool using a heuristic (see Section 5). However, the resulting abstractions can be counterintuitive. Therefore, we present here a simple strategy that we apply to our running example below: we start by identifying the terms that appear in the $\mathit{secret}(\cdot)$ predicates and the equations of the desired properties. Then we determine the cryptographic operations that are essential to achieve these properties and try to remove all other terms and operations.

Example 17 (from $\mathsf{IKE_m}$ to $\mathsf{IKE_m^1}$)**.** We recall the specification of $\mathsf{IKE_m}$ below.

$$\begin{array}{rcl}
\mathsf{IKE_m}(A) & = & \mathsf{send}(sPIa, o, sA1, \mathsf{exp}(g,x), na) \cdot \mathsf{rcv}(sPIa, SPIb, sA1, Gb, Nb) \cdot \\
& & \mathsf{Running} \cdot \mathsf{send}(sPIa, SPIb, \{\!|A, B, AUTHaa, sA2, tSa, tSb|\!\}_{SKa}) \cdot \\
& & \mathsf{rcv}(sPIa, SPIb, \{\!|B, AUTHba, sA2, tSa, tSb|\!\}_{SKa}) \cdot \mathsf{Secret} \cdot \mathsf{Commit} \\
\\
\mathsf{IKE_m}(B) & = & \mathsf{rcv}(SPIa, o, sA1, Ga, na) \cdot \mathsf{send}(SPIa, sPIb, sA1, \mathsf{exp}(g,y), nb) \cdot \\
& & \mathsf{rcv}(SPIa, sPIb, \{\!|A, B, AUTHab, sA2, tSa, tSb|\!\}_{SKb}) \cdot \mathsf{Running} \cdot \\
& & \mathsf{send}(SPIa, sPIb, \{\!|B, AUTHbb, sA2, tSa, tSb|\!\}_{SKb}) \cdot \mathsf{Secret} \cdot \mathsf{Commit}
\end{array}$$

where the session keys and the authenticators are

$$\begin{array}{rcl}
SKa & = & \mathsf{kdf}(na, Nb, \mathsf{exp}(Gb, x), sPIa, SPIb), \\
SKb & = & \mathsf{kdf}(Na, nb, \mathsf{exp}(Ga, y), SPIa, sPIb), \\
AUTHaa & = & \mathsf{mac}(\mathsf{sh}(A,B), sPIa, o, sA1, \mathsf{exp}(g,x), na, Nb, \mathsf{prf}(SKa, A)), \\
AUTHab & = & \mathsf{mac}(\mathsf{sh}(B,A), SPIa, o, sA1, Ga, Na, nb, \mathsf{prf}(SKb, A)), \\
AUTHba & = & \mathsf{mac}(\mathsf{sh}(A,B), sPIa, SPIb, sA1, Gb, Nb, na, \mathsf{prf}(SKa, B)), \\
AUTHbb & = & \mathsf{mac}(\mathsf{sh}(B,A), SPIa, sPIb, sA1, \mathsf{exp}(g,y), nb, Na, \mathsf{prf}(SKb, B)).
\end{array}$$

In order to preserve the secrecy of the Diffie-Hellman key $\mathsf{exp}(\mathsf{exp}(g, x), y)$ and the agreement on na, nb, $\mathsf{exp}(g, x)$, and $\mathsf{exp}(g, y)$, we have to keep either the mac or the symmetric encryption with SK (see Examples 7 and 8). We choose to remove the encryption as this allows us to later remove additional fields (e.g., $sA2$) using untyped abstractions. We remove $sPIa$, $sA1$ from $AUTHaa$ and $SPIa$, $sA1$ from $AUTHab$, as these terms are irrelevant to the properties of interest. Similarly, we remove $sPIa$, $SPIb$, $sA1$ from $AUTHba$ and $SPIa$, $sPIb$, $sA1$ from $AUTHbb$. Note that we keep o in $AUTHaa$ and $AUTHab$ to prevent unifiability with $AUTHba$ or $AUTHbb$, and hence potential false negatives. From the authenticators, we remove the application of prf together with agent variables underneath. In this way, the session keys are still protected by the macs. From the session keys, we remove all fields underneath the kdf, except the Diffie-Hellman keys, as these keys are part of the properties. This leads us to the typed abstraction $F_{f_1} = (f_1, E_{f_1})$ where E_{f_1} is defined by the following equations with variables $X_3 : \gamma_o$, $Y_3 \colon \mathsf{Nonce}$, $U \colon \mathsf{kdf}(\mathsf{msg})$, and all remaining pattern variables are of type msg.

$$
\begin{array}{rcl}
f_1(\{|X, Y|\}_Z) & = & \langle f_1(X), f_1(Y)\rangle \\
f_1(\mathsf{mac}(X_1, \ldots, X_8)) & = & \mathsf{mac}(\widehat{f_1}([X_1, X_3, X_5, X_6, X_7, X_8])) \\
f_1(\mathsf{mac}(Y_1, \ldots, Y_8)) & = & \mathsf{mac}(\widehat{f_1}([Y_1, Y_5, Y_6, Y_7, Y_8])) \\
f_1(\mathsf{kdf}(Z_1, \ldots, Z_5)) & = & \mathsf{kdf}(f_1(Z_3)) \\
f_1(\mathsf{prf}(U, Z)) & = & f_1(U) \\
f_1(\mathsf{exp}(U_1, U_2)) & = & \mathsf{exp}(f_1(U_1), f_1(U_2)), \\
f_1(\mathsf{sh}(U_1, U_2)) & = & \mathsf{sh}(f_1(U_1), f_1(U_2)), \\
f_1(\langle U_3, U_4\rangle) & = & \langle f_1(U_3), f_1(U_4)\rangle,
\end{array}
$$

Intuitively, the first clause removes symmetric encryptions. The second clause removes the second and the forth fields from a mac's argument. In a slightly different way, the third clause abstracts a mac by removing the second, the third, and the forth fields from the mac's argument. The forth clause removes all fields from a kdf's argument, except the third field. The fifth clause simplifies a prf hash by removing the second field of the hash's argument together with the function symbol prf itself. As we do not reorder fields in tuples and want to keep the Diffie-Hellman and the pre-shared keys unchanged, the last three clauses are homomorphic.

Applying f_1 to $\mathsf{IKE_m}$ we obtain $\mathsf{IKE_m^1}$. In the specification of $\mathsf{IKE_m^1}$ given below, we underline the session keys and the authenticators to emphasize that these terms have changed compared to their original versions.

$$
\begin{array}{rcl}
S_{\mathsf{IKE_m^1}}(A) & = & \mathsf{send}(sPIa, o, sA1, \mathsf{exp}(g, x), na) \cdot \mathsf{rcv}(sPIa, SPIb, sA1, Gb, Nb) \cdot \\
& & \mathsf{Running} \cdot \mathsf{send}(sPIa, SPIb, A, B, \underline{AUTHaa}, sA2, tSa, tSb) \cdot \\
& & \mathsf{rcv}(sPIa, SPIb, B, \underline{AUTHba}, sA2, tSa, tSb) \cdot \mathsf{Secret} \cdot \mathsf{Commit}
\end{array}
$$

$$
\begin{array}{rcl}
S_{\mathsf{IKE_m^1}}(B) & = & \mathsf{rcv}(SPIa, o, sA1, Ga, Na) \cdot \mathsf{send}(SPIa, sPIb, sA1, \mathsf{exp}(g, y), nb) \cdot \\
& & \mathsf{rcv}(SPIa, sPIb, A, B, \underline{AUTHab}, sA2, tSa, tSb) \cdot \mathsf{Running} \cdot \\
& & \mathsf{send}(SPIa, sPIb, B, \underline{AUTHbb}, sA2, tSa, tSb) \cdot \mathsf{Secret} \cdot \mathsf{Commit}
\end{array}
$$

where $\underline{SKa} = \mathsf{kdf}(\mathsf{exp}(Gb, x))$, $\underline{SKb} = \mathsf{kdf}(\mathsf{exp}(Ga, y))$, and the authenticators are

$$\begin{array}{rcl}
\underline{AUTHaa} & = & \mathsf{mac}(\mathsf{sh}(A, B), o, \mathsf{exp}(g, x), na, Nb, \underline{SKa}), \\
\underline{AUTHab} & = & \mathsf{mac}(\mathsf{sh}(B, A), o, Ga, Na, nb, \underline{SKb}), \\
\underline{AUTHba} & = & \mathsf{mac}(\mathsf{sh}(A, B), Gb, Nb, na, \underline{SKa}), \\
\underline{AUTHbb} & = & \mathsf{mac}(\mathsf{sh}(B, A), \mathsf{exp}(g, y), nb, Na, \underline{SKb}).
\end{array}$$

In a second step, we will remove most fields in the roles of $\mathsf{IKE}^1_\mathsf{m}$ using untyped abstractions. We detail this step in Section 4.2. The soundness of f_1 is justified in Appendix E.1. ♠

4.1.3. Soundness of Typed Abstractions

To justify the soundness of our abstractions, we show that any attack on a property ϕ of the original protocol P is reflected as an attack on the property $f(\phi)$ of the abstracted protocol $f(P)$. We decompose this into reachability preservation (RP) and an attack preservation (AP) as follows. Our soundness theorem requires that, for all reachable states (tr, th, σ) of P, there is a ground substitution σ' such that

(RP) $(f(tr), f(th), \sigma')$ is a reachable state of $f(P)$, and

(AP) $(tr, th, \sigma) \not\models \phi$ implies $(f(tr), f(th), \sigma') \not\models f(\phi)$.

Our goal is to show (RP) and (AP) for $\sigma' = f(\sigma)$. These properties will require some assumptions about the protocol P, the formula ϕ, and the abstraction f. We will introduce and motivate these assumptions while sketching the proof of the soundness theorem. For the remainder of this section we assume arbitrary but fixed P, ϕ, F_f. By Proposition 1, we also without loss of generality assume that σ is R,Ax-normal for all reachable states (tr, th, σ) of P.

Reachability preservation (RP)

To establish reachability preservation, we prove that every step of P can be simulated by a corresponding step of $f(P)$. In particular, to simulate receive events, we show that intruder deducibility is preserved under abstractions f (see the second premise of rule $RECV$), i.e., for well-typed, R,Ax-normal, and ground substitutions σ, we have

$$T\sigma, IK_0 \vdash_E u\sigma \;\Rightarrow\; f(T)f(\sigma), IK_0' \vdash_E f(u)f(\sigma) \tag{4.1}$$

provided that $f(IK_0) \subseteq IK_0'$, where IK_0 and IK_0' respectively denote the intruder's initial knowledge associated with P and $f(P)$. This property is also required to show the preservation of attacks on secrecy as part of (AP). To prove (4.1), we first establish deducibility preservation for ground terms.

Theorem 1 (Deducibility preservation)**.** *Let $T \cup \{t\}$ be a set of ground terms such that $\mathcal{C} \subseteq T$ and T is R,Ax-normal. Then we have $T \vdash_E t$ implies $f(T) \vdash_E f(t{\downarrow_{R,Ax}})$.*

Second, we bridge the gap between the deducibility preservation for ground terms and (4.1) by establishing a substitution property which states that $f(t\theta) = f(t)f(\theta)$ for terms t and well-typed R,Ax-normal substitutions θ. However, as the following example shows, this property does not hold in general.

Example 18. Let $F_f = (f, E_f)$ be a typed abstraction such that E_f consists of the following equations with variables $X: \gamma_c$, $Y: \mathsf{msg}$ and some constant c:

$$\begin{aligned} f(\mathsf{h}(X)) &= f(X), \\ f(\mathsf{h}(Y)) &= \mathsf{h}(f(Y)). \end{aligned}$$

Let $t = \mathsf{h}(Z)$ and $\theta = \{c/Z\}$ where $Z: \mathsf{msg}$. Then we have

$$\begin{aligned} &f(t\theta) = f(\mathsf{h}(c)) = c, \\ &f(t)f(\theta) = \mathsf{h}(Z)\theta = \mathsf{h}(c). \end{aligned}$$

Hence, we have $f(t\theta) \neq f(t)f(\theta)$. ♠

The problem illustrated in this example is that the terms t and $t\theta$ are transformed by two distinct clauses. Thus, we must ensure that t and all its instance $t\theta$ are transformed uniformly, i.e., match the same clauses. We therefore require that the patterns in E_f must not overlap. This leads us to the notion of *pattern-disjointness.*

Given a list of equations L, we denote the set of its pattern types by $\Pi(L) = \{\Gamma(p) \mid (f(p) = u) \in L\}$ and define $\Pi_f = \Pi(E_f)$.

Definition 6. A function specification $F_f = (f, E_f)$ is *pattern-disjoint* if the types in Π_f are pairwise disjoint, i.e., for all $i, j \in \widetilde{n}$ such that $i \neq j$, we have $\Gamma(p_i){\downarrow} \cap \Gamma(p_j){\downarrow} = \emptyset$.

Intuitively, pattern-disjointness ensures that two patterns in E_f do not have common well-typed instances. Note that E_f^+ is not pattern-disjoint, because for each clause in E_f there is a corresponding one of a more general type in E_f^0. Therefore, it may happen that a term t is transformed by a clause in E_f^0, while its instance $t\theta$ is transformed by a clause in E_f. To exclude such cases, we need that all recursive calls of f on composed terms during the transformation of t are handled by the clauses of E_f, without recourse to the fall-back clauses in E_f^0. Note that it follows from Definition 5 that these composed terms are subterms of t. This motivates the introduction of the *uniform domain* of F_f.

Let V be a set of variables and $Rec(F_f, V, t)$ be the set of terms u such that $f(u)$ is called in the computation of $f(t)$ and $vars(u) \cap V \neq \emptyset$.

Definition 7 (Uniform domain)**.** We define the *uniform domain* of F_f with respect to a set of variables V by $udom(F_f, V) = \{t \in \mathcal{T} \mid \Gamma(Rec(F_f, V, t)) \subseteq \Pi_f{\downarrow} \cup \mathcal{Y}_{at}\}$.

When V is the set of all variables, we overload notation and simply write $udom(F_f)$. It is not hard to see that if F_f is pattern-disjoint then any term t belonging to the uniform domain of F_f with respect to $dom(\theta)$ and its instances $t\theta$ with well-typed R,Ax-normal substitutions θ are transformed uniformly. We henceforth assume that F_f is pattern-disjoint. Note that the abstraction defined in Example 9 is pattern-disjoint.

Theorem 2 (Substitution property). *Let θ be a well-typed and R,Ax-normal substitution and $t \in udom(F_f, dom(\theta))$. Then $f(t\theta) = f(t)f(\theta)$.*

A second natural ingredient that we need in our proof of property (4.1) is that abstractions preserve equality modulo E, i.e., if $t =_E u$ then $f(t) =_E f(u)$ where t, u are terms. We decompose this into preservation of Ax-equality and of rewrite steps, i.e., for all terms t and u, we have

(P1) If $t =_{Ax} u$ then $f(t) =_{Ax} f(u)$ for all terms t and u.

(P2) If $t \rightarrow_{R,Ax} u$ then $f(t) \rightarrow^*_{R,Ax} f(u)$ for all terms t.

Unfortunately, neither of these properties is preserved in general. The following example illustrates a violation of P1.

Example 19. Let na and nb be nonces and $F_f = (f, E_f)$ be a typed abstraction such that $E_f = [f(h(X)) = f(X)]$ with $X : \mathsf{exp}(\mathsf{msg}, \beta_{na})$. We consider two terms $t = \mathsf{h}(\mathsf{exp}(\mathsf{exp}(g, na), nb))$ and $u = \mathsf{h}(\mathsf{exp}(\mathsf{exp}(g, nb), na))$ and the additional axiom $\mathsf{exp}(\mathsf{exp}(g, X), Y) = \mathsf{exp}(\mathsf{exp}(g, Y), X)$. Then, we have

$$\begin{aligned} f(t) &= \mathsf{h}(\mathsf{exp}(\mathsf{exp}(g, na), nb)), \\ f(u) &= \mathsf{exp}(\mathsf{exp}(g, nb), na). \end{aligned}$$

It is clear that $t =_{Ax} u$, but $f(t) \neq_{Ax} f(u)$. The reason is that t and u are not transformed uniformly. In particular, t is transformed by a clause in E_f^0 which keeps t unchanged, while u is transformed by the clause in E_f which removes the hash function h. ♠

To solve the problem described in Example 19, we require that F_f is *Ax-closed*.

Definition 8 (Ax-closedness). F_f is *Ax-closed* if for all terms t, u and all $\tau \in \Pi(E_f^+)$, we have that $t =_{Ax} u$ implies $\Gamma(t) \preccurlyeq \tau$ if and only if $\Gamma(u) \preccurlyeq \tau$.

Intuitively, Ax-closedness ensures that whenever two R,Ax-normal composed terms are Ax-equivalent, each clause in E_f^+ either transforms both or none of them. This, together with the pattern-disjointness assumption, implies that these terms are transformed uniformly. In general, Ax-closedness is hard to check since it contains an universal quantification over terms. In Subsection 4.1.4, we will present a syntactic criterion for checking this condition. We henceforth assume that F_f is Ax-closed.

To achieve the preservation of rewrite steps under abstraction, we must ensure that, for all positions p in t where a rule $l \rightarrow r \in R$ is applicable, the redex $t|_p$ in t is transformed into a redex $f(t|_p)$ in $f(t)$ that still Ax-matches l. This requires that we restrict the interference of our abstractions with the rewrite rules as highlighted in the following example.

Example 20. Let $F_f = (f, E_f)$ be a typed abstraction such that

$$E_f = [f(\{|X|\}_Y) = f(X)]$$

with variables $X, Y : \mathsf{msg}$. Let $t = \{|\{|a|\}_k|\}_k^{-1}$ for some constants a and k. Then, we have $f(t) = \{|a|\}_k^{-1}$, $f(a) = a$, and $t \rightarrow_{R,Ax} a$. However, $f(t)$ is not reducible to a. Note that f does not preserve the left-hand side of the rewrite rule $\{|\{|X|\}_Y|\}_Y^{-1} \rightarrow X$ which was used in the rewrite step $t \rightarrow_{R,Ax} a$. ♠

To address the issue in Example 20, we must ensure that f keeps the left-hand side of any rewrite rule unchanged. This can be achieved by requiring that f is homomorphic for all function symbols in l. We now present a violation of P2 in the following example.

Example 21. Let us consider the case that $E_f = [f(\mathsf{h}(X)) = f(X)]$ with a variable $X : \{\!|\{\!|\gamma_a|\!\}_{\gamma_k}|\!\}^{-1}_{\gamma_k}$ for some constants a, k. Let $t = \mathsf{h}(\{\!|\{\!|a|\!\}_k|\!\}^{-1}_k)$ for some constant k. Then, we have $f(t) = \{\!|\{\!|a|\!\}_k|\!\}^{-1}_k$, $t \rightarrow_{R,Ax} \mathsf{h}(a)$, and $f(\mathsf{h}(a)) = \mathsf{h}(a)$. But we obtain $f(t) \rightarrow_{R,Ax} a \neq \mathsf{h}(a)$. Note that f transforms t and $\mathsf{h}(a)$ in two different ways using two different clauses, one is in E_f and the other is in E^0_f. ♠

To fix the problem in Example 21, it is sufficient to require that f uniformly transforms a protocol term t and the term u obtained from t by one-step rewriting. Observe that u is only different from t in the position p at which the redex is rewritten. By replacing the redex in t with a fresh message variable Z, we obtain a new term whose type is a supertype of both t's and u's. Hence, we can achieve the desired uniformity by ensuring that $t[Z]_p$ is in $udom(F_f, \{Z\})$. This motivates the introduction of *l-reducible terms.* For a term t, a set of terms $\{u_1, \ldots, u_n\}$, and a set of positions $\{p_1, \ldots, p_n\} \subseteq \mathcal{P}os(t)$ with $n \geq 1$, we use $t[t_1, \ldots, t_n]_{p_1,\ldots,p_n}$ to denote the term t_n where $t_0 = t$ and $t_i = t_{i-1}[u_i]_{p_i}$ for all $i \in \widetilde{n}$.

Definition 9. Let t be a non-variable term. We say that t is *l-reducible* if there is a rule $l \rightarrow r \in R$ such that one of the following holds:

(i) there is a well-typed Ax-unifier of t and l, or

(ii) there is a non-empty set of non-variable positions $\{p_1, \ldots, p_n\} \subseteq \mathcal{P}os(t) \setminus \{\epsilon\}$ such that

- p_i and p_j are incomparable with respect to $\sqsubseteq$ for all $i, j \in \widetilde{n}$ and $i \neq j$,
- $t|_{p_i}$ is l_i-reducible for all $i \in \widetilde{n}$, and
- there exists a well-typed Ax-unifier of $t[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n}$ and l for some fresh variables $Z_1, \ldots, Z_n$ such that

$$Z_i \notin vars(t) \cup \bigcup_{1 \leq j \leq n} vars(l_j) \cup vars(l) \text{ and } Z_i \colon \mathsf{msg} \text{ for all } i \in \widetilde{n}.$$

Intuitively, a non-variable term is l-reducible if there is some rewrite rule $l \rightarrow r \in R$ that is potentially applicable to an instance of t (by point (i)), or to a reduced instance of t (by point (ii)). Note that point (ii) takes care of the case that rewriting a term is possible only after some of its subterms are rewritten, but not before.

Example 22. We have that $Y \oplus Z$ is $X \oplus 0$-reducible because using the rewrite rule $X \oplus 0 \rightarrow X$ we obtain $(Y \oplus Z)\{0/Z\} = Y \oplus 0 \rightarrow Y$. Similarly, $Y \oplus (Y \oplus 0)$ is $X \oplus 0$-reducible because $Y \oplus 0$ is $X \oplus 0$-reducible and $Y \oplus Z$ is $X \oplus 0$-reducible where $Y \oplus 0$ is replaced by a fresh variable Z. Note that $Y \oplus Z$ and $Y \oplus (Y \oplus 0)$ are also $X \oplus X$-reducible where the rewrite rule $X \oplus X \rightarrow 0$ is considered instead. ♠

We now define a condition that prevents the issues in Examples 20 and 21.

Definition 10 (R-homomorphism)**.** We say that f is *R-homomorphic* for a term t if the following conditions hold:

(i) for all non-variable positions $p \in \mathcal{P}os(t)$ such that $t|_p$ is l-reducible, we have that f is homomorphic for $ct(l)$,

(ii) for all non-empty sets of non-variable positions $\{p_1, \dots, p_n\} \subseteq \mathcal{P}os(t)$ such that

- p_i and p_j are incomparable with respect to $\sqsubseteq$ for all $i, j \in \widetilde{n}$ and $i \neq j$, and
- $t|_{p_i}$ is l_i-reducible for all $i \in \widetilde{n}$,

we have that $t[Z_1, \dots, Z_n]_{p_1,\dots,p_n} \in udom(F_f, \{Z_1, \dots, Z_n\})$ for some fresh variables $Z_1, \dots, Z_n$ such that $Z_i \notin vars(t) \cup \bigcup_{1\leq j\leq n} vars(l_j)$ and $Z_i\colon \mathsf{msg}$ for all $i \in \widetilde{n}$.

We define $rdom(F_f)$ to be the set of terms for which f is R-homomorphic.

Intuitively, condition (i) addresses the issue described in Example 20. Condition (ii) excludes the counterexample 21. Note that R-homomorphism is checkable if there is an Ax-unification algorithm, i.e., for any terms t and u, it is possible to compute an Ax-unifiers of t and u if they are Ax-unifiable. Note that for the XOR theory in Example 5 and the Diffie-Hellman theory in Example 4, such unification algorithms exist (see [33]).

In practice, many interesting protocols P satisfy $\mathcal{M}_P \subseteq rdom(F_f)$, including those from our case studies. Since we must also cover redexes arising by instantiating protocol terms $t \in \mathcal{M}_P$, this definition employs Ax-unification rather than Ax-matching. The definition ensures that instantiations with R,Ax-normal substitutions and rewriting steps both preserve the membership of terms in $rdom(F_f)$.

Lemma 1. *Let θ be a well-typed R,Ax-normal ground substitution and t be a term such that $t \in rdom(F_f)$. Then, we have*

(i) $t\theta \in rdom(F_f)$,

(ii) t is ground and $t \rightarrow_{R,Ax} u$ imply $u \in rdom(F_f)$.

We now establish equality preservation for terms in $rdom(F_f)$.

Theorem 3 (Equality preservation)**.** *Let t and u be ground terms such that $t, u \in rdom(F_f)$. Then $t =_E u$ implies $f(t) =_E f(u)$.*

We can now derive (4.1) which is formalized in the following proposition.

Proposition 3 (Deducibility preservation for open terms)**.** *Let σ be a well-typed ground R,Ax-normal substitution, T be a set of terms, and u be term such that $vars(T \cup \{u\}) \subseteq dom(\sigma)$ and*

(i) $f(IK_0) \subseteq IK_0'$,

(ii) $T \cup \{u\} \subseteq udom(F_f) \cap rdom(F_f)$.

Then $T\sigma, IK_0 \vdash_E u\sigma$ *implies* $f(T)f(\sigma), IK_0' \vdash_E f(u)f(\sigma)$.

Proof. We present all the derivation steps as follows.

$$
\begin{array}{lllll}
 & T\sigma, IK_0 & \vdash_E & u\sigma & \text{by assumption} \\
\Rightarrow & (T\sigma)\downarrow_{R,Ax}, IK_0 & \vdash_E & (u\sigma)\downarrow_{R,Ax} & \text{by rule } \mathsf{Eq} \\
\Rightarrow & f((T\sigma)\downarrow_{R,Ax}), f(IK_0) & \vdash_E & f((u\sigma)\downarrow_{R,Ax}) & \text{by Theorem 1} \\
\Rightarrow & f((T\sigma)\downarrow_{R,Ax}), IK_0' & \vdash_E & f((u\sigma)\downarrow_{R,Ax}) & \text{by assumption (i)} \\
\Rightarrow & f(T\sigma), IK_0' & \vdash_E & f(u\sigma) & \text{by Lemma 1(i), Theorem 3 and } \mathsf{Eq} \\
\Rightarrow & f(T)f(\sigma), IK_0' & \vdash_E & f(u)f(\sigma) & \text{by Theorem 2}
\end{array}
$$

This completes the proof of the proposition. □

Summarizing, reachability preservation (RP) holds for $\mathcal{M}_P \subseteq \mathit{udom}(F_f) \cap \mathit{rdom}(F_f)$. We refer to Appendix A.1 for the full proofs of Theorems 1, 2, and 3.

Attack preservation (AP)

In this subsection, we explain the conditions on formulas needed to establish attack preservation. Given a formula ϕ, we define the following sets:

- Sec_ϕ be the set of all terms t that occur in formulas $\mathit{secret}(t)$ in ϕ,
- Eq_ϕ be the set of pairs (t, u) such that the equation $t = u$ occurs in ϕ and $\mathit{EqTerm}_\phi = \{t, u \mid (t, u) \in \mathit{Eq}_\phi\}$ is the set of underlying terms, and
- Evt_ϕ be the set of events occurring in ϕ.

Let Eq_ϕ^+ the positively occurring equations in ϕ and similarly for Evt_ϕ. In the following, we introduce *safe* formulas, i.e., those for which attack preservation holds.

Definition 11 (Safe formulas)**.** ϕ is *safe* for P and f if

(i) $\mathit{Sec}_\phi \cup \mathit{EqTerm}_\phi \subseteq \mathit{udom}(F_f) \cap \mathit{rdom}(F_f)$,

(ii) $f(t\sigma) =_E f(u\sigma)$ implies $t\sigma =_E u\sigma$ for all $(t, u) \in \mathit{Eq}_\phi^+$ and for all well-typed R,Ax-normal ground substitutions σ, and

(iii) $f(t) = f(u)$ implies $t = u$, for all $e(t) \in \mathit{Evt}_\phi^+$ and $e(u) \in \mathit{Evt}(\mathcal{M}_P)$.

Condition (i) requires that F_f is uniform and R-homomorphic for the terms in secrecy statements and equalities. Condition (ii) is required to preserve attacks on agreement properties. Concretely, it expresses injectivity of the abstraction on the terms in positively occurring equalities. In Section 4.1.4, we provide a syntactic criterion to check condition (ii) that avoids the universal quantification over substitutions. Condition (iii) is required for properties involving event orderings and *steps* predicates. It states that the abstraction must not identify an event occurring positively in the property with an original distinct protocol event. This is required to preserve event order.

We now state the soundness theorem.

Theorem 4 (Soundness)**.** *Suppose P, ϕ, and F_f satisfy the following conditions:*

- $f(IK_0) \subseteq IK_0'$,
- *F_f is Ax-closed and pattern-disjoint,*
- $\mathcal{M}_P \subseteq udom(F_f) \cap rdom(F_f)$, *and*
- *ϕ is safe for P and f.*

Then, for all states (tr, th, σ) reachable in P, we have

1. *$(f(tr), f(th), f(\sigma))$ is a reachable state of $f(P)$, and*
2. *$(tr, th, \sigma) \nvDash \phi$ implies $(f(tr), f(th), f(\sigma)) \nvDash f(\phi)$.*

The proof of this theorem can be found in Appendix A.5.

4.1.4. Syntactic Criteria for the Soundness of Typed Abstractions

Condition (ii) in the definition of safe formulas (Definition 11) which we call *condition $\mathcal{I}$* for the sake of convenience, and the Ax-closedness (Definition 8) are hard to check in practice. In particular, in order to justify Ax-closedness, we need to check whether for all R,Ax-normal composed ground terms t, u and all $\tau \in \Pi(E_f^+)$, we have that $t =_{Ax} u$ implies $\Gamma(t) \preccurlyeq \tau$ if and only if $\Gamma(u) \preccurlyeq \tau$. Condition $\mathcal{I}$ requires checking for given terms t, u, whether $f(t\sigma) =_E f(u\sigma)$ implies $t\sigma =_E u\sigma$ for all well-typed R,Ax-normal ground substitutions σ. Direct approaches towards answering these questions are impossible since the set of R,Ax-normal composed ground terms in the first question and the set of well-typed R,Ax-normal ground substitutions in the second question are infinite. We therefore propose checkable criteria for verifying these conditions. Our criterion for Ax-closedness can be checked in linear time (in the sizes of E_f^+ and Ax). For condition $\mathcal{I}$, we provide two syntactic criteria that can be applied depending on certain circumstances. More precisely, the first criterion can be used when there are no message variables occurring in t and u. This criterion can be checked in linear time (in the sizes of t, u, and E_f). The second criterion is more complex and checking it requires the existence of an Ax-unification algorithm.

Syntactic Criterion For Ax-closedness

In the following lemma, we provide a syntactic criterion to check if a function specification is Ax-closed. Intuitively, this lemma reduces Ax-closedness to a typing condition on the axioms.

Proposition 4. *Suppose for all types $\tau \in subs(\Pi(E_f^+))$ and all $\{s, t\} \in Ax$ such that $\tau{\downarrow} \cap \Gamma(s){\downarrow} \neq \emptyset$ implies $\Gamma(s) \preccurlyeq \tau$ and $\Gamma(t) \preccurlyeq \tau$. Then F_f is Ax-closed.*

Proof. In this proof, we assume a standard proof system for equational logic including rules for (instantiated) axioms (Axiom), reflexivity (Reflexivity), transitivity (Transitivity), and congruence (Congruence).

Let $\tau \in \Pi(E_f^+)$ and u, v be terms such that $u =_{Ax} v$. Because of symmetry, it is sufficient to show that $\Gamma(u) \preccurlyeq \tau$ implies $\Gamma(v) \preccurlyeq \tau$. We show this by induction on the derivation $u =_{Ax} v$ depending on the last rule that has been applied. Suppose that $\Gamma(u) \preccurlyeq \tau$.

- Reflexivity: In this case, we have $u = v$. Hence it is clear that $\Gamma(v) \preccurlyeq \tau$.
- Axiom: In this case, there is a pair $\{s, t\} \in Ax$ and a substitution σ such that $u = s\sigma$ and $v = t\sigma$. Since $\Gamma(u) \preccurlyeq \tau$, we have $\Gamma(s\sigma) \preccurlyeq \tau$. Moreover, we have $\Gamma(s\sigma) \preccurlyeq \Gamma(s)$. Hence, it holds that $\tau{\downarrow} \cap \Gamma(s){\downarrow} \neq \emptyset$. By assumption, we have $\Gamma(t) \preccurlyeq \tau$. We also have $\Gamma(t\sigma) \preccurlyeq \Gamma(t)$. Therefore, we obtain that $\Gamma(v) \preccurlyeq \tau$.
- Congruence: Suppose that $u = g(u_1, \ldots, u_n)$ for some $n \geq 1$ and $g \in \Sigma^n$. We have $v = g(v_1, \ldots, v_n)$ and $u_i =_{Ax} v_i$. Since $\Gamma(u) \preccurlyeq \tau$, either $\tau = \mathsf{msg}$ or $\tau = g(\tau_1, \ldots, \tau_n)$ and $\Gamma(u_i) \preccurlyeq \tau_i$ for all $i \in \widetilde{n}$. In the first case, i.e., $\tau = \mathsf{msg}$, it is obvious that $\Gamma(v) \preccurlyeq \tau$. In the latter case, by induction hypothesis, we know that $\Gamma(v_i) \preccurlyeq \tau_i$ for all $i \in \widetilde{n}$. This implies $\Gamma(v) \preccurlyeq \tau$ as required.
- Transitivity: In this case, there is a term w such that $t =_{Ax} w$ and $w =_{Ax} u$. By induction hypothesis, we have $\Gamma(w) \preccurlyeq \tau$ and $\Gamma(u) \preccurlyeq \tau$ which concludes this case.

This completes the proof of the proposition. $\square$

We demonstrate an application of Proposition 4 in the following example.

Example 23. Consider the function specification $F_1 = (f_1, E_1)$ in Example 17. We recall the list of equations E_1 with variables $X_3 : \gamma_o$, $Y_3 : \mathsf{Nonce}$, $U : \mathsf{kdf}(\mathsf{msg})$, and all remaining pattern variables are of type msg.

$$\begin{aligned}
f_1(\{\!|X, Y|\!\}_Z) &= \langle f_1(X), f_1(Y)\rangle \\
f_1(\mathsf{mac}(X_1, \ldots, X_8)) &= \mathsf{mac}(\widehat{f_1}([X_1, X_3, X_5, X_6, X_7, X_8])) \\
f_1(\mathsf{mac}(Y_1, \ldots, Y_8)) &= \mathsf{mac}(\widehat{f_1}([Y_1, Y_5, Y_6, Y_7, Y_8])) \\
f_1(\mathsf{kdf}(Z_1, \ldots, Z_5)) &= \mathsf{kdf}(f_1(Z_3)) \\
f_1(\mathsf{prf}(U, Z)) &= f_1(U) \\
f_1(\mathsf{exp}(U_1, U_2)) &= \mathsf{exp}(f_1(U_1), f_1(U_2)), \\
f_1(\mathsf{sh}(U_1, U_2)) &= \mathsf{sh}(f_1(U_1), f_1(U_2)), \\
f_1(\langle U_3, U_4\rangle) &= \langle f_1(U_3), f_1(U_4)\rangle,
\end{aligned}$$

There are only two equations in Ax_{cs}, namely $\mathsf{exp}(\mathsf{exp}(g, X), Y) \simeq \mathsf{exp}(\mathsf{exp}(g, Y), X)$ and $\mathsf{sh}(X, Y) \simeq \mathsf{sh}(Y, X)$ where $X, Y : \mathsf{msg}$. Let $\tau \in subs(\Pi(E_f))$. For both axioms $\{s, t\} \in Ax_{cs}$, it is not hard to see that $\tau{\downarrow} \cap \Gamma(s){\downarrow} \neq \emptyset$ implies that $\tau = \mathsf{msg}$ or $\tau = \Gamma(s) = \Gamma(t)$. Hence, it holds that $\Gamma(s) \preccurlyeq \tau$ and $\Gamma(t) \preccurlyeq \tau$. Thus we conclude that F_1 is Ax-closed. ♠

Criterion for Condition $\mathcal{I}$

In this subsection, we present a syntactic criterion to justify the satisfaction of condition $\mathcal{I}$. More generally, we want to solve the following problem.

Problem 1. *Suppose F_f is Ax-closed and pattern-disjoint. Let $t, u \in \mathcal{M}^\sharp$ be terms and $t, u \in udom(F_f)$. Under which conditions it holds that $f(t\sigma) =_E f(u\sigma)$ implies that $t\sigma =_E u\sigma$ for all R,Ax-normal well-typed ground substitutions σ?*

We assume arbitrary but fixed terms t and u such that $t, u \in udom(F_f)$ and an R,Ax-normal well-typed ground substitution σ. We intend to look for sufficient conditions under which $f(t\sigma) =_E f(u\sigma)$ implies $t\sigma =_E u\sigma$. We also require that these conditions do not depend on σ. This is to ensure that our conditions work for all such substitutions σ.

In the following paragraphs, we present criteria that can be applied when either no message variables are present in t and u or message variables occur only in t or in u. The hardest case is when message variables are present in both t and u, for which we currently do not have a good criterion.

Criterion I Note that by the substitution property, we can derive the following equalities:

$$\begin{array}{rcl} f(t\sigma) & = & f(t)f(\sigma), \\ f(u\sigma) & = & f(u)f(\sigma). \end{array}$$

Hence, the equality $t\sigma =_E u\sigma$ immediately follows from $f(t)f(\sigma) =_E f(u)f(\sigma)$ if it holds that $f(t) = t$, $f(u) = u$, and $Xf(\sigma) = X\sigma$ for all $X \in vars(t) \cup vars(u)$. Clearly, we have that $f(X\sigma) = X\sigma$ for all non-message variables $X \in dom(\sigma)$. Therefore, in the case that t and u do not contain message variables, the following syntactic criterion is straightforward to prove.

Proposition 5. *Let $t, u \in \mathcal{M}^\sharp$ be terms such that*

(i) $\mathsf{msg} \notin \Gamma(vars(t) \cup vars(u))$,

(ii) $t, u \in udom(F_f)$, *and*

(iii) $f(t) = t$ *and* $f(u) = u$.

Then for all well-typed ground substitutions σ, we have that $f(t\sigma) =_E f(u\sigma)$ implies $t\sigma =_E u\sigma$.

Proof. Without loss of generality, we can assume that $dom(\sigma) \subseteq vars(t) \cup vars(u)$. Since $\mathsf{msg} \notin \Gamma(vars(t) \cup vars(u))$, we derive that $X\sigma$ is an atom for all $X \in dom(\sigma)$. This implies $f(\sigma) = \sigma$. Moreover, by Theorem 2, we have $f(t\sigma) = f(t)f(\sigma)$ and $f(u\sigma) = f(u)f(\sigma)$. From assumption (iii), we derive

$$\begin{array}{rcl} f(t\sigma) & = & t\sigma \\ f(u\sigma) & = & u\sigma \end{array}$$

Since $f(t\sigma) =_E f(u\sigma)$, we obtain that $t\sigma =_E u\sigma$ as required. This completes the proof of the proposition. $\square$

Note that for criterion I to be applicable, we require that f is the identity for the terms in equalities. This is often the case, as these terms typically have simple structures and therefore it is reasonable to keep them unchanged.

Criterion II In this paragraph, we consider the case where t or u contain message variables. Without loss of generality, we assume that message variables occur only in t. Using the same approach as above, we assume that $f(t) = t$ and $f(u) = u$. We want to show that

$$f(\sigma)|_{vars(t)\cup vars(u)} = \sigma|_{vars(t)\cup vars(u)} \; . \tag{4.2}$$

Equivalently, we need to show that $Xf(\sigma) = X\sigma$ for every variable $X \in vars(t) \cup vars(u)$. This holds trivially for $X \in vars(u)$ as shown above, and for all non-message variables in t. To prove that $Xf(\sigma) = X\sigma$ for each message variable $X \in vars(t)$, we need to ensure that $f(\sigma)$ is R,Ax-normal. Otherwise, equality (4.2) and the injectivity property of Problem 1 may fail to hold as illustrated in the following example.

Example 24. We consider the case that $t = X$ is a message variable, $u = a$ is an atom and E_f contains the following equations:

$$\begin{array}{rcl} f(\{\!|X_1|\!\}_{Y_1}) & = & \{\!|f(X_1)|\!\}_{f(Y_1)} \\ f(\{\!|X_2|\!\}^{-1}_{Y_2}) & = & \{\!|f(X_2)|\!\}^{-1}_{f(Y_2)} \\ f(\mathsf{h}(X_2)) & = & f(X_2) \end{array}$$

where all pattern variables are of type msg. Intuitively, f is homomorphic for $\{\!|\cdot|\!\}_{\cdot}$ and $\{\!|\cdot|\!\}^{-1}_{\cdot}$ and removes the hash function symbol h from terms. We define the substitution $\sigma = \{\{\!|\{\!|a|\!\}_{\mathsf{h}(a)}|\!\}^{-1}_a/X\}$. Then we have $tf(\sigma) =_E uf(\sigma)$ because $uf(\sigma) = a$ and

$$tf(\sigma) = f(X\sigma) = f(\{\!|\{\!|a|\!\}_{\mathsf{h}(a)}|\!\}^{-1}_a) = \{\!|\{\!|a|\!\}_a|\!\}^{-1}_a =_E a.$$

But $X\sigma = \{\!|\{\!|a|\!\}_{\mathsf{h}(a)}|\!\}^{-1}_a \neq_E a = u\sigma$. ♠

This example also highlights the difficulty of achieving R, Ax-normality for $f(\sigma)$ without substantial restrictions. We therefore take the following approach. We show that for each R,Ax-normal attack σ on a property ϕ in P, there is an R,Ax-normal attack σ' on ϕ such that $f(\sigma')$ is R,Ax-normal. We construct σ' from σ by replacing a subterm v of the terms in $ran(\sigma)$ such that $f(v)$ is not R,Ax-normal with a new constant a. By applying this replacement exhaustively, we eliminate all subterms in the range of σ whose corresponding abstracted terms are redexes. The resulting substitution σ' satisfies that $f(\sigma')$ is R,Ax-normal.

We define the set of rewrite rules $\mathsf{R5} \subseteq \mathsf{R3} \cup \mathsf{R4}$ such that $l \to r \in \mathsf{R5}$ if and only if $l \to r \in R$ and l is of the form $c(u_1, \dots, u_n)$ for some $c \in \Sigma^n$ and $u_1, \dots, u_n$ are variables or constants and pairwise distinct. We overload notation and define the following sets:

$$\begin{array}{rcl} subs(Ax) & = & \bigcup_{\{s,t\}\in Ax} subs(s) \cup subs(t), \\ ct(Ax) & = & \bigcup_{\{s,t\}\in Ax} ct(s) \cup ct(t), \\ subs(R) & = & \bigcup_{l\to r\in R} subs(l) \cup subs(r), \\ lhs(R) & = & \{l \mid l \to r \in R\}. \end{array}$$

For terms v and v', we use $[v'/v]_{\simeq_{Ax}}$ to denote the mapping that replaces each term t' such that $t' =_{Ax} v$ by v'. We also define the domain of $[v'/v]_{\simeq_{Ax}}$ by

$$dom([v'/v]_{\simeq_{Ax}}) = \{t \in \mathcal{T} \mid t =_{Ax} v\}.$$

The following proposition allows us to treat $[v'/v]_{\simeq_{Ax}}$ as a replacement.

Proposition 6. *Suppose that for all $\{s, s'\} \in Ax$, we have that $|s| = |s'|$ and s, s' are linear. Then for all terms t and u such that $t =_{Ax} u$, it holds that u is not a strict subterm of t.*

Proof. Let $\{s, s'\} \in Ax$ and σ be a substitution. For a given set of variables V, we define $[\![\sigma]\!]_V = \sum_{X \in V \cap dom(\sigma)} |X\sigma|$. By assumption, we derive that

$$\begin{array}{rcl} |s\sigma| & = & |s| - |dom(\sigma) \cap vars(s)| + [\![\sigma]\!]_{vars(s)}, \\ |s'\sigma| & = & |s'| - |dom(\sigma) \cap vars(s')| + [\![\sigma]\!]_{vars(s')}. \end{array}$$

Since $vars(s) = vars(s')$ and $|s| = |s'|$, we have $|s\sigma| = |s'\sigma|$. This together with $t =_{Ax} u$ implies that $|t| = |u|$. Hence u cannot be a strict subterm of t. □

For the remainder of this paragraph, we assume that $|s| = |s'|$ and s, s' are linear for all pairs $\{s, s'\} \in Ax$. Given a substitution σ, we denote by $\sigma[v'/v]_{\simeq_{Ax}}$ the substitution such that

$$\begin{array}{rcl} dom(\sigma[v'/v]_{\simeq_{Ax}}) & = & dom(\sigma), \\ \sigma[v'/v]_{\simeq_{Ax}}(X) & = & (X\sigma)[v'/v]_{\simeq_{Ax}}. \end{array}$$

In the next lemma, we present a deducibility preservation result for ground terms, which will later be used to show reachability and attack preservations for each replacement step. Let us first introduce the following definition.

Definition 12 (R,Ax-stable terms)**.** We say that a term t is *R,Ax-stable* if $t\sigma$ is R,Ax-normal whenever σ is R,Ax-normal and well-typed. A set of terms is R,Ax-stable if all its elements are.

Lemma 2. *Let $T \cup \{t, v\}$ be a set of ground terms such that v is R,Ax-normal and let a be an atom such that*

(i) $a \notin subs(t) \cup subs(Ax) \cup subs(R)$,

(ii) $top(v) \notin ct(Ax)$, and

(iii) $top(v) \neq top(l|_p)$ for all $l \in lhs(R)$ and non-variable positions $p \in \mathcal{P}os(l) \setminus \{\epsilon\}$, and

Then $T \vdash_E t$ implies $T[a/v]_{\simeq_{Ax}}, a \vdash_E t[a/v]_{\simeq_{Ax}}$.

Note that conditions (i) and (ii) are required to prevent the interference of the replacement with rewrite rules or axioms. Condition (iii) ensures that applying the replacement to an instance of the left-hand side l of a rewrite rule can only be done in the terms substituted for the variables in l.

Using Lemma 2, we now prove deducibility preservation for open terms.

Proposition 7. *Let $T \cup \{t, v, a\}$ be terms such that v is ground and R,Ax-normal and a is a constant. Let σ be a ground R,Ax-normal well-typed substitution. Suppose that the following holds.*

(i) $a \notin subs(t) \cup subs(v) \cup subs(Ax) \cup subs(R) \cup subs(ran(\sigma))$,

(ii) $top(v) \notin ct(Ax)$,

(iii) $top(v) \neq top(l|_p)$ *for all* $l \in lhs(R)$ *and non-variable positions* $p \in \mathcal{P}os(l) \setminus \{\epsilon\}$,

(iv) there is no $u \in subs(T) \cup subs(t)$ *such that* $u\sigma =_{Ax} v$, *and*

(v) $top(v) \notin ct(IK_0)$.

Then $T\sigma, IK_0 \vdash_E t\sigma$ *implies* $T(\sigma[a/v]_{\simeq_{Ax}}), IK_0 \vdash_E t(\sigma[a/v]_{\simeq_{Ax}})$.

Proof. Suppose that $T\sigma, IK_0 \vdash_E t\sigma$. By assumption (i), we derive that $a \notin subs(t\sigma)$. Hence, by Lemma 2, we have

$$(T\sigma)[a/v]_{\simeq_{Ax}}, IK_0[a/v]_{\simeq_{Ax}}, a \vdash_E (t\sigma)[a/v]_{\simeq_{Ax}}.$$

By assumption (iv), we derive that

$$\begin{array}{rcl}(T\sigma)[a/v]_{\simeq_{Ax}} & = & T(\sigma[a/v]_{\simeq_{Ax}}), \\ (t\sigma)[a/v]_{\simeq_{Ax}} & = & t(\sigma[a/v]_{\simeq_{Ax}}).\end{array}$$

Moreover, by assumption (v), we derive that $IK_0 = IK_0[a/v]_{\simeq_{Ax}}$. Together with the fact that $a \in IK_0$, we obtain $T(\sigma[a/v]_{\simeq_{Ax}}), IK_0 \vdash_E t(\sigma[a/v]_{\simeq_{Ax}})$. This completes the proof of the proposition. □

To ensure that our replacement procedure is terminating, we must show that each replacement step does not create additional redexes. This is achieved by the following lemma.

Lemma 3. *Let t, v be ground terms and let a be an atom. Assume that*

(i) t *is* R,Ax*-normal,*

(ii) v *is composed and not a pair,*

(iii) for all $\tau \in \Pi(E_f^+)$ *and all types* $\tau' \in subs(\tau) \setminus \{\tau\}$, *we have*
   - *(a)* $top(\tau') \neq top(v)$,
   - *(b) if* $\gamma_a \preccurlyeq \tau'$ *then* $\tau' = \mathsf{msg}$.

(iv) for all terms $u \in Rec(F_f, t)$, $u \neq_{Ax} v$,

(v) $a \notin subs(Ax) \cup subs(R)$,

(vi) $top(v) \notin ct(Ax)$.

Then $t[a/v]_{\simeq_{Ax}}$ *is R,Ax-normal and* $f(t) = f(t[a/v]_{\simeq_{Ax}})$.

Intuitively, Lemma 3 ensures that the replacement $[a/v]_{\simeq_{Ax}}$ preserves R, Ax.-normality and abstraction under f. In this lemma, condition (ii) ensures that a replacement is done only in fields. Conditions (ii) and (iii) ensures that t and $t[a/v]_{\simeq_{Ax}}$ are transformed by the same clause. Condition (iv) forbids the interference of the replacement with the computation of abstraction. Likewise, conditions (v) and (vi) do not allow the interference of the replacement with the axioms.

We now establish our attack simulation result. First, we introduce some auxiliary definitions.

Definition 13 (Composite-preserving function specifications)**.** Let $F_f = (f, E_f)$ be a function specification. We say that f is *composite-preserving* if for all clauses $(f(p) = u) \in E_f$ such that $top(p) \in \Sigma^n$ with $n \geq 1$, we have either (i) $top(q) \in \Sigma^m \setminus \{f\}$ with $m \geq 1$, or (ii) $q = f(q')$ for some term q' such that $\Gamma(q')$ is composed.

Intuitively, f is composite-preserving if it cannot produce a non-composed term from a composed one.

We now introduce the notion of *Ax-stability* which is stronger than R,Ax-stability.

Definition 14 (Ax-stable terms)**.** We say that a term t is *Ax-stable* if no non-variable subterm of t is unifiable modulo Ax with an axiom term, i.e., for all equations $\{s, s'\} \in Ax$ and all non-variable subterms $t|_p$, there is no well-typed substitution σ such that $s\sigma = t|_p\, \sigma$.

In order to establish our attack simulation result, we require that the rewrite theory satisfies certain conditions which are formulated in the following definition.

Definition 15 (Top-restricted rewrite theories)**.** We say that a rewrite theory (Σ, Ax, R) is *top-restricted* if the following conditions hold:

(i) $ct(IK_0) \cap top(lhs(R)) = \emptyset$,

(ii) for all $l \in lhs(R)$, we have $top(l) \notin ct(Ax)$ or l is Ax-stable and $l \in lhs(\mathsf{R5})$, and

(iii) for all $l \in lhs(R)$ and all non-variable positions $p \in \mathcal{P}os(l) \setminus \{\epsilon\}$, we have that $top(l|_p) \notin top(lhs(R))$.

Intuitively, condition (i) ensures that reducible function symbols do not occur in IK_0. Condition (ii) restricts the interference of rewrite rules with the axioms. Finally, condition (iii) forbid nested occurrences of reducible function symbols in the left-hand sides of the corresponding rewrite rules.

Next, we define a compatibility condition for our typed abstractions, protocols, and security properties.

Definition 16 ((F_f, P, ϕ)-compatibility)**.** Let $F_f = (f, E_f)$ be a function specification and $\phi \in \mathcal{L}_P$. We say that (F_f, P, ϕ) is *compatible with a rewrite theory* $\mathcal{R} = (\Sigma, Ax, R)$ if the following holds.

(i) f is composite-preserving and homomorphic for $top(lhs(R))$,

(ii) for all $\tau \in \Pi(E_f^+)$ and all types $\tau' \in subs(\tau) \setminus \{\tau\}$, we have

   (a) $top(\tau') \notin top(lhs(R))$,

   (b) for all constants $c \notin subs(\mathcal{M}_P)$, if $\gamma_c \preccurlyeq \tau'$ then $\tau' = \mathsf{msg}$,

(iii) for all non-variable terms $v \in subs(\mathcal{M}_P \cup Sec_\phi \cup EqTerm_\phi)$ and all $l \in lhs(R)$, there is no Ax-unifier of $f(v)$ and l, and

(iv) $subs(\mathcal{M}_P \cup Sec_\phi \cup EqTerm_\phi) \subseteq udom(F_f)$.

In condition (i), the first conjunct allows us to derive the shape of a term whose abstraction is an atom. The second conjunct forbids non-trivial abstractions for reducible function symbols. Conditions (ii)-(iv) are needed to enable the use of Proposition 7 and Lemma 3.

Our attack simulation result is now achieved by the following lemma.

Lemma 4. *Let $\phi \in \mathcal{L}_P$. Suppose that*

(i) (Σ, Ax, R) *is top-restricted,*

(ii) (F_f, P, ϕ) *is compatible with* (Σ, Ax, R).

Then for all reachable states (tr, th, σ) of P, there is an R,Ax-normal well-typed ground substitution σ' such that $f(\sigma')$ is R,Ax-normal and the following holds:

- *(tr, th, σ') is a reachable state in P, and*
- *if $(tr, th, \sigma) \not\models \phi$ then $(tr, th, \sigma') \not\models \phi$.*

Thanks to Lemma 4, it is sufficient to consider Problem 1 under the assumptions (i) and (ii) from Lemma 4 and the assumption that $f(\sigma)$ is R,Ax-normal. Now, provided that these assumptions hold, we attempt to establish (4.2). In order to show this, we need to restrict the shape of the terms in $ran(f(\sigma))$. This requires us to stabilize both $tf(\sigma)$ and $uf(\sigma)$. We therefore prevent rewrite rules and axioms from being applicable to these terms. To ensure that $tf(\sigma)$ and $uf(\sigma)$ are both R,Ax-normal and Ax-stable, we require the following conditions:

(a) t and u are R,Ax-stable,

(b) u is Ax-stable.

By condition (a), the equality $tf(\sigma) =_E uf(\sigma)$ is equivalent to $tf(\sigma) =_{Ax} uf(\sigma)$. This, together with (b), further implies that $tf(\sigma) =_{Ax} uf(\sigma)$ is equivalent to $tf(\sigma) = uf(\sigma)$.

The proofs of lemmas 2, 3, and 4 can be found in Appendix B. It remains to establish $Xf(\sigma) = X\sigma$ for all $X \in vars(t)$ and $X : \mathsf{msg}$. Essentially, this requires to show that

$$f(v) = v' \text{ implies } v = v' \tag{4.3}$$

for $v = X\sigma$ and some term v'. Clearly, $v = v'$ implies $top(v) = top(v')$. Therefore, we at least must be able to show that only terms with the same top-level constructor as v's can be mapped to v under f. This leads us to the notion of *constructor-exclusiveness.*

Definition 17 (Constructor-exclusiveness)**.** We say that f is *constructor-exclusive* for a function symbol $c \in \Sigma^{\geq 1}$ if for all $(f(p) = q) \in E_f$ we have $top(q) = c$ implies $top(p) = c$.

In the following lemma, we prove that every ground abstracted term with top-level constructor c can only be obtained by transforming a term of the same form provided that f is composite-preserving and constructor-exclusive for c.

Lemma 5. *Let t be a ground term and $c \in \Sigma^{\geq 1}$ be a function symbol. Suppose that*

(i) f is composite-preserving,

(ii) f constructor-exclusive for c.

Then $top(f(t)) = c$ implies $top(t) = c$.

Proof. Suppose that $top(f(t)) = c$. It follows that t is composed. Then there exists the first pattern $(f(p) = q) \in E_f^+$ such that $\Gamma(t) \preccurlyeq \Gamma(p)$ and $p\theta = t$ for some substitution θ. We distinguish two cases:

- If $(f(p) = q) \in E_f^0$ then we have that $top(q) = top(p) = top(t) = g'$. We also have $c = top(f(t)) = top(q\theta) = top(t)$. This yields $top(t) = c$ as required.
- If $(f(p) = q) \in E_f$ then we have

$$\begin{array}{rcl} top(t) & = & top(p), \\ top(f(t)) & = & top(q). \end{array}$$

  By assumption (i), $f(t)$ is composed, and assumption (ii), we derive that $top(t) = top(p) = top(q) = c$.

This completes the proof of the lemma. $\square$

In the following lemma, we show that (4.3) holds under some conditions.

Lemma 6. *Let t, u be ground terms and suppose that*

(i) f is composite-preserving,

(ii) f is constructor-exclusive for all $c \in ct(u)$, and

(iii) f is homomorphic for $ct(u)$.

Then $f(t) = u$ implies $t = u$.

Proof. We prove this lemma by induction on u.

- If u is an atom then by $f(t) = u$ and assumption (i), we derive that t must be an atom. Hence, we have $f(t) = t$ and thus obtain $t = u$ as required.

- If $u = c(u_1, \ldots, u_n)$ for $c \in \Sigma^n$ and some terms $u_1, \ldots, u_n$, then since $f(t) = u$, we have $top(f(t)) = c$. By Lemma 5, we derive that there are terms $t_1, \ldots, t_n$ such that $t = c(t_1, \ldots, t_n)$. Moreover, by assumption (iii), we have $f(t) = c(f(t_1), \ldots, f(t_n))$. Since $f(t) = u$, we obtain that $f(t_i) = u_i$ for all $i \in \widetilde{n}$. By induction hypothesis, we have that $t_i = u_i$ for all $i \in \widetilde{n}$. Therefore, we derive that $t = u$ as required.

This completes the proof of the lemma. $\square$

We now establish (4.2) in the following lemma.

Lemma 7. *Let t, u be terms such that $\mathsf{msg} \notin \Gamma(vars(u))$ and σ be a R,Ax-normal well-typed ground substitution such that $tf(\sigma) = uf(\sigma)$. Assume that the following holds:*

(i) f is composite-preserving,

(ii) for all positions $p \in \mathcal{P}os(t) \cap \mathcal{P}os(u)$ such that $t|_p$ is a message variable, we have

a) f is constructor-exclusive for all $c \in ct(u|_p)$, and

b) f is homomorphic for $ct(u|_p)$.

Then we have $f(\sigma)|_{vars(t)} = \sigma|_{vars(t)}$.

Proof. It is sufficient to show that $Xf(\sigma) = X\sigma$ for all $X \in vars(t)$ such that $X : \mathsf{msg}$. Let X be a message variable at position p in t. Since $tf(\sigma) = uf(\sigma)$, it is clear that $p \in \mathcal{P}os(u)$. Moreover, we have $Xf(\sigma) = u|_p\, f(\sigma)$. Note that $Xf(\sigma) = f(X\sigma)$. Hence, by Lemma 6, we derive that $X\sigma = u|_p\, f(\sigma)$. This implies $Xf(\sigma) = X\sigma$ as required. $\square$

Finally, we are in a position to state our criterion.

Proposition 8. *Let $\phi \in \mathcal{L}_P$ be a property formula, $(t, u) \in Eq_\phi$ such that $\mathsf{msg} \notin \Gamma(vars(u))$, and (tr, th, σ) be a reachable state of P such that σ is R,Ax-normal and $(tr, th, \sigma) \nvDash \phi$. Assume that the following holds:*

(i) (Σ, Ax, R) is compatible,

(ii) (F_f, P, ϕ) is compatible with (Σ, Ax, R),

(iii) $f(t) = t$ and $f(u) = u$,

(iv) t and u are R,Ax-stable and u is Ax-stable,

(v) for all positions $p \in \mathcal{P}os(t) \cap \mathcal{P}os(u)$ such that $t|_p$ is a message variable, we have

a) f is constructor-exclusive for all $c \in ct(u|_p)$,

b) f is homomorphic for $ct(u|_p)$,

Then there is an R,Ax-normal ground substitution σ' such that $(tr, th, \sigma') \nvDash \phi$ and $f(\sigma')$ is R,Ax-normal. Furthermore, if $f(t\sigma') =_E f(u\sigma')$ then $t\sigma' =_E u\sigma'$.

Proof. By Lemma 4, we know that there is a ground R,Ax-normal substitution σ' such that $(tr, th, \sigma') \not\models \phi$ and $f(\sigma')$ is R,Ax-normal. Suppose that $f(t\sigma') =_E f(u\sigma')$. We need to show that $t\sigma' =_E u\sigma'$. By Definition 16(iv), we apply Theorem 2 and derive that

$$\begin{array}{rcl} f(t\sigma') & = & f(t)f(\sigma'), \\ f(u\sigma') & = & f(u)f(\sigma'). \end{array}$$

By assumption (iii), we derive that $tf(\sigma') =_E uf(\sigma')$. Therefore, we obtain that

$$(tf(\sigma'))\downarrow_{R,Ax} =_{Ax} (uf(\sigma'))\downarrow_{R,Ax} .$$

This by assumption (iv) and the R, Ax-normality of $f(\sigma')$ implies $tf(\sigma') =_{Ax} uf(\sigma')$. By (iv), we know that $uf(\sigma')$ is Ax-stable. Therefore, we derive that $tf(\sigma') = uf(\sigma')$. By Definition 16(i), assumption (v), and Lemma 7, we have $f(\sigma')|_{vars(t)} = \sigma'|_{vars(t)}$. Note that since $\mathsf{msg} \notin \Gamma(vars(u))$, we also have that $f(\sigma')|_{vars(u)} = \sigma'|_{vars(u)}$. Hence, we obtain $t\sigma' = u\sigma'$ which yields $t\sigma' =_E u\sigma'$. □

In Proposition 8, checking R,Ax-stability and Ax-stability (conditions (iii) in Definition 16 and (i) and (iv) in Proposition 8) requires that an Ax-unification algorithm exists. The other conditions can be effectively checked. Note that the Diffie-Hellman theory in Example 4 is top-restricted and therefore satisfies condition (i) in Proposition 8. Unfortunately, the XOR theory in Example 5 is not top-restricted. Weaker conditions are therefore desirable to support such a theory.

We now apply our criterion to justify condition $\mathcal{I}$ with respect to protocol $\mathsf{IKE_m}$ and the typed abstraction specified in Example 17, and the property ϕ_a formalized in Example 8.

Example 25. First, we recall the typed abstraction $F_1 = (f_1, E_1)$ in Example 17, where E_1 is defined by the equations:

$$\begin{array}{rcl} f_1(\{|X, Y|\}_Z) & = & \langle f_1(X), f_1(Y)\rangle \\ f_1(\mathsf{mac}(X_1, \dots, X_8)) & = & \mathsf{mac}(\widehat{f_1}([X_1, X_3, X_5, X_6, X_7, X_8])) \\ f_1(\mathsf{mac}(Y_1, \dots, Y_8)) & = & \mathsf{mac}(\widehat{f_1}([Y_1, Y_5, Y_6, Y_7, Y_8])) \\ f_1(\mathsf{kdf}(Z_1, \dots, Z_5)) & = & \mathsf{kdf}(f_1(Z_3)) \\ f_1(\mathsf{prf}(U, Z)) & = & f_1(U) \\ f_1(\mathsf{exp}(U_1, U_2)) & = & \mathsf{exp}(f_1(U_1), f_1(U_2)), \\ f_1(\mathsf{sh}(U_1, U_2)) & = & \mathsf{sh}(f_1(U_1), f_1(U_2)), \\ f_1(\{|V_1, V_2|\}_Z) & = & \langle f_1(V_1), f_1(V_2)\rangle \\ f_1(\langle U_3, U_4\rangle) & = & \langle f_1(U_3), f_1(U_4)\rangle, \end{array}$$

and $V_1 : \alpha$, $X_3 : \gamma_o$, $Y_3 : \mathsf{Nonce}$, $U : \mathsf{kdf}(\mathsf{msg})$, and all remaining pattern variables are of type msg. We also recall the property ϕ_a as below.

$$\begin{array}{rcl} \phi_a & = & \forall j.\ (role(j, A) \wedge honest(j, [A, B]) \wedge steps(j, \mathsf{Commit})) \\ & & \Rightarrow (\exists k.\ role(k, B) \wedge steps(k, \mathsf{Running}) \wedge \\ & & \quad \langle A^j, B^j, na^j, Nb^j, \mathsf{exp}(g, x^j), Gb^j\rangle = \langle A^k, B^k, Na^k, nb^k, Ga^k, \mathsf{exp}(g, y^k)\rangle)). \end{array}$$

Second, we verify Condition $\mathcal{I}$ for the equations in ϕ_a. Note that the equation

$$\langle A^j, B^j, na^j, Nb^j, \mathsf{exp}(g, x^j), Gb^j\rangle = \langle A^k, B^k, Na^k, nb^k, Ga^k, \mathsf{exp}(g, y^k)\rangle)$$

can be decomposed into smaller ones, i.e.,

$$A^j = A^k;\ B^j = B^k;\ na^j = Na^k;\ Nb^j = nb^k;\ Ga^k = \mathsf{exp}(g, x^j);\ Gb^j = \mathsf{exp}(g, y^k).$$

The first four equations do not contain message variables. It is not hard to see that Condition $\mathcal{I}$ holds for these equations by Proposition 5. We now justify Condition $\mathcal{I}$ for the fifth equation by checking conditions in Proposition 8. The last equation can be treated in a similar way. We assume that IK_0 does not contain function symbols from $top(lhs(R))$. Then condition (i) holds for the considered equational theory. We now check conditions (ii)-(v) in Definition 16. Note that conditions (ii), (iii), and (v) immediately follow from the specification of E_f. Condition (iv) is satisfied, since no term in the set $subs(\mathcal{M}_{\mathsf{IKE_m}} \cup Sec_\phi \cup EqTerm_\phi)$ contains a function symbol from $top(lhs(R))$ and f is constructor-preserving. Hence condition (i) in Proposition 8 holds. It remains to check conditions (iii)-(v) in Proposition 8. Since $u = \mathsf{exp}(g, x^j)$ and $t = Ga^k$, where $j, k \in TID$ and x is a nonce, we have $f_1(t) = t$ and $f_1(u) = u$. Therefore, condition (iii) is satisfied. Since t and u are R,Ax-stable and u is Ax-stable, condition (iv) also holds. Moreover, note that $t|_p$ is a variable if and only if $p = \epsilon$. Since $ct(u|_\epsilon) = ct(u) = \{\mathsf{exp}\}$. It is not hard to see that f is constructor-exclusive and homomorphic for exp. Therefore, condition (v) holds. ♠

In practice, syntactic Criteria I (Proposition 5) and II (Proposition 8) are sufficient for many relevant verification problems, e.g., all case studies in this thesis including those in Chapter 6 can be justified using these criteria. For authentication properties that involve agreement on atoms or variables of simple types such as nonces or timestamps, Criterion I is applicable. Compared to I, Criterion II has a larger scope and can be applied for authentication properties that involves message variables. The complexity of checking criterion II mostly contributes to computing Ax-unifiers. Nevertheless, an Ax-unification algorithm required for this criterion needs not compute a complete set of most general Ax-unifiers. Computing an Ax-unifier of two given terms in the case that such a unifier exists is sufficient.

4.2. Untyped Abstractions

Typed abstractions offer a wide range of possibilities to transform cryptographic operations including subterm removal, splitting, and pulling fields outside a cryptographic operation. However, these abstractions cannot remove atoms/variables or redundant terms. We therefore complement typed abstractions with two kinds of *untyped abstractions* that allow us to remove:

(1) unprotected atoms and variables of any type and

(2) redundancy in the form of intruder-derivable terms.

Untyped protocol abstractions are functions $g : \mathcal{T} \to \mathcal{T} \cup \{\mathsf{nil}\}$ where messages to be removed are mapped to nil. We remove events with nil arguments from the roles.

4.2.1. Atom-and-variable Removal Abstractions

Atom-and-variable removal abstractions allow us to eliminate non-composed terms u from a term t if u is not present in the properties of interest and appears only unprotected (clear) in t. We formally define *clear terms* as follows.

Definition 18 (Clear terms)**.** A term u is *clear* in a term t if $u \notin \mathit{subs}(t) \setminus \mathit{split}(t)$, i.e., u occurs only as a field or not at all in t.

Intuitively, the presence of unprotected atoms and variables does not make any difference in the security of a protocol. For atom removal abstractions, the intuition that such abstractions are safe is obvious. The reason is that the adversary can always replace unprotected fresh values by his own ones of the same types. Therefore, such terms are not meaningful in terms of security and can be safely removed.

Specification of Atom-and-variable Removal Abstractions In this subsection, we first present the formal definition of atom-and-variable removal abstractions. Then, we illustrate their applications with an example.

Definition 19. An *atom-and-variable removal abstraction* $\mathit{rem}_T : \mathcal{T} \to \mathcal{T} \cup \{\mathsf{nil}\}$ for a set T of atoms and variables is defined by

(i) $\mathit{rem}_T(u) = \mathsf{nil}$ if $u \in T^\flat$,

(ii) $$\mathit{rem}_T(\langle t_1, t_2\rangle) = \begin{cases} \mathit{rem}_T(t_1) & \text{if } \mathit{rem}_T(t_2) = \mathsf{nil} \\ \mathit{rem}_T(t_2) & \text{if } \mathit{rem}_T(t_1) = \mathsf{nil} \\ \langle \mathit{rem}_T(t_1), \mathit{rem}_T(t_2)\rangle & \text{otherwise} \end{cases}$$

(iii) $\mathit{rem}_T(t) = t$ for all other terms.

Point (i) allows us to remove a term whenever it is in $T^\flat$. Note that we consider $T^\flat$ instead of just T, because a removal must be done both in protocol terms, i.e., those in role specifications, and in instantiated terms, e.g., those in traces. Point (ii) allows us to remove all fields that are in $T^\flat$ from a pair. Finally, the atom-and-variable removal abstraction keeps non-pair terms unchanged (point (iii)).

Intuitively, an atom-and-variable removal abstraction does not remove all occurrences of an atom or a variable from a given term, but only its fields. That means the abstraction cannot remove anything from a non-pair term. This is because we do not know how to define a removal of an atom or a variable from a non-pair composed term $t = c(a_1, \ldots, a_n)$ for $c \in \Sigma^n$ when one of the arguments $a_1, \ldots, a_n$ goes to nil. One may think of two possible definitions in this case: (1) rem_T maps t to nil or (2) rem_T maps t to a tuple consisting of the arguments that are not nil. However, an atom-and-variable removal abstraction corresponding to either of these definitions does not preserve deducibility in general. The following example presents an atom removal abstraction that violates this property if definition (1) is chosen.

Example 26. Consider two nonces $na, nb\colon \mathsf{Nonce}$ and the terms $t = \langle na, \{\!|nb|\!\}_{na}\rangle$ and $u = nb$. Let $T = \{na\}$. Then, we have $rem_T(t) = \mathsf{nil}$ and $rem_T(u) = nb$. Moreover, we also have $t \vdash_E u$, but $rem_T(t) \vdash_E rem_T(u)$ does not hold, as nb is not deducible from nil. ♠

As an atom-and-variable removal abstraction corresponding to definition (2) may map a non-pair term to a pair, it typically changes the top-level constructor of the input term. This change is dangerous as equality modulo Ax may not be preserved and therefore intruder may no longer be able to derive a term, which he could derive before abstraction. We illustrate this phenomenon in the following example.

Example 27. Suppose that $c_1, c_2, c_3 \in \Sigma$ and Ax contains the following axiom

$$c_1(c_2(x_1, x_2), c_3(x_3)) \simeq c_1(c_2(x_2, x_3), c_3(x_1)).$$

Let $n_1, n_2, n_3\colon \mathsf{Nonce}$ and consider two terms t and u where

$$\begin{array}{rcl} t & = & c_1(c_2(n_1, n_2), c_3(n_3)), \\ u & = & c_1(c_2(n_2, n_3), c_3(n_1)). \end{array}$$

We consider the atom removal abstraction rem_T where $T = \{n_1\}$. Then we have $rem_T(t) = c_1(n_2, c_3(n_3))$ and $rem_T(u) = c_2(n_2, n_3)$. Moreover, we also have $t \vdash_E u$, since $t =_{Ax} u$. But $rem_T(u)$ is not deducible from $rem_T(t)$. ♠

Similar counterexamples can also be constructed if variable removal abstractions are considered. This highlights the necessity of point (iii) in Definition 19.

Analogous to typed abstractions, we extend atom-and-variable removal abstractions to protocol specifications, events, event sequences, and traces.

Definition 20. Let T be a set of atoms and variables. We define

(i) $rem_T(ev(t)) = ev(rem_T(t))$ for events $ev(t) \in Evt(\mathcal{T}) \setminus Sig$ and $rem_T(s) = s$ for $s \in Sig$,

(ii) for event sequences, $rem_T(\epsilon) = \epsilon$ and $rem_T(e \cdot tl) = rem_T(tl)$ if $term(rem_T(e)) = \mathsf{nil}$ and $rem_T(e \cdot tl) = rem_T(e) \cdot rem_T(tl)$ otherwise; the lifting to traces and thread pools is defined analogously.

(iii) $rem_T(P)(X) = rem_T(P(X))$ for all $X \in dom(P)$ such that $rem_T(P(X)) \neq \epsilon$ and undefined otherwise.

Note that for all terms t, nil can occur in $rem_T(t)$ only at the top-level, i.e., $rem_T(t)$ either does not contain nil or is equal to nil. From Definition 20, this ensures that nil does not occur in abstracted roles and traces. Therefore, it follows immediately from Definition 20 that $rem_T(P)$ is a protocol, which is stated in the following proposition.

Proposition 9. *Let T be a set of atoms and variables. Then $rem_T(P)$ is a protocol.*

In the following example, we illustrate the use of atom removal abstractions to transform $\mathsf{IKE}^1_\mathsf{m}$ into $\mathsf{IKE}^2_\mathsf{m}$.

Example 28 ($\mathsf{IKE}^1_\mathsf{m}$ to $\mathsf{IKE}^2_\mathsf{m}$)**.** We use atom-and-variable removal to simplify the protocol $\mathsf{IKE}^1_\mathsf{m}$. First, we recall the specification of $\mathsf{IKE}^1_\mathsf{m}$.

$$\begin{aligned}
S_{\mathsf{IKE}^1_\mathsf{m}}(A) \quad = \quad & \mathsf{send}(\underline{sPIa}, o, \underline{sA1}, \mathsf{exp}(g,x), na) \cdot \mathsf{rcv}(\underline{sPIa}, \underline{SPIb}, \underline{sA1}, Gb, Nb) \cdot \\
& \mathsf{Running} \cdot \mathsf{send}(\underline{sPIa}, \underline{SPIb}, A, B, AUTHaa, \underline{sA2}, \underline{tSa}, \underline{tSb}) \cdot \\
& \mathsf{rcv}(\underline{sPIa}, \underline{SPIb}, B, AUTHba, \underline{sA2}, \underline{tSa}, \underline{tSb}) \cdot \mathsf{Secret} \cdot \mathsf{Commit}
\end{aligned}$$

$$\begin{aligned}
S_{\mathsf{IKE}^1_\mathsf{m}}(B) \quad = \quad & \mathsf{rcv}(\underline{SPIa}, o, \underline{sA1}, Ga, Na) \cdot \mathsf{send}(SPIa, \underline{sPIb}, \underline{sA1}, \mathsf{exp}(g,y), nb) \cdot \\
& \mathsf{rcv}(\underline{SPIa}, \underline{sPIb}, A, B, AUTHab, \underline{sA2}, \underline{tSa}, \underline{tSb}) \cdot \mathsf{Running} \cdot \\
& \mathsf{send}(\underline{SPIa}, \underline{sPIb}, B, AUTHbb, \underline{sA2}, \underline{tSa}, \underline{tSb}) \cdot \mathsf{Secret} \cdot \mathsf{Commit}
\end{aligned}$$

We remove the constants $sA1$, $sA2$, tSa, and tSb, the fresh values $sPIa$ and $sPIb$, and the variables $SPIa$ and $SPIb$ using the atom-and-variable removal abstraction rem_T, where $T = \{sA1, sA2, tSa, tSb, sPIa, sPIb, SPIa, SPIb\}$. Note that we cannot remove the constant o because o is not clear in the authenticators $AUTHaa$ and $AUTHab$, hence does not satisfy the soundness condition which we will define later. In the specification of $\mathsf{IKE}^1_\mathsf{m}$, we underline terms that are removed in this step to highlight the changes between the original and the abstract protocols. Applying rem_T to $\mathsf{IKE}^1_\mathsf{m}$, we obtain the protocol $\mathsf{IKE}^2_\mathsf{m}$ as given below.

$$\begin{aligned}
S_{\mathsf{IKE}^2_\mathsf{m}}(A) \quad = \quad & \mathsf{send}(o, \mathsf{exp}(g,x), na) \cdot \mathsf{rcv}(Gb, Nb) \cdot \\
& \mathsf{Running} \cdot \mathsf{send}(A, B, AUTHaa) \cdot \\
& \mathsf{rcv}(B, AUTHba) \cdot \mathsf{Secret} \cdot \mathsf{Commit}
\end{aligned}$$

$$\begin{aligned}
S_{\mathsf{IKE}^2_\mathsf{m}}(B) \quad = \quad & \mathsf{rcv}(o, Ga, Na) \cdot \mathsf{send}(\mathsf{exp}(g,y), nb) \cdot \\
& \mathsf{rcv}(A, B, AUTHab) \cdot \mathsf{Running} \cdot \\
& \mathsf{send}(B, AUTHbb) \cdot \mathsf{Secret} \cdot \mathsf{Commit}
\end{aligned}$$

Note that the session keys and the authenticators are non-pair composed terms and hence remain unchanged by an atom-and-variable removal. In Subsection 4.2.2, we will use a redundancy removal to further simplify $\mathsf{IKE}^2_\mathsf{m}$ by removing the constant o and the agent variables A and B from the role descriptions. ♠

In order to show that the abstraction step in Example 28 preserves attacks, we need to define sufficient conditions on protocols and security formulas under which an atom-and-variable removal is sound. Analogous to typed abstractions, the soundness result for atom-and-variable removal requires safeness conditions for security properties. We first introduce some auxiliary definitions: we denote by $atoms(t)$ and $fresh(t)$ the set of atoms and fresh values occurring in t, respectively; we also define $av(t) = atoms(t) \cup vars(t)$; we extend the notion of clearness to a set of terms T as expected, i.e., T is *clear* in a term t if every term in T is; we also say that T is clear in a set of terms T' if every $t \in T$ is clear in every $t' \in T'$. Below, we define what it means for a security formula to be safe with respect to an atom-and-variable removal abstraction.

Definition 21 ((P, rem_T)-safe formulas)**.** We assume a set of atoms and variables $T \subseteq av(\mathcal{M}_P)$. A formula $\phi \in \mathcal{L}_P$ is *(P, rem_T)-safe* if the following conditions hold:

(i) $T^\sharp \cap (av(Sec_\phi \cup EqTerm_\phi)) = \emptyset$,

(ii) for all $e(t) \in Evt_\phi$, we have $rem_T(t) \neq \mathsf{nil}$, and

(iii) $rem_T(m) = rem_T(m')$ implies $m = m'$ for all $s(m) \in Evt^+_\phi$ and $s(m') \in Evt(\mathcal{M}_P)$.

Condition (i) requires that no removed atom/variable occurs in secrecy statements and equalities. This ensures that the removal of these values does not interfere with the properties of interest. Condition (ii) requires that no event terms in the property are mapped to nil. This ensures that the corresponding abstracted property does not contain any nil event. Finally, condition (iii) is similar to that of Definition 11, i.e., it ensures that no two events, one from the protocol specification and one occurring positively in the property, are mapped to the same event under an atom-and-variable removal abstraction.

Note that for a (P, rem_T)-safe property formula ϕ, no terms occurring in the events, the equalities, or the secrecy statements of ϕ are mapped to nil by rem_T. We therefore extend rem_T to (P, rem_T)-safe property formulas in the same way as we did for typed abstractions.

Soundness result for atom-and-variable removal abstractions Note that an atom-and-variable removal abstraction can be seen as a composition of an atom removal abstraction and a variable removal abstraction. The soundness of atom-and-variable removal abstractions therefore boils down to that of these two types of protocol abstractions. In order to establish our the soundness result for variable removal abstractions, we work with a subclass of protocols, namely *well-formed* protocols. First, we introduce the notion of *extractable variables*.

Definition 22 (Extractable variables)**.** Let t be a term and X be a variable. We say that X is *extractable from* t if either

(i) $t = X$ or

(ii) $top(t)$ is defined and there is $j \in pp(top(t))$ such that X is extractable from $t|_j$.

Intuitively, a variable is extractable from a term if the path from the root of the term to the variable consists of only extractable indices. This is to ensure that if X is extractable then it could be deducible. For example, X is extractable from $\{|X|\}_k$ for some term k since X could be derived from $\{|X|\}_k$ using the rewrite rule $\{|\{|X|\}_K|\}^{-1}_K \to X$. In contrast, X is not extractable from $\mathsf{h}(X)$ since there is no way to deduce X from $\mathsf{h}(X)$. However, X is extractable from $\langle X, \mathsf{h}(X)\rangle$ since X is extractable from the first component of the pair.

We now give the formal definition of *well-formed protocols*.

Definition 23. A protocol P is *well-formed* if all non-agent variables first occur in receive events, i.e., for all events e in a role $P(R)$ and all variables $X \in vars(term(e))$ such that $\Gamma(X) \neq \alpha$, there is an event $\mathsf{rcv}(t)$ in $P(R)$ such that $\mathsf{rcv}(t)$ equals or precedes e in $P(R)$ and X is extractable from t.

A well-formed protocol captures the intuition that an agent must know what he sends and the encrypted messages he receives must be properly decrypted. This is a reasonable condition for precisely modeling protocols. Our soundness result for atom-and-variable removal abstractions is stated in the following theorem.

Theorem 5 (Soundness for atom-and-variable removal abstractions)**.** *Let P be a well-formed protocol, $T \subseteq av(\mathcal{M}_P)$ and $\phi \in \mathcal{L}_P$ be a formula such that*

(i) T is clear in $\mathcal{M}_P$,

(ii) $T^\sharp \cap fresh(IK_0) = \emptyset$,

(iii) $IK_0 \subseteq IK_0'$, and

(iv) ϕ is (P, rem_T)-safe

Let (tr, th, σ) be a reachable state of P. Then there is a substitution θ such that the following holds.

1. *$(rem_T(tr), rem_T(th), \theta)$ is reachable in $rem_T(P)$, and*
2. *$(tr, th, \sigma) \nvDash \phi$ implies $(rem_T(tr), rem_T(th), \theta) \nvDash rem_T(\phi)$.*

In order to preserve attacks, condition (i) restricts the removal of atoms and variables from a protocol term t to those that are clear in t. Condition (ii) requires that the removed atoms do not occur in IK_0. Condition (iii) requires that the initial knowledge of the intruder in the abstract protocol subsumes that in the original protocol. Finally, condition (iv) ensures that the considered property satisfies the safeness condition.

Note that without condition (i), deducibility preservation may not hold as it is shown in the following example.

Example 29. Consider a nonce $na\colon \mathsf{Nonce}$, and terms $t = \langle na, \mathsf{h}_1(na)\rangle$ and $u = \mathsf{h}_2(na)$, where h_1 and h_2 are hash function symbols. We consider the atom removal rem_{At} where $At = \{na\}$. Then we have $rem_{At}(t) = \mathsf{h}_1(na)$ and $rem_{At}(u) = \mathsf{h}_2(na)$. Moreover, we also have $t \vdash_E u$, but $rem_{At}(u)$ is not deducible from $rem_{At}(t)$. ♠

As mentioned before, the proof of Theorem 5 can be achieved by establishing the soundness result for atom removal and variable removal abstractions seperately. These proofs can be found in Appendices C.1 and C.2.

4.2.2. Redundancy Removal Abstractions

A redundancy removal abstraction rd enables the elimination of redundancies within each role of a protocol. Intuitively, a protocol term t appearing in a role r can be abstracted to $rd(t)$ if t and $rd(t)$ are derivable from each other under the intruder knowledge T containing the terms preceding t in r and the initial knowledge IK_0. For example, we can simplify $r = \mathsf{snd}(t) \cdot \mathsf{rcv}(\langle t, u \rangle)$ to $\mathsf{snd}(t) \cdot \mathsf{rcv}(u)$. In contrast to atom-and-variable removal, redundancy removal can also remove composed terms. It is therefore a very effective ingredient for automatic abstraction, which we describe in Chapter 5.

Unfortunately, redundancy removal abstractions do not preserve the well-formedness of security protocols without additional assumptions. In the following example, we illustrate such a case.

Example 30. Suppose that $c \in \Sigma^2$, $d \in \Sigma^1$ is a hash function, $d(c(X, Y)) \to Y \in R$, and $c(\mathsf{h}(X), Y) \simeq c(\mathsf{h}(Y), X) \in Ax$ where h is a hash function. Let P be a protocol of two roles whose specifications are given as follows.

$$\begin{array}{rcl} P(A) & = & \mathsf{send}(na_1, \mathsf{h}(na_1), na_2) \\ P(B) & = & \mathsf{rcv}(Na_1, \mathsf{h}(Na_1), Na_2) \end{array}$$

where na_1 and na_2 are nonces. It is clear that P is well-formed since Na_1 and Na_2 are both extractable from $\langle Na_1, \mathsf{h}(Na_1), Na_2 \rangle$. Let rd be a redundancy abstraction such that

$$\begin{array}{rcl} rd(\langle na_1, \mathsf{h}(na_1), na_2 \rangle) & = & \langle na_1, \mathsf{h}(na_1), na_2 \rangle \\ rd(\langle Na_1, \mathsf{h}(Na_1), Na_2 \rangle) & = & \langle Na_1, c(h(Na_2), Na_1) \rangle \end{array}$$

Note that $Na_1, \mathsf{h}(Na_1), Na_2 \vdash_E Na_1, c(\mathsf{h}(Na_2), Na_1)$. Moreover, we also have

$$\begin{array}{rcl} Na_1, c(\mathsf{h}(Na_2), Na_1) & \vdash_E & Na_1, \mathsf{h}(Na_1), c(\mathsf{h}(Na_2), Na_1) \\ & \vdash_E & Na_1, \mathsf{h}(Na_1), c(\mathsf{h}(Na_1), Na_2) \\ & \vdash_E & Na_1, \mathsf{h}(Na_1), d(c(\mathsf{h}(Na_1), Na_2)) \\ & \vdash_E & Na_1, \mathsf{h}(Na_1), Na_2 \end{array}$$

However, $rd(P)$ is not well-formed as Na_2 is not extractable from $\langle Na_1, c(h(Na_2), Na_1) \rangle$. ♠

Specification of redundancy removal abstractions We now formally define our class of redundancy removal abstractions.

Definition 24. A function $rd : \mathcal{M}_P \to \mathcal{M} \cup \{\mathsf{nil}\}$ is a *redundancy removal abstraction* for a protocol P if the following conditions holds:

(i) for all $R \in dom(P)$, we have that $RD_{rd}(IK_0, P(R))$ is derivable, where the predicate $RD_{rd}(T, S)$ is inductively defined by the following two rules:

$$\frac{}{RD_{rd}(T, \epsilon)} \qquad \frac{RD_{rd}(T, r)}{RD_{rd}(T, s \cdot r)}\ s \in Sig$$

$$\frac{RD_{rd}(T \cup \{t\}, r) \quad T, \mathcal{V}_\alpha, rd(t) \vdash_E t \quad T, \mathcal{V}_\alpha, t \vdash_E rd(t)}{RD_{rd}(T, ev(t) \cdot r)}\ ev \in \{\mathsf{send}, \mathsf{rcv}\}$$

Note that in these rules, $rd(t)$ is removed from the deducibility conditions if it equals nil. We also define $rd(t^{\sharp i}) = rd(t)^{\sharp i}$ for all $i \in TID$ and lift rd over roles, traces, protocols, and thread pools in the same way as we did for atom-and-variable removal abstractions.

(ii) for all non-agent variables $X \in \mathcal{V}_P$ and all receive events $\mathsf{rcv}(t)$ in which X first occurs, we have that X is extractable from $rd(t)$.

Intuitively, the predicate $RD_{rd}(T, ev(t) \cdot r)$ ensures for a protocol message t that the intruder is able to derive t from $rd(t)$ and his knowledge T, and vice versa. The first rule says that $RD_{rd}(T, \epsilon)$ always holds. This captures the intuition that any redundancy removal works for the empty role description. The second rule allows us to ignore all the signals events as they do not affect the intruder's knowledge. In the last rule, the first premise requires that the predicate holds for T plus the term t in the first element of the event sequence, and the tail r. By adding t to T, we capture the fact that the intruder learns t after the event $ev(t)$ has been executed. The second premise ensures that t is derivable from T, $\mathcal{V}_\alpha$, and $rd(t)$. The set of agent variables $\mathcal{V}_\alpha$ is added to T to symbolically represent the intruder knowledge of all agent names. The third premise is the same as the second one, except that the roles of t and $rd(t)$ are swapped. Note that condition (ii) eliminates the violation of well-formedness preservation in Example 30.

In the following example, we illustrate the use of redundancy removal abstractions to further simplify the protocol $\mathsf{IKE}^2_\mathsf{m}$.

Example 31. First, we recall $\mathsf{IKE}^2_\mathsf{m}$ whose role descriptions are given below.

$$\begin{aligned} S_{\mathsf{IKE}^2_\mathsf{m}}(A) &= \mathsf{send}(\underline{o}, \mathsf{exp}(g, x), na) \cdot \mathsf{rcv}(Gb, Nb) \cdot \\ &\quad\ \mathsf{Running} \cdot \mathsf{send}(\underline{A}, \underline{B}, AUTHaa) \cdot \\ &\quad\ \mathsf{rcv}(\underline{B}, AUTHba) \cdot \mathsf{Secret} \cdot \mathsf{Commit} \end{aligned}$$

$$\begin{aligned} S_{\mathsf{IKE}^2_\mathsf{m}}(B) &= \mathsf{rcv}(\underline{o}, Ga, Na) \cdot \mathsf{send}(\mathsf{exp}(g, y), nb) \cdot \\ &\quad\ \mathsf{rcv}(\underline{A}, \underline{B}, AUTHab) \cdot \mathsf{Running} \cdot \\ &\quad\ \mathsf{send}(\underline{B}, AUTHbb) \cdot \mathsf{Secret} \cdot \mathsf{Commit} \end{aligned}$$

To remove the underlined terms, we consider the following function rd:

$$\begin{aligned} rd(\langle \underline{o}, \mathsf{exp}(g, x), na \rangle) &= \langle \mathsf{exp}(g, x), na \rangle \\ rd(\langle \underline{A}, \underline{B}, AUTHaa \rangle) &= AUTHaa \\ rd(\langle \underline{B}, AUTHba \rangle) &= AUTHba \\ rd(\langle \underline{o}, Ga, Na \rangle) &= \langle Ga, Na \rangle \\ rd(\langle \underline{A}, \underline{B}, AUTHab \rangle) &= AUTHab \\ rd(\langle \underline{B}, AUTHbb \rangle) &= AUTHbb \\ rd(t) &= t \text{ for all other messages } t \end{aligned}$$

It is not hard to see that rd is a redundancy removal abstraction according to Definition 24. Applying rd to $\mathsf{IKE}^2_\mathsf{m}$, we obtain the protocol $\mathsf{IKE}^3_\mathsf{m}$ whose specification is as

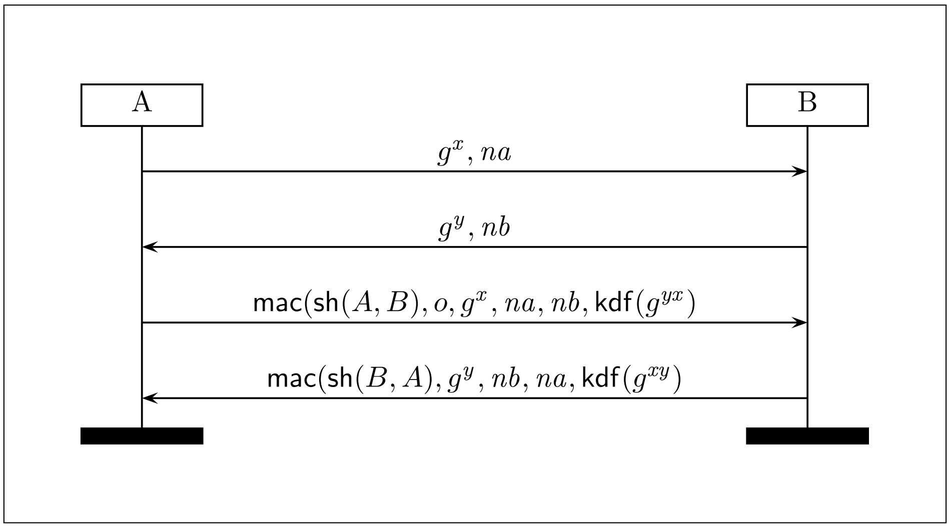

Figure 4.1.: The $\mathsf{IKE}^3_\mathsf{m}$ protocol.

follows.

$$\begin{array}{rcl}
S_{\mathsf{IKE}^2_\mathsf{m}}(A) & = & \mathsf{send}(\mathsf{exp}(g,x), na) \cdot \mathsf{rcv}(Gb, Nb) \cdot \\
& & \mathsf{Running} \cdot \mathsf{send}(AUTHaa) \cdot \\
& & \mathsf{rcv}(AUTHba) \cdot \mathsf{Secret} \cdot \mathsf{Commit} \\
\\
S_{\mathsf{IKE}^2_\mathsf{m}}(B) & = & \mathsf{rcv}(Ga, Na) \cdot \mathsf{send}(\mathsf{exp}(g,y), nb) \cdot \\
& & \mathsf{rcv}(AUTHab) \cdot \mathsf{Running} \cdot \\
& & \mathsf{send}(AUTHbb) \cdot \mathsf{Secret} \cdot \mathsf{Commit}
\end{array}$$

In Figure 4.1, we depict the description of this protocol. ♠

Soundness result for redundancy removal abstractions First, we extend redundancy removal abstractions to security property formulas. Then, we introduce the class of formulas that are safe for these abstractions. Finally, we present our soundness result.

Suppose that rd is a redundancy removal abstraction for P. We define $rd(A) = A$ for all atomic predicates and equalities A, except the following two, which are related to events:

$$\begin{array}{rcl}
rd(steps(i,e)) & = & steps(i, rd(e)) \\
rd((i,e) \prec (j,e')) & = & (i, rd(e)) \prec (j, rd(e'))
\end{array}$$

We lift rd to $\neg$, $\vee$, $\wedge$, $\forall$, and $\exists$ as usual.

Analogous to atom-and-variable removal abstractions, we require the following two conditions for a formula to be (P, rd)-safe.

Definition 25 ((P, rd)-safe formulas)**.** Let rd a redundancy abstraction for P. A formula ϕ is *(P, rd)-safe* if

(i) $rd(t) = rd(u)$ implies $t = u$, for all $e(t) \in Evt^+_\phi$ and $e(u) \in Evt(\mathcal{M}_P)$,

(ii) for all $ev(t) \in Evt_{\phi}$, we have $rd(t) \neq \mathsf{nil}$.

The soundness result for redundancy removal abstractions is stated in the following theorem.

Theorem 6 (Soundness for redundancy removal abstractions)**.** *Let rd be a redundancy removal abstraction for P and $\phi \in \mathcal{L}_P$ be a (P, rd)-safe formula. Suppose that $IK_0 \subseteq IK'_0$. Then, for all states (tr, th, σ) reachable in P, we have*

1. *$(rd(tr), rd(th), \sigma)$ is reachable in $rd(P)$ and*
2. *$(tr, th, \sigma) \nvDash \phi$ implies $(rd(tr), rd(th), \sigma) \nvDash rd(\phi)$.*

4.3. Splitting Abstractions

In this section, we present a new class of protocol abstractions called *splitting abstractions*, which directly transform protocol roles. Intuitively, splitting abstractions allow us to split a protocol event into several ones. These abstractions are typically used to reduce the size of protocol messages. They can be used together with atom-and-variable removal abstractions to, for example, remove ticket forwarding in key establishment protocols such as Kerberos variants (see Section 4.5).

Splitting abstractions have been previously introduced by Hui and Lowe [72]. Here we formally justify the soundness of these abstractions in our setting.

4.3.1. A Motivating Example

We start from a core version of Kerberos IV protocol, called K4. This protocol aims at providing mutual authentication and session key distribution for client/server applications by using symmetric encryptions [82]. In Alice&Bob notation, the protocol K4 reads as follows.

K4(1). $A \to S : A, B, na$
K4(2). $S \to A : \{\!|B, ts, na, k_{ab}, \{\!|A, ts, k_{ab}|\!\}_{\mathsf{sh}(B,S)}|\!\}_{\mathsf{sh}(A,S)}$
K4(3). $A \to B : \{\!|A, ts, k_{ab}|\!\}_{\mathsf{sh}(B,S)}, \{\!|c, ta|\!\}_{k_{ab}}$
K4(4). $B \to A : \{\!|ta|\!\}_{k_{ab}}$

In this protocol, a client A wants to communicate with an application server B through a trusted third party S, also known as Key Distribution Center (KDC), who shares a secret key with each of them. Initially, A requests a ticket from S by sending his name together with B's identity and a nonce na. The KDC S sends to A a client ticket that contains a session key k_{ab}, a timestamp, and a server ticket for B with similar information, all encrypted with the pre-shared key with A. Once received, A uses the session key to encrypt a timestamp ta together with a constant c. He then sends this encrypted message (known as A's authenticator) together with the server ticket to B. Finally, B re-encrypts the timestamp ta with the session key and sends this message to

A. In this way, B allows A to verify his identity and to confirm B's reception of the session key.

Suppose that we are interested in the secrecy of the session key k_{ab}. To improve the performance of verification tools, we remove protocol elements that are irrelevant for a given property to hold. In the first abstraction step, we pull B's ticket out of the encryption in message K4(2) using a typed abstraction. This simplification aims at removing the unnecessary double encryption in this message. The result is the core of Kerberos V, called K5, which differs from K4 as follows.

$$\mathsf{K5}(2). \qquad S \to A : \{\!|B, ts, na, k_{ab}|\!\}_{\mathsf{sh}(A,S)}, \{\!|A, ts, k_{ab}|\!\}_{\mathsf{sh}(B,S)}$$

In the second step, we eliminate the forwarding of B's ticket by A by applying a combination of a splitting abstraction to split K5(2) into two events and an atom-and-variable removal abstraction to remove B's ticket from K5(3). This allows the KDC S to send the server and the client tickets to the corresponding receivers directly. The result is protocol K3. We omit the message K3(1) which equals K5(1).

$$\begin{array}{ll}
\mathsf{K3}(2). & S \to A : \{\!|B, ts, na, k_{ab}|\!\}_{\mathsf{sh}(A,S)} \\
\mathsf{K3}(3). & S \to B : \{\!|A, ts, k_{ab}|\!\}_{\mathsf{sh}(B,S)} \\
\mathsf{K3}(4). & A \to B : \{\!|c, ta|\!\}_{k_{ab}} \\
\mathsf{K3}(5). & B \to A : \{\!|ta|\!\}_{k_{ab}}
\end{array}$$

In the third step, we pull ts and na out of the encryptions in K3(2) and K3(3) since these terms are irrelevant to the property of interest. We also remove the encryptions in the last two messages since these protections are unnecessary for achieving session-key secrecy. This results in the protocol K2.

$$\begin{array}{ll}
\mathsf{K2}(1). & A \to S : A, B, na \\
\mathsf{K2}(2). & S \to A : \{\!|B, k_{ab}|\!\}_{\mathsf{sh}(A,S)}, ts, na \\
\mathsf{K2}(3). & S \to B : \{\!|A, k_{ab}|\!\}_{\mathsf{sh}(B,S)}, ts \\
\mathsf{K2}(4). & A \to B : c, ta \\
\mathsf{K2}(5). & B \to A : ta
\end{array}$$

In a final transformation, we remove the timestamps ta and ts, the constant c, and the initiator nonce na. The result is protocol K1 for which we verify the secrecy of the session key.

$$\begin{array}{ll}
\mathsf{K1}(1). & A \to S : A, B \\
\mathsf{K1}(2). & S \to A : \{\!|B, k_{ab}|\!\}_{\mathsf{sh}(A,S)} \\
\mathsf{K1}(3). & S \to B : \{\!|A, k_{ab}|\!\}_{\mathsf{sh}(B,S)}
\end{array}$$

The transformation from K5 to K3 cannot be achieved by typed or untyped abstractions alone, because it involves changing the structure of the protocol. This motivates our introduction of splitting abstractions.

4.3.2. Specification of Splitting Abstractions

We provide a formal definition of splitting abstractions as follows.

Definition 26 (Splitting abstractions)**.** A function f_{sp}: $Evt(\mathcal{M}) \rightarrow Evt(\mathcal{M})^+$ is called a *splitting abstraction* iff for all $ev \in Evt(\mathcal{M})$, we have that

(i) if $ev \in Sig$ then $f_{sp}(ev) = ev$,

(ii) if $ev = s(t)$ for some $s \in \{\mathsf{send}, \mathsf{rcv}\}$ then $f_{sp}(s(t)) = s(t_1) \cdots s(t_n)$ where $n \geq 1$ and $\bigcup_{i=1}^{n} split(t_i) = split(t)$.

Intuitively, condition (i) ensures that the abstraction does not modify a signal event. Condition (ii) allows us to split a send/receive event into several ones. This condition also ensures a splitting abstraction preserves fields. In the following, we lift f_{sp} to indexed events, event sequences, thread pools, traces, and property formulas.

Definition 27. Let P be a protocol and $f_{sp} : Evt(\mathcal{M}) \rightarrow Evt(\mathcal{M})^+$ be a splitting abstraction. We define

- for events $s(t) \in Evt(\mathcal{M}) \setminus Sig$ and $i \in TID$, we have $f_{sp}(s(t^{\sharp i})) = s(t_1^{\sharp i}) \cdots s(t_n^{\sharp i})$ where $f_{sp}(s(t)) = s(t_1) \cdots s(t_n)$,
- for event sequences, $f_{sp}(\epsilon) = \epsilon$ and $f_{sp}(e \cdot tl) = f_{sp}(e) \cdot f_{sp}(tl)$; the lifting to roles, protocols, and thread pools is defined analogously, and
- for traces, $f_{sp}(\epsilon) = \epsilon$ and $f_{sp}((i, s(t)) \cdot tl) = (i, s(t_1)) \cdots (i, s(t_n)) \cdot f_{sp}(tl)$ where $f_{sp}(s(t)) = s(t_1) \cdots s(t_n)$.

We now extend splitting abstractions to security properties. Analogous to redundancy removal abstractions, we define $f_{sp}(A) = A$ for all atomic predicates and equalities, except the following two, which are related to events:

$$\begin{aligned} f_{sp}(steps(i, e)) &= steps(i, e_1) \wedge \ldots \wedge steps(i, e_n) \\ f_{sp}((i, e) \prec (j, e')) &= (i, e_n) \prec (j, e'_1) \end{aligned}$$

where $n, m \geq 1$ and

$$\begin{aligned} f_{sp}(e) &= e_1 \cdots e_n, \\ f_{sp}(e') &= e'_1 \cdots e'_m. \end{aligned}$$

We lift f_{sp} to $\neg$, $\vee$, $\wedge$, $\forall$, and $\exists$ as usual.

4.3.3. Soundness Result for Splitting Abstractions

In this subsection, we present a soundness result for splitting abstractions. First, we show that splitting abstractions preserve reachability.

Proposition 10 (Reachability preservation)**.** *Let P be a protocol and f_{sp}: $Evt(\mathcal{M}) \rightarrow Evt(\mathcal{M})^+$ be a splitting abstraction. Assume that $IK_0 \subseteq IK'_0$. Then for all states (tr, th, σ) reachable in P, we have $(f_{sp}(tr), f_{sp}(th), \sigma)$ is reachable in $f_{sp}(P)$.*

Next, we prove that splitting abstractions preserve attacks. For attack preservation to hold, we require that every event in an abstracted protocol is split from a unique event in the original protocol. Intuitively, this condition is required to preserve attacks on properties involving event occurrences and orderings. Let $Event_P$ be the set of events of a protocol P. We now formally define a class of protocols that satisfy the condition.

Definition 28. A protocol P is *non-overlapping* if no message field is sent or received more than once, i.e., $split(t) \cap split(t') = \emptyset$ holds for all events $\{s(t), s(t')\} \subseteq Event_P$ that are (i) at two different positions in a role or (ii) from two different roles.

Note that an overlapping protocol contains redundant fields. Therefore, we can easily make a protocol non-overlapping by first applying a redundancy removal abstraction.

In the following theorem, we prove our attack preservation result for non-overlapping protocols.

Theorem 7 (Attack preservation)**.** *Let P be a non-overlapping protocol, $f\colon Event_P \to Evt^*$ be a splitting abstraction, and $\phi \in \mathcal{L}_P$ be closed. Then we have, for all states (tr, th, σ) reachable in P,*

- *$(f_{sp}(tr), f_{sp}(th), \sigma)$ is reachable in $f_{sp}(P)$,*
- *$(tr, th, \sigma) \nvDash_E \phi$ implies $(f_{sp}(tr), f_{sp}(th), \sigma) \nvDash_E f_{sp}(\phi)$.*

4.4. Well-formedness Preservation for Protocol Abstractions

In this section, we show that protocol abstractions preserve well-formedness. First, we present the well-formedness preservation result for typed abstractions. The proof can be found in Appendix A.6.

Proposition 11. *Let P be a well-formed protocol and $F_f = (f, E_f)$ be a typed abstraction. Then $f(P)$ is well-formed.*

In the following proposition, we prove that atom-and-variable removal abstractions preserve well-formedness.

Proposition 12. *Let T be a set of atoms and variables such that $vars(T)$ is clear in $\mathcal{M}_P$. If P is well-formed then so is $rem_T(P)$.*

Proof. It is sufficient to consider the case where $T = V$ contains only variables. Since all variables in V are clear in $\mathcal{M}_P$, all occurrences of these variables are removed from the roles of P and all other variables are kept under rem_V. Let $R \in dom(P)$ and e is an event in $rem_V(P(R))$. Suppose that $X \in vars(term(rem_V(e)))$ such that $\Gamma(X) \neq \alpha$. Then we have $X \in vars(term(e))$ and $X \notin V$. As P is well-formed, we know that X must be present in a receive event $\mathsf{rcv}(t)$ preceding e. Since $X \notin V$, we have that X is also present in $rem_V(rem_V(t))$. Moreover, we know that X is extractable from t. Since $X \notin V$, we have that X is also extractable from $rem_V(t)$. That means $rem_V(P)$ is well-formed. This completes the proof of the proposition. □

We now state the well-formedness preservation result for redundancy removal abstractions in the following proposition.

Proposition 13. *Let rd be a redundancy removal abstraction and P be a well-formed protocol. Then $rd(P)$ is well-formed.*

Proof. Let $R \in dom(P)$, $X \in \mathcal{V}_{rd(P)}$ be a non-agent variable, and an event $ev(t) \in P(R)$ such that $X \in vars(rd(t))$. Note that $\mathcal{V}_{rd(P)} = \mathcal{V}_P$. Since P is well-formed, there must be a receive event $\mathsf{rcv}(t') \in P(R)$ in which X first occurs and X is extractable from t'. We define a set of terms

$$T = \{u \mid ev(u) \in P(R) \wedge ev(u) \text{ precedes or equals } \mathsf{rcv}(t')\}.$$

We know that $IK_0, T, \mathcal{V}_\alpha, rd(t') \vdash_E t'$. Since $\mathsf{rcv}(t')$ is the first receive event in which X occurs, we have $X \notin vars(T)$. Thus, we derive that $X \in vars(rd(t'))$. By Definition 24(ii), we know that X is extractable from $rd(t')$. Moreover, $\mathsf{rcv}(rd(t'))$ precedes or equals $ev(t)$ in $rd(P)(R)$. Therefore, we conclude that $rd(P)$ is well-formed. □

Finally, we show that splitting abstractions also preserve well-formedness.

Proposition 14. *Let P be a well-formed non-overlapping protocol and $f_{sp}\colon Event_P \to Evt^*$ be a splitting abstraction. Then $f_{sp}(P)$ is well-formed.*

Proof. Let $R \in dom(P)$, $X \in \mathcal{V}_{rd(P)}$ be a non-agent variable, and an event $ev(t) \in P(R)$ such that $X \in vars(rd(t))$. Then we have $\mathcal{V}_{rd(P)} = \mathcal{V}_P$. Since P is well-formed, we know that there is a receive event $\mathsf{rcv}(t') \in P(R)$ in which X first occurs and X is extractable from t'. By Definition 26, we have $f_{sp}(\mathsf{rcv}(t')) = \mathsf{rcv}(t'_1) \cdots \mathsf{rcv}(t'_n)$ where $n \geq 1$ and $\bigcup_{i=1}^{n} split(t'_i) = split(t')$. Since $\bigcup_{i=1}^{n} split(t'_i) = split(t')$, there exists the smallest $i \in \widetilde{n}$ such that $X \in vars(t'_i)$. Since X is extractable from t', we derive that X is extractable from t'_i. This implies that $f_{sp}(P)$ is well-formed. □

Propositions 11, 12, 13, and 14 enable different combinations of typed abstractions, atom-and-variable removal abstractions, redundancy removal abstractions, and splitting abstractions to transform well-formed protocols.

4.5. Example: Abstraction of Kerberos Protocol

In this subsection, we formalize our abstractions presented in Section 4.3.1 by combining all three types of protocol abstractions we have seen. First, we introduce a type for session keys, namely $\mathsf{SessionKey}$, and a type for timestamps, namely $\mathsf{TimeStamp}$. We specify the original protocol $\mathsf{K4}$ as follows:

$$\begin{aligned}
\mathsf{K4}(A) &= \mathsf{send}(A, B, na) \cdot \mathsf{rcv}(\{\!|B, Ts, na, K_{ab}, X|\!\}_{\mathsf{sh}(A,S)}) \cdot \\
&\quad\ \mathsf{send}(X, \{\!|c, ta|\!\}_{K_{ab}}) \cdot \mathsf{rcv}(\{\!|ta|\!\}_{K_{ab}}) \cdot \mathsf{Secret} \\[1em]
\mathsf{K4}(B) &= \mathsf{rcv}(\{\!|A, Ts, K_{ab}|\!\}_{\mathsf{sh}(B,S)}, \{\!|c, Ta|\!\}_{K_{ab}}) \cdot \mathsf{send}(\{\!|Ta|\!\}_{K_{ab}}) \cdot \mathsf{Secret} \\[1em]
\mathsf{K4}(S) &= \mathsf{rcv}(A, B, Na) \cdot \mathsf{send}(\{\!|B, ts, Na, k_{ab}, \{\!|A, ts, k_{ab}|\!\}_{\mathsf{sh}(B,S)}|\!\}_{\mathsf{sh}(A,S)})
\end{aligned}$$

The variables and their types are given by $Na : \mathsf{Nonce}$, $X : \mathsf{msg}$, $K_{ab} : \mathsf{SessionKey}$, and Ts, $Ta\colon \mathsf{TimeStamp}$. We formulate the secrecy properties for role A and role B as follows.

$$\phi_s^A = \forall j.\ (role(j, A) \wedge honest(j, [A, B, S]) \wedge steps(j, \mathsf{Secret})) \Rightarrow secret({K_{ab}}^j)$$

$$\phi_s^B = \forall j.\ (role(j, B) \wedge honest(j, [A, B, S]) \wedge steps(j, \mathsf{Secret})) \Rightarrow secret({K_{ab}}^j)$$

Transforming K4 **into** K5

In this transformation, we pull B's ticket out of the encryptions in the message sent by S and in the second message received by A using a typed abstraction $F_{f_1} = (f_1, E_{f_1})$ where E_{f_1} consists of the following clauses with variables X_1, Y_1: α, Y_4: γ_c, X_2, Y_2, Y_5, Z_1: TimeStamp, X_3: Nonce, $X_4, Y_3, X_5, Z_2, Z_3, K$: msg.

$$\begin{aligned}
f_1(\{\!|X_1, X_2, X_3, X_4, X_5|\!\}_K) &= \langle \{\!|f_1(X_1), f_1(X_2), f_1(X_3), f_1(X_4)|\!\}_{f_1(K)}, f(X_5)\rangle \\
f_1(\{\!|Y_1, Y_2, Y_3|\!\}_K) &= \{\!|f_1(Y_1), f_1(Y_2), f_1(Y_3)|\!\}_{f_1(K)} \\
f_1(\{\!|Y_4, Y_5|\!\}_K) &= \{\!|f_1(Y_4), f_1(Y_5)|\!\}_{f_1(K)} \\
f_1(\{\!|Z_1|\!\}_K) &= \{\!|f_1(Z_1)|\!\}_{f_1(K)} \\
f_1(\langle Z_2, Z_3\rangle) &= \langle f_1(Z_2), f_1(Z_3)\rangle
\end{aligned}$$

Intuitively, the first clause performs the main task, i.e., pulling B's ticket out of the encryption. The second clause is used to transform B's ticket. The third clause transforms A's authenticator. The fourth clause transforms the last messages of role A and role B. Finally, the last clause transforms all pairs. All clauses except for the first clause, are homomorphic. Note that the first four clauses for symmetric encryptions are pattern-disjoint. Therefore, F_{f_1} is pattern-disjoint.

Applying this abstraction allows us to obtain $\mathsf{K5} = f_1(\mathsf{K4})$ defined as follows.

$$\begin{aligned}
\mathsf{K5}(A) &= \mathsf{send}(A, B, na) \cdot \mathsf{rcv}(\{\!|B, Ts, na, K_{ab}|\!\}_{\mathsf{sh}(A,S)}, X) \cdot \\
&\quad\ \mathsf{send}(X, \{\!|c, ta|\!\}_{K_{ab}}) \cdot \mathsf{rcv}(\{\!|ta|\!\}_{K_{ab}}) \cdot \mathsf{Secret} \\[1em]
\mathsf{K5}(B) &= \mathsf{rcv}(\{\!|A, Ts, K_{ab}|\!\}_{\mathsf{sh}(B,S)}, \{\!|c, Ta|\!\}_{K_{ab}}) \cdot \mathsf{send}(\{\!|Ta|\!\}_{K_{ab}}) \cdot \mathsf{Secret} \\[1em]
\mathsf{K5}(S) &= \mathsf{rcv}(A, B, Na) \cdot \mathsf{send}(\{\!|B, ts, Na, k_{ab}|\!\}_{\mathsf{sh}(A,S)}, \{\!|A, ts, k_{ab}|\!\}_{\mathsf{sh}(B,S)})
\end{aligned}$$

Transforming K5 **into** K3

Note that K5 is well-formed. To transform K5 to K3, we first remove variable X from the messages of role A by applying the atom-and-variable removal abstraction rem_{T_1} where $T_1 = \{X\}$. The resulting protocol $\mathsf{K3'} = rem_{T_1}(\mathsf{K3})$ is given as follows.

$$\begin{aligned}
\mathsf{K3'}(A) &= \mathsf{send}(A, B, na) \cdot \mathsf{rcv}(\{\!|B, Ts, na, K_{ab}|\!\}_{\mathsf{sh}(A,S)}) \cdot \\
&\quad\ \mathsf{send}(\{\!|c, ta|\!\}_{k_{ab}}) \cdot \mathsf{rcv}(\{\!|ta|\!\}_{K_{ab}}) \cdot \mathsf{Secret} \\[1em]
\mathsf{K3'}(B) &= \mathsf{rcv}(\{\!|A, Ts, K_{ab}|\!\}_{\mathsf{sh}(B,S)}, \{\!|c, Ta|\!\}_{K_{ab}}) \cdot \mathsf{send}(\{\!|Ta|\!\}_{K_{ab}}) \cdot \mathsf{Secret} \\[1em]
\mathsf{K3'}(S) &= \mathsf{rcv}(A, B, Na) \cdot \mathsf{send}(\{\!|B, ts, Na, k_{ab}|\!\}_{\mathsf{sh}(A,S)}, \{\!|A, ts, k_{ab}|\!\}_{\mathsf{sh}(B,S)})
\end{aligned}$$

We can now transform K3′ to K3 by applying a splitting abstraction. In particular, we split the last event of role S and the first event of role B into two events each. Let f_{sp}

be the splitting abstraction such that

$$\begin{aligned}
&f_{sp}(\mathsf{rcv}(\{\!|A, Ts, K_{ab}|\!\}_{\mathsf{sh}(B,S)}, \{\!|c, Ta|\!\}_{k_{ab}}))\\
&= \mathsf{rcv}(\{\!|A, Ts, K_{ab}|\!\}_{\mathsf{sh}(B,S)}) \cdot \mathsf{rcv}(\{\!|c, Ta|\!\}_{K_{ab}})
\end{aligned}$$

$$\begin{aligned}
&f_{sp}(\mathsf{send}(\{\!|B, ts, Na, k_{ab}|\!\}_{\mathsf{sh}(A,S)}, \{\!|A, ts, k_{ab}|\!\}_{\mathsf{sh}(B,S)}))\\
&= \mathsf{send}(\{\!|B, ts, Na, k_{ab}|\!\}_{\mathsf{sh}(A,S)}) \cdot \mathsf{send}(\{\!|A, ts, k_{ab}|\!\}_{\mathsf{sh}(B,S)})
\end{aligned}$$

We obtain $\mathsf{K3} = f_{sp}(\mathsf{K3}')$ as desired.

$$\begin{aligned}
\mathsf{K3}(A) &= \mathsf{send}(A, B, na) \cdot \mathsf{rcv}(\{\!|B, Ts, na, K_{ab}|\!\}_{\mathsf{sh}(A,S)}) \cdot\\
&\quad\ \mathsf{send}(\{\!|c, ta|\!\}_{K_{ab}}) \cdot \mathsf{rcv}(\{\!|ta|\!\}_{K_{ab}}) \cdot \mathsf{Secret}\\
\mathsf{K3}(B) &= \mathsf{rcv}(\{\!|A, Ts, K_{ab}|\!\}_{\mathsf{sh}(B,S)}) \cdot \mathsf{rcv}(\{\!|c, Ta|\!\}_{K_{ab}}) \cdot \mathsf{send}(\{\!|Ta|\!\}_{K_{ab}}) \cdot \mathsf{Secret}\\
\mathsf{K3}(S) &= \mathsf{rcv}(A, B, Na) \cdot \mathsf{send}(\{\!|B, ts, Na, k_{ab}|\!\}_{\mathsf{sh}(A,S)}) \cdot \mathsf{send}(\{\!|A, ts, k_{ab}|\!\}_{\mathsf{sh}(B,S)})
\end{aligned}$$

Transforming K3 into K2

In this transformation, we pull all nonces and timestamps out of the encryptions containing them, and remove the encryptions with the session key. We therefore apply a typed abstraction $F_{f_2} = (f_2, E_{f_2})$ where E_{f_2} consists of the following clauses with variables X_1, Y_1: α, Z_1: γ_c, X_2, Y_2, Z_2, Z_3: $\mathsf{TimeStamp}$, X_3: Nonce, X_4, Y_3, Z_4, Z_5, K : msg.

$$\begin{aligned}
f_2(\{\!|X_1, X_2, X_3, X_4|\!\}_K) &= \langle \{\!|f_2(X_1), f_2(X_4)|\!\}_{f_2(K)}, f_2(X_2), f_2(X_3)\rangle\\
f_2(\{\!|Y_1, Y_2, Y_3|\!\}_K) &= \langle \{\!|f_2(Y_1), f_2(Y_3)|\!\}_{f_2(K)}, f_2(Y_2)\rangle\\
f_2(\{\!|Z_1, Z_2|\!\}_K) &= \langle f_2(Z_1), f_2(Z_2)\rangle\\
f_2(\{\!|Z_3|\!\}_K) &= f_2(Z_3)\\
f_2(\langle Z_4, Z_5\rangle) &= \langle f_2(Z_4), f_2(Z_5)\rangle
\end{aligned}$$

The first clause is used to pull the nonces and timestamps out of the second message of role A and the second message of role S. The next clause pulls the timestamps out of the last message of role S and the first message of role B. The third and the fourth clauses remove encryptions from the last two messages of role A and role B. As before, the last clause deals with pairs. We obtain the protocol $\mathsf{K2} = f_2(\mathsf{K3})$.

$$\begin{aligned}
\mathsf{K2}(A) &= \mathsf{send}(A, B, na) \cdot \mathsf{rcv}(\{\!|B, K_{ab}|\!\}_{\mathsf{sh}(A,S)}, Ts, na) \cdot\\
&\quad\ \mathsf{send}(c, ta) \cdot \mathsf{rcv}(ta) \cdot \mathsf{Secret}\\
\mathsf{K2}(B) &= \mathsf{rcv}(\{\!|A, K_{ab}|\!\}_{\mathsf{sh}(B,S)}, Ts) \cdot \mathsf{rcv}(c, Ta) \cdot \mathsf{send}(Ta) \cdot \mathsf{Secret}\\
\mathsf{K2}(S) &= \mathsf{rcv}(A, B, Na) \cdot \mathsf{send}(\{\!|B, k_{ab}|\!\}_{\mathsf{sh}(A,S)}, ts, Na) \cdot \mathsf{send}(\{\!|A, k_{ab}|\!\}_{\mathsf{sh}(B,S)}, ts)
\end{aligned}$$

Transforming K2 into K1

In this last abstraction, we remove all constants, timestamps, and nonces that are in unprotected positions. We therefore apply the atom-and-variable removal abstraction

rem_{T_2} where $T_2 = \{na, Na, ta, Ta, ts, Ts, c\}$. We obtain $\mathsf{K1} = rem_{T_2}(\mathsf{K2})$.

$$
\begin{aligned}
\mathsf{K1}(A) &= \mathsf{send}(A, B) \cdot \mathsf{rcv}(\{\!|B, k_{ab}|\!\}_{\mathsf{sh}(A,S)}) \cdot \mathsf{Secret} \\
\mathsf{K1}(B) &= \mathsf{rcv}(\{\!|A, k_{ab}|\!\}_{\mathsf{sh}(B,S)}) \cdot \mathsf{Secret} \\
\mathsf{K1}(S) &= \mathsf{rcv}(A, B) \cdot \mathsf{send}(\{\!|B, k_{ab}|\!\}_{\mathsf{sh}(A,S)}) \cdot \mathsf{send}(\{\!|A, k_{ab}|\!\}_{\mathsf{sh}(B,S)})
\end{aligned}
$$

Verifying the properties

Note that these abstractions do not change ϕ_s^A and ϕ_s^B. Moreover, since we do not consider authentication properties, it is easy to see that the abstractions from K5 to K3 and from K2 to K1 are sound. In Appendix E.2, we justify the soundness of all abstraction steps from K4 to K5 and from K3 to K2. As K1 satisfies ϕ_s^A and ϕ_s^B, we conclude that the original protocol K5 also satisfies these properties.

Note that the abstractions we present above are just an example. They are chosen to maximally simplify the protocol. There are of course many other possible abstractions supported by our theory. In Chapter 6, we present experimental results for the KSL protocol which is similar to the K5 protocol in our example, but much more complex. These experimental results show impressive speedups and thus illustrate the usefulness of our protocol abstractions and abstraction heuristics.

Part II.

Implementation and Experimental Results

5. Implementation

In this chapter, we present an algorithm that automatically simplifies protocols. Our algorithm combines both typed and untyped abstractions in order to extensively abstract a given protocol specification. These abstractions are generated automatically using heuristics which we describe in Section 5.1. In Section 5.2, we present an implementation of our abstraction methodology for the Scyther tool.

5.1. Abstraction Heuristics

In Figure 5.1, we depict the abstraction workflow for the analysis of security protocols. The rectangles denote functionalities, the rounded rectangles denote input/output, and the diamonds denote a decision. Outgoing edges present the outcomes of computations, while incoming edges present the data used for computations. The abstraction module takes as an input a protocol specification together with its intended security properties. To analyze a security protocol P against security properties ϕ, the abstraction module performs the following steps:

Step 1 builds a stack of successively more abstract protocols and properties using Algorithm 1.

Step 2 pops a model P' and property ϕ' from the stack and feeds the model to the verifier.

Step 3 examines the verification result. There are two possibilities:

- If there is no attack then it concludes that the original P satisfies ϕ by Theorems 4, 5, and 6.
- If an attack on ϕ' is found then it checks whether the stack is empty or not.
  - If the stack is empty then the model being analyzed is the original protocol P. It therefore concludes that P does not satisfy ϕ.
  - If the stack is not empty then it goes back to Step 2.

The construction of abstract models in Step 1 is guided by a heuristic. In particular, the abstraction generator computes a series of successively more abstract protocols in several abstraction steps. Each abstraction step consists of a typed abstraction followed by an atom-and-variable removal and a redundancy removal abstraction. First, the typed abstraction simplifies the topmost cryptographic operations in the original protocol messages. This is done by pulling terms that are not essential to the security properties out of the scope of the cryptographic operators, or removing them completely. Determining

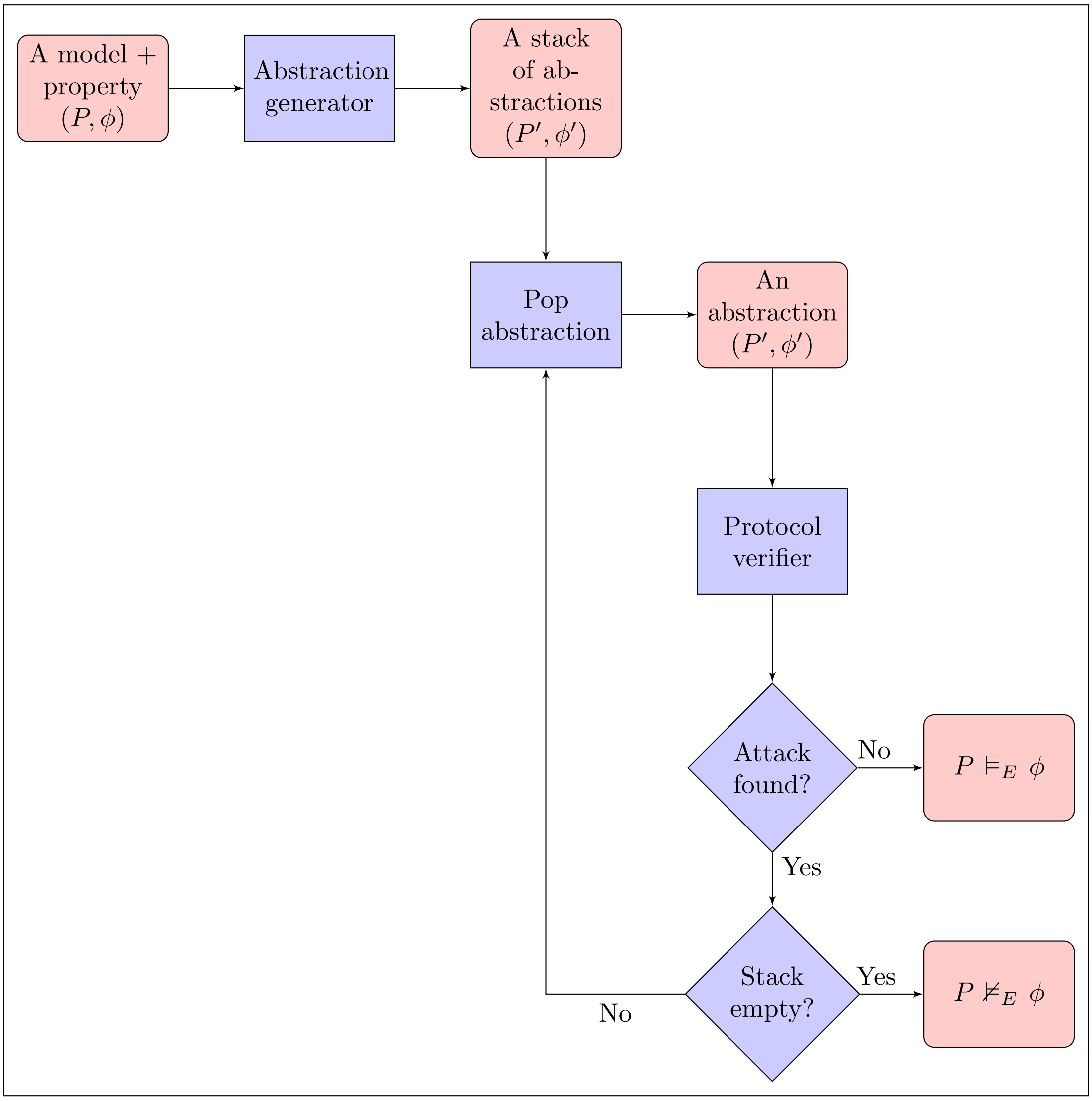

Figure 5.1.: The workflow of protocol analysis combining the abstraction mechanism and protocol verification tools.

whether a term can be pulled out or removed is guided by a heuristic which we will describe in more detail in Subsection 5.1.1. Second, the atom-and-variable removal abstraction eliminates unprotected atoms and variables that do not occur in the security properties. In Subsection 5.1.2, we present an algorithm that generates this abstraction. Finally, the redundancy removal abstraction gets rid of intruder-deducible terms in the protocol messages. These terms are typically constants or replicated messages. We refer to Subsection 5.1.3 for a detailed description of how this abstraction is computed.

The computation of the abstraction series is described in Algorithm 1. This algorithm

Algorithm 1 Algorithm for computing abstract models.

`Input:` a protocol specification P and a property formula ϕ.

`Output:` a stack S of abstract models of P.

1: $S \leftarrow [(P, \phi)]$
2: `repeat`
3: $\quad (P', \phi') \leftarrow top(S)$
4: $\quad (P_1, \phi_1) \leftarrow typedAbstract(P', \phi')$
5: $\quad (P_2, \phi_2) \leftarrow removeAV(P_1, \phi_1)$
6: $\quad (P_3, \phi_3) \leftarrow removeRedundancy(P_2, \phi_2)$
7: $\quad$ `if` $P_3 \neq P$ `then`
8: $\qquad push(S, (P_3, \phi_3))$
9: $\quad$ `endif`
10: `until` $P_3 = P$ `or` $|S| \geq LIMIT$
11: `return` S

takes as an input a protocol specification P and a property formula ϕ, and returns a stack that contains different abstract models of P. Initially, the stack only contains the original protocol P and security properties ϕ (line 1). The loop (lines 2-10) computes different abstractions of P and ϕ. The condition in line 7 checks whether the abstract protocol is different from the considered original protocol. If it is the case then the abstract protocol and the corresponding abstract security properties are pushed onto the stack (line 8). The loop terminates when no simplification is produced or the number of abstract models exceeds a given limit, e.g., $LIMIT = 10$.

5.1.1. Generating a Typed Abstraction

In order to generate a typed abstraction, we use a heuristic to decide whether a term should be removed, pulled out of a cryptographic operator, or kept. Central to our heuristic are the notions of *security labels* and *essential terms*. Intuitively, a security label associated with a given term is an approximation of the security guarantees for that term. This approximation relies on the security properties of interest and the intended security properties of cryptographic primitives. For example, public-key encryptions provide confidentiality and MACs (with long-term keys) provide both confidentiality and authentication. We now formally define the set of terms that are essential for a given security property.

Definition 29 (Essential terms)**.** Let ϕ be a security property. We say that a term t is essential of ϕ if $t \in subs(Sec_\phi \cup EqTerm_\phi)$.

Intuitively, essential terms are (sub)terms of the terms occurring in security properties. In many cases, the security of the essential terms is required in order to achieve the property ϕ. For example, if ϕ expresses the secrecy of hash $\mathsf{h}(u)$ for a term u then the satisfaction of ϕ implies the secrecy of u.

In the following, we first define the notion of *security labels*. Then, we explain how to generate a typed abstraction based on these labels.

Security labels

We define a set of confidentiality labels *Conf* and a set of authenticity labels *Auth* as follows.

$$\begin{array}{rcl} \mathit{Conf} & = & \{\mathsf{NO\_CONF}, \mathsf{MAYBE\_CONF}, \mathsf{CONF}\} \\ \mathit{Auth} & = & \{\mathsf{NO\_AUTH}, \mathsf{MAYBE\_AUTH}, \mathsf{AUTH}\} \end{array}$$

with a total order $\leq_{lb}$ over these sets such that:

$$\begin{array}{ccccc} \mathsf{NO\_CONF} & \leq_{lb} & \mathsf{MAYBE\_CONF} & \leq_{lb} & \mathsf{CONF} \\ \mathsf{NO\_AUTH} & \leq_{lb} & \mathsf{MAYBE\_AUTH} & \leq_{lb} & \mathsf{AUTH} \end{array}$$

Intuitively, *Conf* and *Auth* are ordered by the strength of the security guarantees that are likely to hold. For example, $\mathsf{NO\_CONF}$ means that no secrecy is provided, while $\mathsf{MAYBE\_CONF}$ expresses that secrecy may be achieved. Finally, CONF denotes our strongest belief that secrecy holds. *A security label* is a pair (c, a) where $\mathsf{c} \in \mathit{Conf}$ and $\mathsf{a} \in \mathit{Auth}$. We extend the order $\leq_{lb}$ to pairs of security labels by $(\mathsf{c}, \mathsf{a}) \leq_{lb} (\mathsf{c}', \mathsf{a}')$ if and only if $\mathsf{c} \leq_{lb} \mathsf{c}'$ and $\mathsf{a} \leq_{lb} \mathsf{a}'$. For a set of security labels *lbs*, we use $lub(lbs)$ and $glb(lbs)$ to denote the least upper-bound and the greatest lower-bound of all elements in *lbs*.

We distinguish three types of security labels that can be assigned to terms, namely *protocol labels*, *property labels*, and *global labels*. Informally, the protocol label of a given term t approximates the strongest security guarantees we can achieve for t. In contrast, the property label of t approximates the security guarantees we wish to provide for t. The global label of t is a compromise between t's protocol label and property label.

Protocol labels In order to approximate the security guarantees that the protocol achieves for a given term, we rely on the intended security properties provided by each cryptographic primitive. In Table 5.1, we present different cryptographic operations and the corresponding security guarantees they provide. Note that we assign security labels $(\mathsf{MAYBE\_CONF}, \mathsf{MAYBE\_AUTH})$ and $(\mathsf{CONF}, \mathsf{MAYBE\_AUTH})$ to symmetric encryptions and MACs with session keys respectively, since the security of session keys is protocol-dependent. Recall that our signature scheme allows message recovery. Hence signatures provide no secrecy. For a given term t, we use $\ell_1(t)$ to denote the security label associated with t based on Table 5.1. We define a function $\ell_2 \colon \mathcal{T} \times \mathbb{N}^* \to \mathit{Conf} \times \mathit{Auth}$ such that

(i) $\ell_2(t, \epsilon) = (\mathsf{NO\_CONF}, \mathsf{NO\_AUTH})$,

Cryptographic operations	Confidentiality	Authenticity
MACs with long-term keys	CONF	AUTH
MACs with session keys	CONF	MAYBE_AUTH
symmetric encryptions with long-term keys	CONF	AUTH
symmetric encryptions with session keys	MAYBE_CONF	MAYBE_AUTH
public-key encryptions	CONF	NO_AUTH
hashes	CONF	NO_AUTH
signatures	NO_CONF	AUTH
other constructors	NO_CONF	NO_AUTH

Table 5.1.: Security labels that might be achieved by different cryptographic operations.

(ii) $\ell_2(t, p \cdot i) = \begin{cases} lub(\{\ell_2(t,p), \ell_1(t|_p)\}) & \text{if } p \cdot i \notin \mathcal{P}os(t) \text{ and } IK_0 \cup \mathcal{V}_\alpha \nvdash_E t|_{p\cdot i} \\ (\mathsf{NO\_CONF}, \mathsf{NO\_AUTH}) & \text{otherwise} \end{cases}$

Intuitively, $\ell_2(t,p)$ approximates the security guarantees that t provides for its subterm $t\,|_p$. These security guarantees are the least upper bound of those provided by the cryptographic operations lying on the path from the root of t to the position p. Of course, t provides no security guarantees for terms that are not its strict subterms. We also decide that terms that are deducible from the intruder's knowledge, e.g., constants or agent names, will get no security guarantees. This allows us to remove these terms and therefore enables more abstractions.

Example 32. The term $[h(u)]_{\mathsf{pri}(A)}$ provides $(\mathsf{CONF}, \mathsf{AUTH})$ for its subterm u given that $IK_0 \nvdash_E u$, because the signature provides $(\mathsf{NO\_CONF}, \mathsf{AUTH})$ and the hash provides $(\mathsf{CONF}, \mathsf{NO\_AUTH})$. Then by taking the least upper-bound of these labels, we obtain $(\mathsf{CONF}, \mathsf{AUTH})$ which is the approximated security guarantee for u. ♠

Note that ℓ_2 only computes security labels for subterms at particular positions. However, a term t may contain several occurrences of a subterm u. In order to approximate security guarantees that t provides for u, we need to combine all of the individual approximations. Let $Pos_t(u)$ be the set of all positions of t at which u occurs. If $Pos_t(u) = \emptyset$ then the security label of u provided by t is $(\mathsf{NO\_CONF}, \mathsf{NO\_AUTH})$. Otherwise, we have $Pos_t(u) = \{p_1, \ldots, p_n\}$ and $\ell_2(t, p_i) = (\mathsf{c}_i, \mathsf{a}_i)$ for $i \in \widetilde{n}$. We define the security label of u provided by t as $(min\{\mathsf{c}_1, \ldots, \mathsf{c}_n\}, max\{\mathsf{a}_1, \ldots, \mathsf{a}_n\})$. The intuition behind this definition is that confidentiality needs to be ensured in all positions, while authenticity only needs to be achieved in some position.

We now give the formal definition of *protocol labels*.

Definition 30 (Protocol labels)**.** Suppose that $\mathcal{M}_P = \{t_1, \ldots, t_n\}$ and t is a term. Let $(\mathsf{c}_i, \mathsf{a}_i)$ be the security label of t provided by t_i for $i \in \widetilde{n}$. The *protocol label of* t, written as $\ell_P(t)$, is defined as $(min\{\mathsf{c}_1, \ldots, \mathsf{c}_n\}, max\{\mathsf{a}_1, \ldots, \mathsf{a}_n\}\})$.

Intuitively, the protocol label of a given term is the security guarantees for that term provided by the protocol terms. In the following example, we explain how these labels are computed.

Example 33. Suppose that our protocol P's specifications is given as follows.

$$\begin{array}{rcl} P(A) & = & \mathsf{send}(na) \cdot \mathsf{rcv}(\mathsf{mac}(\mathsf{sh}(A,B), na)) \\ P(B) & = & \mathsf{rcv}(Na) \cdot \mathsf{send}(\mathsf{mac}(\mathsf{sh}(B,A), Na)) \end{array}$$

In this protocol, the initiator in role A generates a nonce na and sends it to the responder in role B. The responder authenticates himself to the initiator by sending a MAC including the nonce he has received.

Suppose that we want to compute the protocol label of the nonce na. There are only two protocol messages that contain na as subterms, namely na and $\mathsf{mac}(\mathsf{sh}(A,B), na)$. Clearly, na provides no security guarantees for itself, hence we obtain security label $(\mathsf{NO\_CONF}, \mathsf{NO\_AUTH})$ for na from the first message. According to Table 5.1, we know that $\ell_1(\mathsf{mac}(\mathsf{sh}(A,B), na)) = (\mathsf{CONF}, \mathsf{AUTH})$. Therefore, we obtain $(\mathsf{CONF}, \mathsf{AUTH})$ for na from the second message. Hence, the protocol label of na is $(\mathsf{NO\_CONF}, \mathsf{AUTH})$. ♠

Property labels While protocol labels stem from the protocol terms, property labels are derived from the security properties of interest. We formally define these labels as follows.

Definition 31 (Property labels)**.** Let t be a term. The *property label of* t, written as $\ell_\phi(t)$, is defined as

$$\ell_\phi(t) = \begin{cases} (\mathsf{CONF}, \mathsf{NO\_AUTH}) & \text{if } t \in \mathit{subs}(\mathit{Sec}_\phi) \setminus \mathit{subs}(\mathit{EqTerm}_\phi) \\ (\mathsf{NO\_CONF}, \mathsf{AUTH}) & t \in \mathit{subs}(\mathit{EqTerm}_\phi) \setminus \mathit{subs}(\mathit{Sec}_\phi) \\ (\mathsf{CONF}, \mathsf{AUTH}) & t \in \mathit{subs}(\mathit{Sec}_\phi) \cap \mathit{subs}(\mathit{EqTerm}_\phi) \\ (\mathsf{NO\_CONF}, \mathsf{NO\_AUTH}) & \text{otherwise} \end{cases}$$

Intuitively, the property label of a given term t is the security guarantees for t that are potentially needed for the security properties of interest to hold. For example, if ϕ expresses the secrecy of $\mathsf{h}(u)$ then the property label of both $\mathsf{h}(u)$ and u is $(\mathsf{CONF}, \mathsf{NO\_AUTH})$. Note that the secrecy of u is required to achieve the secrecy of $\mathsf{h}(u)$.

Global labels In general, $\ell_P(t)$ and $\ell_\phi(t)$ may not agree with each other for a given term t. We therefore define *global labels* which are uniquely determined for each term.

Definition 32 (Global labels)**.** Let t be an essential term of ϕ. We define the *global label of* t, written as $\ell_{P,\phi}(t)$, by $\mathit{glb}(\{\ell_P(t), \ell_\phi(t)\})$.

Note that the global label of a given term is defined as the greatest lower-bound of the term's protocol label and property label. This ensures that global labels (i) do not over-estimate the security properties of the protocol and (ii) capture security properties which are not stronger than those we want to achieve. These points make global labels suitable for our heuristic. In practice, we often have that $\ell_\phi(t) \leq_{lb} \ell_P(t)$, i.e., P provides stronger security guarantees than those expressed by the security property. In this case, property labels and global labels of essential terms coincide.

Algorithm for generating a typed abstraction

In order to generate a typed abstraction $F_f = (f, E_f)$, the abstraction generator first determines an abstraction for each original protocol message. Then, it generates a clause that corresponds to the abstraction and adds the clause to E_f. It also resolves any violation of pattern-disjointness that may occur. Finally, it completes the typed abstraction by adding missing clauses to E_f. These clauses handle all the protocol and property subterms that are not transformed by the initial clauses. Thereby, it ensures that $\mathcal{M}_P \cup Sec_\phi \cup EqTerm_\phi \subseteq udom(F_f)$ which is part of the soundness conditions. As soon as the abstraction has been computed, the abstraction generator checks whether the abstraction satisfies the remaining soundness conditions as pattern-disjointness is guaranteed at this point. If it is the case then the abstraction generator computes the corresponding abstract model. Otherwise, it returns the original one. The computations of a typed abstraction and the corresponding abstract model are depicted in Algorithm 2.

Algorithm 2 Implementation of *typedAbstract*.

`Input:` a protocol specification P and property formula ϕ.
`Output:` abstract protocol P' and property ϕ'.

$E_f \leftarrow []$
for all $t \in split(\mathcal{M}_P)$
  $E_f \leftarrow genClauses(t, P, E_f, \phi)$
endfor
$completeClauses(E_f)$
if $safeTypedAbstraction(P, \phi, E_f)$ **then**
  return $(f(P), f(\phi))$
else return (P, ϕ)
endif

In the following paragraphs, we describe in more detail how to generate clauses for a typed abstraction.

Generating clauses for a typed abstraction For a given protocol term, the abstraction generator decides based on global labels which fields are pulled outside of or removed from the topmost cryptographic operation of the term. The main criterion is that these abstractions must preserve the global labels of essential terms in order to avoid oversimplifying protocols, hence false negatives. The successive abstractions work from the outside to the inside of the term. More precisely, if the abstraction generator fails to simplify the outermost constructor of a given term t then it goes into the immediate subterms of t. Note that the user can also provide a typed abstraction specification of his choice. This specification does not need to be complete.

Algorithm 3 describes how to determine a transformation for a given term t and to generate a clause that transforms the term accordingly. In this algorithm, we represent a clause $(f(p) = q)$ by (p, q). Additionally, we introduce functions *removeElem*, *protAbst*, *propAbst*, *termAbst*, *genOneClause*, and *updateClause* whose purposes are given as follows.

removeElem(l, i) returns the list that is obtained by removing the i-th element from the list l.

protAbst(P, p, q), *propAbst*(ϕ, p, q), and *termAbst*(t, p, q) transform the protocol P, the property ϕ, and the term t using $E_f = [f(p) = q]$, respectively.

genOneClause$(t, \mathit{pull}, \mathit{keep})$ generates a clause (p, q) that transforms the composed term t of the form $c(t_1, \ldots, t_n)$. List *pull* contains fields of t_i for all $i \in \widetilde{n}$ that should be pulled out of t's top function symbol. Array *keep* maps an index $i \in \widetilde{n}$ and returns the list of fields of t_i that should be kept. Note that if all the fields are pulled out then the constructor c is dropped. In Example 34, we illustrate how to generate such clauses.

updateClause(E_f, p, q) adds the clause (p, q) to list E_f and returns the resulting list. Note that whenever a clause is added to E_f, we need to ensure that the clause does not create a violation of pattern-disjointness. Therefore in the implementation of *updateClause*, we check whether the clause overlaps with some other clauses in E_f or not. If it is the case, we merge any two clauses that violate the pattern-disjointness into a new one as follows. Suppose that there are two clauses $f(p_1) = q_1$ and $f(p_2) = q_2$ such that $\Gamma(p_1)\!\downarrow \cap\, \Gamma(p_2)\!\downarrow \neq \emptyset$. Let τ be the least supertype of $\Gamma(p_1)$ and $\Gamma(p_2)$. We select one of these clauses, i.e., $f(p_k) = q_k$ for some $k \in \{1, 2\}$ and adapt the type of p_k to τ. This ensures that any term that can be transformed by one of these clauses will also be transformed by the new clause. We must be careful because not any k will work. For example, if $p_1 = h(g(X))$ and $p_2 = h(Y)$ for $X, Y : \mathsf{Nonce}$ then $k = 1$ does not work, because p_1 is deeper than p_2 and thus we cannot update the type of $h(g(X))$ to $h(\mathsf{msg})$. Our selection is therefore defined as follows.

(i) If a pattern is strictly shallower than the other then the clause corresponding to the shallower pattern is selected.

(ii) Otherwise, if one of these clauses is defined by users then that clause is selected. Here, we assume that user-defined clauses are pattern-disjoint. Note that this is also checked by our abstraction generator.

(iii) Otherwise, the clause that makes less changes on terms than the other is selected. We define the amount of changes a clause makes as the ratio of the number of fields the clause removes or pulls out to the size of the clause. The intuition behind this rule is that the selected clause has less chance of oversimplifying the protocol.

If p_1 and p_2 are comparable with respect to shallowness then updating the type of p_k to τ can always be done by adapting the types of the variables of p_k accordingly. This is because τ is always deeper than p_k in this case. Otherwise, the type updating fails and no abstraction is produced.

Algorithm 3 Implementation of *genClauses*.

Input: a term t, a protocol P, a list of clauses E_f, and a formula ϕ, where $t = c(t_1, \ldots, t_n)$ for $c \in \Sigma^n$ and $t_i = \langle t_{i,1}, \ldots, t_{i,a_i} \rangle$.

Output: a list of clauses.

if $c = \langle \cdot, \cdot \rangle$ **then**
  $E_f \leftarrow genClauses(t_1, P, E_f, \phi)$
  $E_f \leftarrow genClauses(t_2, P, E_f, \phi)$
  return E_f
endif
if $c \in \mathsf{C_R}$ **then**
  $m \leftarrow n - 1; \quad keep[n] \leftarrow [t_n]$
else $m \leftarrow n$
endif
$pull \leftarrow []$
for $i = 1$ **to** m
  $keep[i] \leftarrow [t_{i,1}, \ldots, t_{i,a_i}]$
  for $j = 1$ **to** a_i
    let $keep'[k] = keep[k]$, $\forall k \in \widetilde{m} \setminus \{i\}$ and $keep'[i] = removeElem(keep[i], j)$
    if $i \notin pp(c)$ **then**
      $(p, q) \leftarrow genOneClause(t, pull, keep')$
      $P' \leftarrow protAbst(P, p, q)$
      $\phi' \leftarrow propAbst(\phi, p, q)$
      if $\forall u \in subs(Sec_\phi \cup EqTerm_\phi).\ \ell_{P,\phi}(u) \leq_{lb} \ell_{P',\phi'}(termAbst(u, p, q))$ **then**
        $keep[i] \leftarrow removeElem(keep[i], j)$
      endif
    endif
    if $|keep[i]| = a_i$ **then**
      $(p, q) \leftarrow genOneClause(t, pull \cdot [t_{i,j}], keep')$
      $P' \leftarrow protAbst(P, p, q)$
      $\phi' \leftarrow propAbst(\phi, p, q)$
      if $\forall u \in subs(Sec_\phi \cup EqTerm_\phi).\ \ell_{P,\phi}(u) \leq_{lb} \ell_{P',\phi'}(termAbst(u, p, q))$ **then**
        $pull \leftarrow pull \cdot [t_{i,j}]; \quad keep[i] \leftarrow removeElem(keep[i], j)$
      endif
    endif
  endfor
endfor
if $|keep[i]| = a_i$ for all $i \in \widetilde{m}$ **then**
  for $i = 1$ **to** m
    $E_f \leftarrow genClauses(t_i, P, E_f, \phi)$
  endfor
  return E_f
endif
$(p, q) \leftarrow genOneClause(t, pull, keep)$
return $updateClause(E_f, p, q)$

In order to determine a transformation for a given term t, we first check whether t is a pair or not (line 1). If t is a pair then we recursively call the procedure for its fields as we are not interested in reordering pairs at this point. Line 6 checks whether $c \in \mathsf{C_R}$. If $c \in \mathsf{C_R}$ then we are only allowed to manipulate t_i's fields for $1 \leq i \leq n-1$, and we cannot interfere with the last argument t_n as it is at the key position. In this case, we set m to $n-1$ and $keep[n]$ to the singleton list $[t_n]$. Otherwise, we set m to n since we do not have these restrictions. Initially, $keep[i]$ contains all fields of the argument t_i for all $i \in \widetilde{m}$ (line 12). The outermost loop (lines 11-32) determines which fields of t_i are pulled out, removed, or kept. In particular, lines 16-20 attempt to remove the field $t_{i,j}$ if i is not an extractable position of c. The condition in line 19 checks whether removing $t_{i,j}$ preserves global labels of essential terms. In the case that the preservation holds, line 20 removes $t_{i,j}$ from the keeping list $keep[i]$. If no removal is possible, lines 24-28 try to pull $t_{i,j}$ out the the cryptographic operation. Similarly, line 27 checks whether pulling out $t_{i,j}$ preserves global labels of essential terms. If it is the case then $t_{i,j}$ is pulled out (line 28). Lines 34-36 recursively go into t's immediate subterms if the outermost constructor c cannot be simplified. When the abstraction is completely determined, we generate the final clause (line 39) and add this clause to E_f using *updateClause* (line 40).

Completing a typed abstraction To complete a typed abstraction, we need to ensure that the transformations of the protocol and property (sub)terms only require clauses in E_f. We therefore run a test that transforms P and ϕ. Whenever a term does not have a corresponding clause in E_f, we add to E_f the homomorphic clause for that term using *updateClause*. After the completion, we ensure that E_f is pattern-disjoint and the condition that $\mathcal{M}_P \cup Sec_\phi \cup EqTerm_\phi \subseteq udom(F_f)$ is satisfied. Let us illustrate how the abstraction generator generates a typed abstraction in the following example.

Example 34. We compute the first abstraction for the $\mathsf{IKE_m}$ protocol from Example 7. In this abstraction, we consider the following messages for role A

$$\begin{array}{lcl} \mathsf{M1} & : & \{\!|A, B, AUTHaa, sA2, tSa, tSb|\!\}_{SKa} \\ \mathsf{M2} & : & \{\!|B, AUTHba, sA2, tSa, tSb|\!\}_{SKa} \end{array}$$

and the corresponding two messages for role B

$$\begin{array}{lcl} \mathsf{M3} & : & \{\!|A, B, AUTHab, sA2, tSa, tSb|\!\}_{SKb} \\ \mathsf{M4} & : & \{\!|B, AUTHbb, sA2, tSa, tSb|\!\}_{SKb} \end{array}$$

where the authenticators and session keys are recalled below.

$$\begin{array}{rcl} SKa & = & \mathsf{kdf}(\underline{na}, \underline{Nb}, \underline{\mathsf{exp}(Gb, x)}, sPIa, SPIb), \\ SKb & = & \mathsf{kdf}(\underline{Na}, \underline{nb}, \underline{\mathsf{exp}(Ga, y)}, SPIa, sPIb), \\ AUTHaa & = & \mathsf{mac}(\mathsf{sh}(A, B), sPIa, o, sA1, \underline{\mathsf{exp}(g, x)}, \underline{na}, \underline{Nb}, \mathsf{prf}(SKa, A)), \\ AUTHab & = & \mathsf{mac}(\mathsf{sh}(B, A), SPIa, o, sA1, \underline{Ga}, \underline{Na}, \underline{nb}, \mathsf{prf}(SKb, A)), \\ AUTHba & = & \mathsf{mac}(\mathsf{sh}(A, B), sPIa, SPIb, sA1, \underline{Gb}, \underline{Nb}, \underline{na}, \mathsf{prf}(SKa, B)), \\ AUTHbb & = & \mathsf{mac}(\mathsf{sh}(B, A), SPIa, sPIb, sA1, \underline{\mathsf{exp}(g, y)}, \underline{nb}, \underline{Na}, \mathsf{prf}(SKb, B)). \end{array}$$

We consider the security property ϕ_a specified in Example 8. The essential terms are underlined. In the first abstraction step, the algorithm removes encryptions from M1-M4 since the global labels of essential terms are preserved by the mac. We therefore obtain the following transformation.

$$\begin{array}{rcl}
\{\!|A, B, AUTHaa, sA2, tSa, tSb|\!\}_{SKa} & \mapsto & \langle A, B, AUTHaa, sA2, tSa, tSb\rangle \\
\{\!|B, AUTHba, sA2, tSa, tSb|\!\}_{SKa} & \mapsto & \langle B, AUTHba, sA2, tSa, tSb\rangle \\
\{\!|A, B, AUTHab, sA2, tSa, tSb|\!\}_{SKb} & \mapsto & \langle A, B, AUTHab, sA2, tSa, tSb\rangle \\
\{\!|B, AUTHbb, sA2, tSa, tSb|\!\}_{SKb} & \mapsto & \langle B, AUTHbb, sA2, tSa, tSb\rangle
\end{array}$$

In order to generate a clause that corresponds to this transformation, we replace the session keys and each field in the plaintexts of the encryptions with fresh variables. By applying this replacement on both sides of the transformation above, we derive the desired clauses which are given by

$$\begin{array}{rcl}
f(\{\!|X_1, X_2, X_3, X_4, X_5, X_6|\!\}_{X_7}) & = & \langle f(X_1), f(X_2), f(X_3), f(X_4), f(X_5), f(X_6)\rangle, \\
f(\{\!|Y_1, Y_2, X_3, Y_4, Y_5|\!\}_{Y_6}) & = & \langle f(Y_1), f(Y_2), f(Y_3), f(Y_4), f(Y_5)\rangle, \\
f(\{\!|Z_1, Z_2, Z_3, Z_4, Z_5, Z_6|\!\}_{Z_7}) & = & \langle f(Z_1), f(Z_2), f(Z_3), f(Z_4), f(Z_5), f(Z_6)\rangle, \\
f(\{\!|U_1, U_2, U_3, U_4, U_5|\!\}_{U_6}) & = & \langle f(U_1), f(U_2), f(U_3), f(U_4), f(U_5)\rangle.
\end{array}$$

with variables and their corresponding types as follows.

$$\begin{array}{l}
X_1, X_2, Y_1, Z_1, Z_2, U_1 : \alpha \\
X_3 : \mathsf{mac}(\mathsf{sh}(\alpha, \alpha), \beta_{sPIa}, \gamma_o, \gamma_{sA1}, \mathsf{exp}(\gamma_g, \gamma_x), \beta_{na}, \mathsf{Nonce}, \\
\qquad \mathsf{prf}(\mathsf{kdf}(\beta_{na}, \mathsf{Nonce}, \mathsf{exp}(\mathsf{msg}, \beta_x), \beta_{sPIa}, \mathsf{Nonce}), \alpha)) \\
Z_3 : \mathsf{mac}(\mathsf{sh}(\alpha, \alpha), \mathsf{Nonce}, \gamma_o, \gamma_{sA1}, \mathsf{msg}, \mathsf{Nonce}, \beta_{nb}, \\
\qquad \mathsf{prf}(\mathsf{kdf}(\mathsf{Nonce}, \beta_{nb}, \mathsf{exp}(\mathsf{msg}, \beta_y), \mathsf{Nonce}, \beta_{sPIb}), \alpha)) \\
X_4, Y_3, Z_4, U_3 : \gamma_{sA2} \\
X_5, Y_4, Z_5, U_4 : \gamma_{tSa} \\
X_6, Y_5, Z_6, U_5 : \gamma_{tSb} \\
Y_6, X_7 : \mathsf{kdf}(\beta_{na}, \mathsf{Nonce}, \mathsf{exp}(\mathsf{msg}, \beta_x), \beta_{sPIa}, \mathsf{Nonce}) \\
U_6, Z_7 : \mathsf{kdf}(\mathsf{Nonce}, \beta_{nb}, \mathsf{exp}(\mathsf{msg}, \beta_y), \mathsf{Nonce}, \beta_{sPIb}) \\
Y_2 : \mathsf{mac}(\mathsf{sh}(\alpha, \alpha), \beta_{sPIa}, \mathsf{Nonce}, \gamma_{sA1}, \mathsf{msg}, \mathsf{Nonce}, \beta_{na}, \\
\qquad \mathsf{prf}(\mathsf{kdf}(\beta_{na}, \mathsf{Nonce}, \mathsf{exp}(\mathsf{msg}, \beta_x), \beta_{sPIa}, \mathsf{Nonce}), \alpha)) \\
U_2 : \mathsf{mac}(\mathsf{sh}(\alpha, \alpha), \mathsf{Nonce}, \beta_{sPIb}, \gamma_{sA1}, \mathsf{exp}(\gamma_g, y), \beta_{nb}, \mathsf{Nonce}, \\
\qquad \mathsf{prf}(\mathsf{kdf}(\mathsf{Nonce}, \beta_{nb}, \mathsf{exp}(\mathsf{msg}, \beta_y), \mathsf{Nonce}, \beta_{sPIb}), \alpha)).
\end{array}$$

Note that neither the first and the third clauses nor the second and the last clauses are pattern-disjoint. As the clauses in each pair are similar and they are not user-defined, we can choose either one of them. Let us choose the first clause in the first pair and the second clause in the second pair. We therefore obtain the following clauses:

$$\begin{array}{rcl}
f(\{\!|X_1, X_2, X_3, X_4, X_5, X_6|\!\}_{X_7}) & = & \langle f(X_1), f(X_2), f(X_3), f(X_4), f(X_5), f(X_6)\rangle, \\
f(\{\!|Y_1, Y_2, X_3, Y_4, Y_5|\!\}_{Y_6}) & = & \langle f(Y_1), f(Y_2), f(Y_3), f(Y_4), f(Y_5)\rangle.
\end{array}$$

We also update the types of the patterns and obtain new types for X_3 and Y_2 as follows.

$$\begin{aligned}
X_3 :\ & \mathsf{mac}(\mathsf{sh}(\alpha,\alpha), \mathsf{Nonce}, \gamma_o, \gamma_{sA1}, \mathsf{msg}, \mathsf{Nonce}, \mathsf{Nonce}, \\
& \quad \mathsf{prf}(\mathsf{kdf}(\mathsf{Nonce}, \mathsf{Nonce}, \mathsf{exp}(\mathsf{msg}, \mathsf{Nonce}), \mathsf{Nonce}, \mathsf{Nonce}), \alpha)) \\
Y_2 :\ & \mathsf{mac}(\mathsf{sh}(\alpha,\alpha), \mathsf{Nonce}, \mathsf{Nonce}, \gamma_{sA1}, \mathsf{msg}, \mathsf{Nonce}, \mathsf{Nonce}, \\
& \quad \mathsf{prf}(\mathsf{kdf}(\mathsf{Nonce}, \mathsf{Nonce}, \mathsf{exp}(\mathsf{msg}, \mathsf{Nonce}), \mathsf{Nonce}, \mathsf{Nonce}), \alpha)).
\end{aligned}$$

We complete the typed abstraction by adding homomorphic clauses for all other protocol subterms that need to be transformed. Since the macs only occur as the top-level fields, no such subterms have types that overlap with the types of the macs. Therefore, pattern-disjointness is guaranteed. $\spadesuit$

5.1.2. Generating an Atom-and-variable Removal Abstraction

An atom-and-variable removal simply eliminates atoms and variables that are unprotected and do not occur in the properties of interest. We describe in Algorithm 4 how to generate this abstraction. The loop in lines 3-11 attempts to remove an atom-and-

Algorithm 4 Generating an atom-and-variable removal abstraction.

`Input:` a protocol specification P and property formula ϕ.
`Output:` abstract protocol P' and property ϕ'.

1: $S \leftarrow av(\mathcal{M}_P) \setminus av(Sec_\phi \cup EqTerm_\phi)$
2: $T \leftarrow \emptyset$
3: **for all** $t \in S$ such that t is clear in $\mathcal{M}_P$
4: $\quad T \leftarrow T \cup \{t\}$
5: $\quad P' \leftarrow protAVRemoval(P, T)$
6: $\quad \phi' \leftarrow propAVRemoval(\phi, T)$
7: $\quad safe \leftarrow safeAVRemovalAbstraction(P', \phi', T)$
8: $\quad$ **if not** $safe$ **then**
9: $\quad\quad T \leftarrow T \setminus \{t\}$
10: $\quad$ **endif**
11: **endfor**
12: **return** $(rem_T(P), rem_T(\phi))$

variable that is clear in the protocol messages. The condition in line 8 checks whether this atom-and-variable removal satisfies the soundness conditions of Theorem 5. If these conditions are violated then we keep the corresponding atom-and-variable. Otherwise, it is added to the removal set T. Once T is completely computed, the algorithm abstracts P and ϕ accordingly and returns the result.

5.1.3. Generating a Redundancy Removal Abstraction

A redundancy abstraction removes fields u from terms t in protocol events $ev(t)$ whenever u is deducible from the remaining fields of t and the terms occurring in the events preceding $ev(t)$. This is described in Algorithm 5. Overloading notation, we denote by $set(l)$ the set of the elements of the list l. We introduce two auxiliary functions.

makeList(t) returns the list that contains all components of t. The resulting list also preserves the order of these components in t, e.g.,

$$makeList(\langle a, \langle a, b\rangle\rangle) = [a, a, b].$$

filter(l, l') returns the list obtained by eliminating elements of l that occur in l', e.g.,

$$filter([a, b], [a, c]) = [b].$$

makeTerm(l) turns the list l into a tuple, e.g.,

$$makeTerm([a, a, b]) = \langle a, a, b\rangle.$$

Algorithm 5 Generating a redundancy removal abstraction.

`Input:` a protocol specification P and property formula ϕ.
`Output:` abstract protocol P' and property ϕ'.

1: $rd \leftarrow id$
2: **for all** $r \in dom(P)$
3:     **for all** $ev(t)$ **in** $P(r)$
4:         $T \leftarrow IK_0 \cup \{m \mid ev'(m)$ preceding $ev(t)$ in $P(r)\} \cup \mathcal{V}_\alpha$
5:         $rem \leftarrow []$
6:         $S \leftarrow makeList(t)$
7:         **for all** u in S
8:             $l = filter(S, rem \cdot [u])$
9:             **if** $T \cup set(l) \vdash_E u$ **then**
10:                 $rd[t \leftarrow makeTerm(l)]$
11:                 $safe \leftarrow safeRedundancyAbstraction(rd(P), rd(\phi))$
12:                 **if** $safe\ \wedge$ condition (ii) of Definition 24 holds **then**
13:                     $rem \leftarrow rem \cdot [u]$
14:                 **endif**
15:             **endif**
16:         **endfor**
17:         $rd[t \leftarrow makeTerm(filter(S, rem))]$
18:     **endfor**
19: **endfor**
20: **return** $(rd(P), rd(\phi))$

In line 1, the redundancy abstraction rd is initialized with the identity function. The nested loop (lines 2-19) identifies redundancies in each protocol message and defines the corresponding redundancy removal. The condition in line 9 checks if each field u of the considered protocol message t is redundant, i.e., deducible from the initial knowledge of the intruder plus the messages in preceding events and the other fields and agent variables of t. If it is the case then the condition in line 12 checks whether the redundancy removal abstraction that removes u from t satisfies the corresponding soundness conditions of Theorem 6 and the extractability condition of Definition 24. If these conditions hold then u is added to the removal list (line 13). Finally, the redundancy removal abstraction is completed and the resulting abstract protocol and security properties are returned in line 20.

Note that by the algorithm, it is clear that $split(rd(t)) \subseteq split(t)$ for all $t \in \mathcal{M}_P$. Hence, the condition in line 9 ensures that point (i) in Definition 24 holds. Moreover, the second conjunct of the condition in line 12 ensures that point (ii) in Definition 24 also holds. Hence, the function rd computed by the algorithm is a redundancy removal abstraction.

5.2. Implementation of the Abstraction Mechanism for the Scyther Tool

Scyther [41] is a state-of-the art automated security protocol verification tools. It supports verification for both a bounded and an unbounded number of threads. It also supports multi-protocol analysis, i.e., verifying a composition of multiple protocols. Scyther takes as input a security protocol description specified by a set of linear role scripts which include the intended security properties. The tool supports both types and hash function symbols defined by the user. These features match our setting very well.

Scyther allows us to specify various security properties such as secrecy and authentication in terms of *security claims* (or *claim events*), which are part of security protocol descriptions. Furthermore, the tool is able to verify *synchronization* properties which are considerably stronger than the classical authentication properties defined by Lowe [87]. In order to prove (or falsify) security claims, the tool uses a backward-search technique based on trace patterns, i.e., symbolic representations of sets of traces. Even though Scyther terminates on a vast majority of case studies for an unbounded number of sessions, the verification of complex protocols is time consuming and sometimes infeasible. This makes Scyther an excellent candidate for our protocol abstractions.

The abstraction module is built internally into Scyther. This design choice allows us to re-use Scyther's data structures and functionalities without requiring an additional interface. To guide the abstraction generation process, the user can provide as an input a (possibly partial) specification of a typed abstraction, i.e., a list of clauses that specifies how a term is transformed. We distinguish two types of user-defined clauses, namely *linear* and *persistent* clauses. Linear clauses are only taken into account in the first abstraction step, while persistent clauses affect the whole abstraction generation process. Persistent clauses are marked with the symbol '@'.

In this section, we first present the correspondence between claim events in Scyther and our security property formulas. Then, we describe extensions of the labeling mechanism and the abstraction heuristics for Scyther. Finally, we demonstrate our abstraction mechanism on the Needham-Schoeder public-key protocol.

5.2.1. Claim Events and Security Properties

In Scyther, security properties are specified by means of *claim events* which are integrated into protocol role specifications. Intuitively, claim events express the intended security goal that an agent executing a given protocol role expects to achieve. For our implementation, we consider the following types of claim events that are used to express secrecy and various forms of authentication properties. We adopt the definitions of these properties from [87, 41, 40].

1. $\mathsf{claim}(A, \mathsf{Secret}, t)$ expresses the *secrecy* of term t for role A, i.e., whenever an agent a executes role A up to the claim event and all protocol roles are mapped to honest agents, term t cannot be derived to the adversary.

2. $\mathsf{claim}(A, \mathsf{Alive})$ expresses the *aliveness* property for role A, i.e., whenever an agent a in role A completes a run of the protocol, apparently with an agent b, then b has previously been running the protocol.

   Note that this property still holds even when b's thread was running the protocol with someone else (not a). Strengthening aliveness leads us to the notion of *weak agreement* property.

3. $\mathsf{claim}(A, \mathsf{Weakagree})$ expresses *weak agreement* property for role A, i.e., whenever an agent a in role A completes a run of the protocol, apparently with an agent b, then b has previously been running the protocol, apparently with a.

   Neither aliveness nor weak agreement guarantee that agents agree on their respective roles or the data exchanged. This additional requirement is captured by *non-injective agreement.*

4. $\mathsf{claim}(A, \mathsf{Commit}, B, m)$ and $\mathsf{claim}(B, \mathsf{Running}, A, m')$ are used to formalize *non-injective agreement* as defined by Lowe [87]. We say that a protocol guarantees to an agent a in role A non-injective agreement with an agent b on a message m if, whenever a completes a run of the protocol in role A, apparently with b, then b in role B has previously been running the protocol, apparently with a, and the instances of m and m' agree according to the local views of these agents.

5. $\mathsf{claim}(A, \mathsf{Niagree})$ expresses another form of non-injective agreement stating that a protocol satisfies non-injective agreement if for all instantiated claims in any trace of a given security protocol, there exists runs for the other roles in the protocol, such that all events causally preceding the claim in the protocol specification must have occurred before the claim in the trace, and each pair of matching send and receive events agree on the messages they contain.

6. $\mathsf{claim}(A, \mathsf{Nisynch})$ expresses the *non-injective synchronization* property. It strengthens $\mathsf{claim}(A, \mathsf{Niagree})$ by additionally requiring that the order of the events preceding $\mathsf{claim}(A, \mathsf{Nisynch})$ must be correct as found in the protocol description, i.e., the send events occur before the corresponding receive events.

Note that non-injective agreement specified by $\mathsf{claim}(A, \mathsf{Niagree})$ is different from that specified by the $\mathsf{Running}$ and Commit signals. The property does not require agreement on a set of data values. Instead, it requires agreement on the messages exchanged between the agents.

We now explain how to formalize these properties in our security property language using an example.

Example 35. Let us consider the Needham-Schroeder public-key protocol from Example 6. In Figure 5.2, we present a description of this protocol in Scyther's specification language.

protocol $NS(A,B)$
{

role A
{
fresh na : Nonce;
var Nb : Nonce;

send_1$(A, B, \{A, na\}\mathsf{pk}(B))$;
recv_2$(B, A, \{na, Nb\}\mathsf{pk}(A))$;
send_3$(A, B, \{Nb\}\mathsf{pk}(B))$;

claim$(A,$ Commit$, B, na, Nb)$;
claim$(A,$ Secret$, na)$;
claim$(A,$ Alive$)$;
claim$(A,$ Weakagree$)$;
claim$(A,$ Niagree$)$;
claim$(A,$ Nisynch$)$;
}
role B
{
fresh nb : Nonce;
var Na : Nonce;

recv_1$(A, B, \{A, Na\}\mathsf{pk}(B))$;

claim$(B,$ Running$, A, Na, nb)$;

send_2$(B, A, \{Na, nb\}\mathsf{pk}(A))$;
recv_3$(A, B, \{nb\}\mathsf{pk}(B))$;

}
}

Figure 5.2.: A specification of the Needham-Schroeder public-key protocol in Scyther.

The fresh and var keywords are used to declare fresh values and variables of certain types, respectively. The send and recv events denote sending and receiving of messages, respectively. For example, send event send_1$(A, B, \{A, na\}\mathsf{pk}(B))$ denotes the sending of the message $\{A, na\}\mathsf{pk}(B)$ by an agent executing role A to an agent executing role B. In Scyther's protocol specification language, each send (receive) event is subscripted with a number. These numbers are used to specify the correspondence between matching send and receive events. In order to formalize the security properties specified in the protocol description, we mimic the claim events by introducing corresponding signal events. Hence, we define

$$Sig = \{\mathsf{Create}, \mathsf{Secret}, \mathsf{Commit}, \mathsf{Running}, \mathsf{Alive}, \mathsf{Weakagree}, \mathsf{Niagree}, \mathsf{Nisynch}\}.$$

The signal event Create marks the start of a new protocol thread, which mimics the semantics of the Create event defined in [40, page 27]. The remaining signals represent the corresponding claim events. Our formalization of the Needham-Schroeder public-key protocol is now given as follows.

$$\begin{aligned} NS(A) &= \mathsf{Create} \cdot \mathsf{send}(\{A, na\}_{\mathsf{pk}(B)}) \cdot \mathsf{rcv}(\{na, Nb\}_{\mathsf{pk}(A)}) \cdot \mathsf{Running} \cdot \mathsf{send}(\{Nb\}_{\mathsf{pk}(B)}) \cdot \\ &= \mathsf{Commit} \cdot \mathsf{Secret} \cdot \mathsf{Alive} \cdot \mathsf{Weakagree} \cdot \mathsf{Niagree} \cdot \mathsf{Nisynch} \end{aligned}$$

$$\begin{aligned} NS(B) &= \mathsf{Create} \cdot \mathsf{rcv}(\{A, Na\}_{\mathsf{pk}(B)}) \cdot \mathsf{Running} \cdot \mathsf{send}(\{Na, nb\}_{\mathsf{pk}(A)}) \cdot \mathsf{rcv}(\{nb\}_{\mathsf{pk}(B)}) \cdot \\ &= \mathsf{Commit} \cdot \mathsf{Secret} \cdot \mathsf{Alive} \cdot \mathsf{Weakagree} \cdot \mathsf{Niagree} \cdot \mathsf{Nisynch} \end{aligned}$$

We formalize the secrecy of nonce na, aliveness, weak agreement, non-injective agreement, and non-injective synchronization properties for role A as follows.

1. **Secrecy** of na:

$$\begin{aligned} \phi^{NS}_{sec} &= \forall j.\ (role(j, A) \wedge honest(j, [A, B]) \wedge steps(j, \mathsf{Secret})) \\ &\quad \Rightarrow secret(na) \end{aligned}$$

2. **Aliveness**:

$$\begin{aligned} \phi^{NS}_{\mathsf{alive}} &= \forall j.\ (role(j, A) \wedge honest(j, [A, B]) \wedge steps(j, \mathsf{Alive})) \\ &\quad \Rightarrow (\exists k.\ steps(k, \mathsf{Create}) \wedge \\ &\quad\quad ((role(k, A) \wedge A^k = B^j) \vee \\ &\quad\quad\ (role(k, B) \wedge B^k = B^j))) \end{aligned}$$

3. **Weak agreement**:

$$\begin{aligned} \phi^{NS}_{\mathsf{wagree}} &= \forall j.\ (role(j, A) \wedge honest(j, [A, B]) \wedge steps(j, \mathsf{Weakagree})) \\ &\quad \Rightarrow (\exists k.\ steps(k, \mathsf{Create}) \wedge \\ &\quad\quad ((role(k, A) \wedge A^k = B^j \wedge B^k = A^j) \vee \\ &\quad\quad\ (role(k, B) \wedge B^k = B^j \wedge A^k = A^j))) \end{aligned}$$

4. **Non-injective agreement** (on na and nb) **based on Running and Commit claims**:

$$\begin{aligned} \phi^{NS}_{\mathsf{cm}} &= \forall j.\ (role(j, A) \wedge honest(j, [A, B]) \wedge steps(j, \mathsf{Commit})) \\ &\quad \Rightarrow (\exists k.\ role(k, B) \wedge steps(k, \mathsf{Running}) \wedge \\ &\quad\quad \langle A^j, B^j, na^j, Nb^j \rangle = \langle A^k, B^k, Na^k, nb^k \rangle) \end{aligned}$$

5. **Non-injective agreement specified by** $\mathsf{claim}(A, \mathsf{Niagree})$:

$$\begin{aligned}
\phi^{NS}_{\mathsf{niagree}} \;=\; & \forall j.\ (role(j, A) \wedge honest(j, [A, B]) \wedge steps(j, \mathsf{Niagree})) \\
& \Rightarrow (\exists k.\ role(k, B) \wedge \\
& \quad steps(k, \mathsf{rcv}(\{A, Na\}_{\mathsf{pk}(A)})) \prec steps(j, \mathsf{Niagree}) \wedge \\
& \quad steps(k, \mathsf{send}(\{Na, nb\}_{\mathsf{pk}(A)})) \prec steps(j, \mathsf{Niagree}) \wedge \\
& \quad \langle A^j, B^j \rangle = \langle A^k, B^k \rangle \wedge \\
& \quad \{A^j, na^j\}_{\mathsf{pk}(B^j)} = \{A^k, Na^k\}_{\mathsf{pk}(B^k)} \wedge \\
& \quad \{na^j, Nb^j\}_{\mathsf{pk}(A^j)} = \{Na^k, nb^k\}_{\mathsf{pk}(A^k)})
\end{aligned}$$

6. **Non-injective synchronization**:

$$\begin{aligned}
\phi^{NS}_{\mathsf{nisyn}} \;=\; & \forall j.\ (role(j, A) \wedge honest(j, [A, B]) \wedge steps(j, \mathsf{Nisynch})) \\
& \Rightarrow (\exists k.\ role(k, B) \wedge \\
& \quad steps(j, \mathsf{send}(\{A, na\}_{\mathsf{pk}(B)})) \prec steps(k, \mathsf{rcv}(\{A, Na\}_{\mathsf{pk}(B)})) \wedge \\
& \quad steps(k, \mathsf{send}(\{Na, nb\}_{\mathsf{pk}(A)})) \prec steps(j, \mathsf{rcv}(\{na, Nb\}_{\mathsf{pk}(A)})) \wedge \\
& \quad \langle A^j, B^j \rangle = \langle A^k, B^k \rangle \wedge \\
& \quad \{A^j, na^j\}_{\mathsf{pk}(B^j)} = \{A^k, Na^k\}_{\mathsf{pk}(B^k)} \wedge \\
& \quad \{na^j, Nb^j\}_{\mathsf{pk}(A^j)} = \{Na^k, nb^k\}_{\mathsf{pk}(A^k)})
\end{aligned}$$

♠

5.2.2. Refinement of the Abstraction Heuristics

In this subsection, we discuss some limitations of our abstraction heuristics that potentially lead to introduction of false negatives. We then provide an extension of the heuristics that addresses these limitations.

Labeling mechanism

In practice, it turns out that the labeling mechanism described in Subsection 5.1.1 is not sufficient to achieve good abstractions. There are protocols that employ cryptographic primitives in particular ways to achieve certain security goals, even though these primitives do not provide the desired properties themselves. In such cases, the heuristic may assign security labels to essential terms incorrectly, or accidentally remove elements that are important to achieve these properties, as illustrated in the following example.

Example 36. Let us consider the Needham-Schroeder public-key protocol from Example 6 whose specification is given as follows.

$$\begin{aligned}
NS(A) &= \mathsf{send}(\{A, na\}_{\mathsf{pk}(B)}) \cdot \mathsf{rcv}(\{na, Nb\}_{\mathsf{pk}(A)}) \cdot \mathsf{send}(\{Nb\}_{\mathsf{pk}(B)}) \\
NS(B) &= \mathsf{rcv}(\{A, Na\}_{\mathsf{pk}(B)}) \cdot \mathsf{send}(\{Na, nb\}_{\mathsf{pk}(A)}) \cdot \mathsf{rcv}(\{nb\}_{\mathsf{pk}(B)})
\end{aligned}$$

Suppose that we are interested in non-injective agreement for an agent in role A with an agent in role B on the nonce na. The agent variable A in the first sent message is crucial to achieve this property. However, our heuristic may pull A out of the messages $\{A, na\}_{\mathsf{pk}(B)}$ and $\{A, Na\}_{\mathsf{pk}(B)}$, as this abstraction preserves global label $(\mathsf{NO\_CONF}, \mathsf{NO\_AUTH})$ of A. It is not hard to see that the resulting abstracted protocol no longer provides the desired property. Furthermore, the heuristic incorrectly decides that na has global label $(\mathsf{NO\_CONF}, \mathsf{NO\_AUTH})$ according to our labeling mechanism. Thus, we may also pull na out of the encryptions in the first two events of role A. Since na has global label $(\mathsf{NO\_CONF}, \mathsf{NO\_AUTH})$, this label is clearly preserved under this abstraction. However, no authentication is guaranteed for the abstracted protocol. ♠

Similarly, the presence of agent identities may also allow us to prevent certain attacks on authentication such as the one in the following example. To deal with this issue, we enable the heuristic to detect such a pattern, i.e., an asymmetric encryption that includes an agent identity which is different from the one indicated in the encryption key. In these case, at least one occurrence of the identity must be kept, and the encryptions are associated with security label $(\mathsf{CONF}, \mathsf{AUTH})$.

Treatment of agent variables in symmetric encryptions

As discussed above, agent identities can be crucial to asymmetric encryptions in certain contexts. For symmetric encryptions, the presence of agent identities may also allow us to prevent certain attacks such as the one in the following example.

Example 37. We consider the protocol P where the initiator in role A sends to the responder in role B a nonce na encrypted together with his name using their (bidirectional) pre-shared long-term key.

$$\begin{array}{rcl} P(A) & = & \mathsf{send}(\{\!|A, na|\!\}_{\mathsf{sh}(A,B)}) \\ P(B) & = & \mathsf{rcv}(\{\!|A, Na|\!\}_{\mathsf{sh}(A,B)}) \end{array}$$

Once the responder has received his message, he can deduce that the initiator indeed has sent the message. This is no longer true if we pull the initiator identity out of the protocol messages, as the protocol P'

$$\begin{array}{rcl} P'(A) & = & \mathsf{send}(A, \{\!|na|\!\}_{\mathsf{sh}(A,B)}) \\ P'(B) & = & \mathsf{rcv}(A, \{\!|Na|\!\}_{\mathsf{sh}(A,B)}) \end{array}$$

is vulnerable to the following reflection attack:

$$\begin{array}{rcl} b \to I(a) & : & b, \{\!|na|\!\}_{\mathsf{sh}(b,a)} \\ I(a) \to b & : & a, \{\!|na|\!\}_{\mathsf{sh}(b,a)} \end{array}$$

In this attack, the intruder pretends to be a and intercepts the message sent by b. He later replays this message back to b and tricks b into thinking that the message was coming from a. Therefore, we must keep agent identities that occur in symmetric encryptions. ♠

5.2.3. Demonstration on the Needham-Schoeder Public-Key Protocol

To illustrate how our abstraction mechanism works for the Scyther tool, we come back to the Needham-Schroeder public-key (NSPK) protocol from Example 6. For simplicity, we consider only one security property, namely the secrecy of the nonce na for the initiator role. We specify this protocol as follows.

$$\begin{array}{rcl} NS(A) & = & \mathsf{send}(\{A, na\}_{\mathsf{pk}(B)}) \cdot \mathsf{rcv}(\{na, Nb\}_{\mathsf{pk}(A)}) \cdot \mathsf{send}(\{Nb\}_{\mathsf{pk}(B)}) \cdot \mathsf{Secret} \\ NS(B) & = & \mathsf{rcv}(\{A, Na\}_{\mathsf{pk}(B)}) \cdot \mathsf{send}(\{Na, nb\}_{\mathsf{pk}(A)}) \cdot \mathsf{rcv}(\{nb\}_{\mathsf{pk}(B)}) \end{array}$$

In order to verify this property, the following three steps are taken.

Step 1 The abstraction generator computes different more abstract protocols in two rounds.

- **Abstraction 1**: It starts with the original protocol. To generate a typed abstraction, it first computes global labels of essential terms. As the property we are interested in is the secrecy of the nonce na which is the only essential term and has property label $(\mathsf{CONF}, \mathsf{NO\_AUTH})$. It then computes the protocol label of na. Note that na occurs only in the first two events of role A. As $\{A, na\}_{\mathsf{pk}(B)}$ is a public-key encryption that contains an agent identity in plaintext, we have $\ell_1(\{A, na\}_{\mathsf{pk}(B)}) = (\mathsf{CONF}, \mathsf{AUTH})$. Hence na receives the label $(\mathsf{CONF}, \mathsf{AUTH})$ from $\{na, Nb\}_{\mathsf{pk}(A)}$. The message $\{na, Nb\}_{\mathsf{pk}(A)}$ does not contain essential agent identities and therefore $\ell_1(\{na, Nb\}_{\mathsf{pk}(A)}) = (\mathsf{CONF}, \mathsf{NO\_AUTH})$. Thus na has label $(\mathsf{CONF}, \mathsf{NO\_AUTH})$ from $\{na, Nb\}_{\mathsf{pk}(A)}$. Therefore, the global label of na is $(\mathsf{CONF}, \mathsf{NO\_AUTH})$.

  After computing the global labels of essential terms, it defines typed abstractions for the topmost cryptographic operations, i.e., for terms $\{A, na\}_{\mathsf{pk}(B)}$, $\{na, Nb\}_{\mathsf{pk}(A)}$, and $\{Nb\}_{\mathsf{pk}(B)}$ from role A, and $\{A, Na\}_{\mathsf{pk}(B)}$, $\{Na, nb\}_{\mathsf{pk}(A)}$, and $\{nb\}_{\mathsf{pk}(B)}$ from role B. The list of clauses E_f is initially empty. It computes typed abstractions for the terms in role A as follows.

  - $\{A, na\}_{\mathsf{pk}(B)}$: It keeps the agent variable A as this variable is essential. Pulling na out of the encryption does not preserve label CONF. Therefore, it also keeps na. It defines a clause for this abstraction by

    $$f(\{X_1, Y_1\}_{Z_1}) = \{f(X_1), f(Y_1)\}_{f(Z_1)}$$

    where $X_1 : \alpha$, $Y_1 : \beta_{na}$, and $Z_1 : \mathsf{pk}(\alpha)$.
  - $\{na, Nb\}_{\mathsf{pk}(A)}$: By the same reason as before, it keeps na. Since Nb is not an essential term, it pulls Nb out of the encryption. The corresponding clause is defined by

    $$f(\{X_2, Y_2\}_{Z_2}) = \langle \{f(X_2)\}_{f(Z_2)}, f(Y_2) \rangle$$

    where $X_2 : \beta_{na}$, $Y_2 : \mathsf{Nonce}$, and $Z_2 : \mathsf{pk}(\alpha)$.

- $\{Nb\}_{\mathsf{pk}(B)}$: It removes the encryption with the clause

  $$f(\{X_3\}_{Z_3}) = f(X_3)$$

  where $X_3 \colon \mathsf{Nonce}$ and $Z_3 \colon \mathsf{pk}(\alpha)$.

These clauses are added to E_f without creating any violation of pattern-disjointness. Therefore no changes in E_f need to be done. The typed abstractions for role B are defined as follows.

- $\{A, Na\}_{\mathsf{pk}(B)}$: The only difference we observe here compared to the previous case is that Na is not essential. Therefore, It pulls Na out of the encryption and defines the clause

  $$f(\{X_4, Y_4\}_{Z_4}) = \langle \{f(X_4)\}_{f(Z_4)}, f(Y_4)\rangle$$

  where $X_4 \colon \alpha$, $Y_4 \colon \mathsf{Nonce}$, and $Z_1 \colon \mathsf{pk}(\alpha)$. Adding this clause to E_f violates the pattern-disjointness since

  $$\begin{array}{rcl} \Gamma(\{X_1, Y_1\}_{Z_1}) & = & \{\alpha, \beta_{na}\}_{\mathsf{pk}(\alpha)}, \\ \Gamma(\{X_4, Y_4\}_{Z_4}) & = & \{\alpha, \mathsf{Nonce}\}_{\mathsf{pk}(\alpha)}, \text{ and} \\ \{\alpha, \beta_{na}\}_{\mathsf{pk}(\alpha)} & \preccurlyeq & \{\alpha, \mathsf{Nonce}\}_{\mathsf{pk}(\alpha)}. \end{array}$$

  In this case, it chooses the first clause in E_f because it abstracts less than this one does. The updated E_f consists of the following clauses:

  $$\begin{array}{lrcl} (1) & f(\{X_1, Y_1\}_{Z_1}) & = & \{f(X_1), f(Y_1)\}_{f(Z_1)} \\ (2) & f(\{X_2, Y_2\}_{Z_2}) & = & \langle \{f(X_2)\}_{f(Z_2}, f(Y_2)\rangle \\ (3) & f(\{X_3\}_{Z_3}) & = & f(X_3) \end{array}$$

- $\{Na, nb\}_{\mathsf{pk}(A)}$: Since neither Na nor nb is essential, it removes the encryption and thus defines the clause

  $$f(\{X_5, Y_5\}_{Z_5}) = \langle f(X_5), f(Y_5)\rangle$$

  where $X_5 : \mathsf{Nonce}$, $Y_5 : \beta_{nb}$, and $Z_2 : \mathsf{pk}(\alpha)$. This clause and clause (2) in E_f violate the pattern-disjointness. As these clauses have the same shape, we can keep either of them and update types accordingly. The abstraction generator chooses to keep clause (2) as it abstracts less and updates the type of X_2 to Nonce.

- $\{nb\}_{\mathsf{pk}(B)}$: Since nb is not essential, it removes the encryption defines the clause

  $$f(\{X_6\}_{Z_6}) = f(X_6)$$

  where $X_6 \colon \beta_{nb}$ and $Z_6 \colon \mathsf{pk}(\alpha)$. This clause and clause (3) in E_f violate the pattern-disjointness. As before, clause (3) is chosen and no changes need to be done.

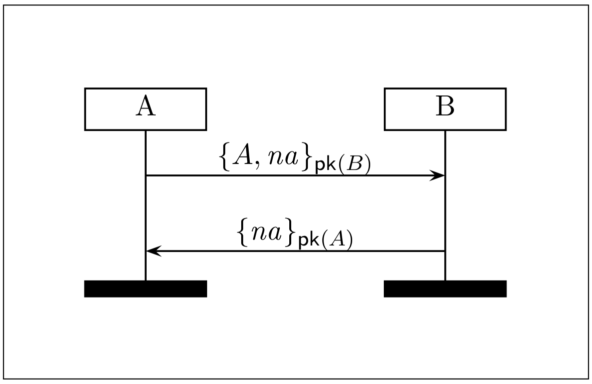

Figure 5.3.: An abstracted protocol of NSPK.

Finally, the list E_f consists of the following clauses:

$$\begin{array}{lrcl} (1) & f(\{X_1, Y_1\}_{Z_1}) & = & \{f(X_1), f(Y_1)\}_{f(Z_1)} \\ (2) & f(\{X_2, Y_2\}_{Z_2}) & = & \langle \{f(X_2)\}_{f(Z_2}, f(Y_2)\rangle \\ (3) & f(\{X_3\}_{Z_3}) & = & f(X_3) \end{array}$$

where $X_4 \colon \alpha$, $X_2, X_3, Y_4, Y_2 \colon \mathsf{Nonce}$, and $Z_2, Z_4 \colon \mathsf{pk}(\alpha)$.

The abstraction generator now checks whether $\mathcal{M}_{NS} \cup \{na\} \subseteq udom(F_f,)$. It realizes that a clause for transforming public keys is missing. It therefore adds to E_f the homomorphic clause $f(\mathsf{pk}(U)) = \mathsf{pk}(f(U))$ with $U \colon \alpha$. It also checks that the soundness conditions are satisfied. Applying this abstraction, it achieves the following abstracted protocol.

$$\begin{array}{rcl} NS(A) & = & \mathsf{send}(\{A, na\}_{\mathsf{pk}(B)}) \cdot \mathsf{rcv}(\{na\}_{\mathsf{pk}(A)}, Nb) \cdot \mathsf{send}(Nb) \cdot \mathsf{Secret} \\ NS(B) & = & \mathsf{rcv}(\{A, Na\}_{\mathsf{pk}(B)}) \cdot \mathsf{send}(\{Na\}_{\mathsf{pk}(A)}, nb) \cdot \mathsf{rcv}(nb) \end{array}$$

Next, it applies untyped abstractions to remove inessential terms that are unprotected. It first applies the atom-and-variable removal that eliminates all occurrences of nb and Nb. As there are no redundancies, no further abstraction is applied. Thus, it computes the final abstracted protocol in the first abstraction step as follows.

$$\begin{array}{rcl} NS(A) & = & \mathsf{send}(\{A, na\}_{\mathsf{pk}(B)}) \cdot \mathsf{rcv}(\{na\}_{\mathsf{pk}(A)}) \cdot \mathsf{Secret} \\ NS(B) & = & \mathsf{rcv}(\{A, Na\}_{\mathsf{pk}(B)}) \cdot \mathsf{send}(\{Na\}_{\mathsf{pk}(A)}) \end{array}$$

In Figure 5.3, we depict the description of this abstraction protocol.

- **Abstraction 2**: The abstraction module picks the abstracted protocol (depicted in Figure 5.3) computed in the previous abstraction step and tries to simplify it. However, no further simplification is produced.

Step 2 : It analyzes the protocol on the top of the stack. Scyther terminates and verifies the property.

Step 3 : As there is no attack, it concludes that our original protocol provides the secrecy property.

Note that our abstractions are sound, but not complete. Therefore, we may encounter false negatives, i.e., spurious attacks. We currently do not check automatically whether an attack is spurious. Whenever an attack on a protocol P is found, we proceed to analyze (only) the failed properties on the next more concrete protocol in the series of abstractions. Hence, we implicitly consider all attacks found in abstract models as spurious attacks. However, if the attack is not spurious, i.e., the original protocol indeed does not satisfies the properties, then we can only conclude after verifying all the protocol models including the original one. This apparently increases the overall verification time. For future work, we plan to complete the automation of the abstraction-refinement process by implementing the spurious-attack checking functionality.

6. Experimental Results

We have validated the effectiveness of our abstractions on 23 members of the IKE and ISO/IEC 9798 protocol families, the PANA-AKA protocol [8], and the KSL protocol. We verify these protocols using five tools based on four different techniques: Scyther [41], CL-Atse [122], OFMC [15], SATMC [9], and ProVerif [19]. Only Scyther and ProVerif support verification of an unbounded number of threads. We have tested our protocol abstractions on Scyther with automatically generated abstractions and on the other tools with manually produced abstractions.

6.1. Experimental Results for Scyther

In Table 6.1, we present the experimental results for Scyther. Our models of the IKE and ISO/IEC 9798 protocols are based on Cremers' [38, 39]. Since Scyther uses a fixed signature with standard cryptographic primitives and no equational theories, the IKE models approximate the DH theory by oracle roles. For the IKE protocols, we use two hash functions h and g and under-approximate the congruence relation induced by the equation $h(g(X), Y) = h(g(Y), X)$ using *oracle roles*, which translate between equivalent term representations. One such role translates $h(g(X), Y)$ into $h(g(Y), X)$.

$$S_{\mathsf{IKE_m}}(DH) = \mathsf{rcv}(h(g(X), Y)) \cdot \mathsf{send}(h(g(Y), X))$$

Other oracle roles ensure the protocol's executability by translating the messages sent by a protocol role into those received by another role and vice versa.

For our case studies, we verify several security properties including secrecy, aliveness, weak agreement, and non-injective authentication. We mark verified properties by ✓ and falsified ones by ×. An entry ✓/× means the property holds for one role but not for the other. Each row consists of two lines, corresponding to the analysis time without (line 1) and with (line 2) abstraction for 3-8 or unboundedly many (∞) threads. The times include computing the abstractions (4-20 ms) and the verification itself. We ran the experiments on a cluster of 12-core AMD Opteron 6174 processors each with 64 GB RAM each.

Verification For 11 of the 23 original protocols that are analyzed, an unbounded verification attempt results in a timeout (TO, set to 8h cpu time) or memory exhaustion (ME). In 8 of these cases our abstractions enabled the verification of all properties in less than 0.4 seconds and in one case in 78 seconds. However, for the first three protocols, we still get a timeout for the abstractions. For the large majority of the bounded verification tasks, we significantly push the bound on the number of threads and achieve massive

protocol	No	S	A	W	N	3	4	5	6	7	8	∞
IKEv1-pk2-a2	1	✓			✓	44.01	302.86	1843.80	10999.80	TO	TO	TO
						6.01	26.36	151.88	1014.63	6838.97	TO	TO
IKEv1-pk2-a	1	✓			✓	1103.63	27808.72	TO	TO	TO	TO	TO
						133.65	3356.59	TO	TO	TO	TO	TO
IKEv1-pk-a2	1	✓			✓	10.95	61.47	125.25	237.76	409.35	744.75	TO
						0.84	1.79	2.43	3.63	6.01	9.61	TO
IKEv1-pk-a22	1	✓			✓	18.48	82.93	249.55	554.09	1006.04	1734.85	TO
						0.83	1.26	2.08	3.47	5.96	10.28	TO
IKEv2-eap	5	✓			✓	TO	TO	TO	TO	TO	TO	TO
						78.35	798.44	4212.71	20911.20	TO	TO	TO
IKEv2-eap2	5	✓			✓	TO	TO	TO	TO	TO	TO	TO
						70.18	690.26	4169.87	20071.45	TO	TO	TO
IKEv2-mac	5	✓			✓	1.85	4.91	6.72	8.07	8.42	8.49	8.70
						0.62	1.77	1.83	1.73	1.73	1.80	1.74
IKEv2-mac2	5	✓			✓	2.16	4.09	6.43	9.41	8.16	8.44	8.69
						0.81	1.60	1.73	1.75	1.73	1.74	1.73
IKEv2-mactosig	4	✓			✓	11.65	141.37	1075.46	7440.81	TO	TO	TO
						2.89	12.38	24.54	38.68	53.36	65.07	77.68
IKEv2-mactosig2	4	✓			✓	11.71	133.20	1064.30	7229.13	TO	TO	TO
						2.85	11.81	24.14	38.22	53.25	64.51	77.03
IKEv2-sigtomac	5	✓			✓	6.15	33.19	65.05	115.34	204.93	206.45	237.34
						3.59	12.72	28.44	44.44	55.11	66.97	67.15
IKEv1-pk-m	2				×	48.62	269.92	507.40	869.23	16254.80	TO	TO
						0.04	0.05	0.05	0.05	0.05	0.05	TO
IKEv1-pk-m2	2				✓/×	18.26	274.87	4438.72	TO	TO	TO	TO
						1.48	7.79	32.75	110.32	339.93	963.08	TO
IKEv1-sig-m	2				×	0.34	0.45	0.45	0.45	0.45	0.46	0.44
						0.05	0.05	0.05	0.06	0.05	0.05	0.06
IKEv1-sig-m-perlman	2				×	2.86	13.99	40.78	67.83	72.08	72.15	109.03
						0.05	0.05	0.05	0.05	0.05	0.05	0.05
ISO/IEC 9798-2-5	1	✓				0.78	8.96	73.87	564.67	4214.22	TO	TO
						0.07	0.11	0.12	0.11	0.11	0.11	0.11
ISO/IEC 9798-2-6	1	✓				0.57	3.74	18.42	67.01	196.30	488.04	21278.58
						0.05	0.04	0.05	0.05	0.05	0.05	0.05
ISO/IEC 9798-3-6-1	2		✓		✓	43.08	802.95	8903.70	ME	ME	ME	ME
						0.13	0.18	0.19	0.19	0.19	0.19	0.19
ISO/IEC 9798-3-6-2	1		✓		✓	2.74	8.67	19.56	33.91	52.51	69.48	90.04
						0.12	0.15	0.15	0.15	0.15	0.15	0.15
ISO/IEC 9798-3-7-1	2		✓		✓	40.43	740.47	7483.36	16631.42	ME	ME	ME
						0.13	0.18	0.19	0.19	0.19	0.19	0.19
ISO/IEC 9798-3-7-2	1		✓		✓	2.38	7.71	16.68	26.99	35.06	49.49	TO
						0.22	0.32	0.33	0.33	0.33	0.33	0.33
PANA-AKA	5	✓	✓	✓	✓	5769.53	TO	TO	TO	TO	TO	TO
						0.10	0.10	0.10	0.10	0.10	0.10	0.10
KSL	1	✓				113.82	21378.68	TO	TO	TO	TO	TO
						0.03	0.04	0.04	0.04	0.04	0.03	0.04

Table 6.1.: Experimental results for the Scyther tool. The time is in seconds. **No**: Number of abstractions. Properties of interest are **S**ecrecy, **A**liveness, **W**eak agreement, and **N**on-injective agreement.

speedups. For example, our abstractions enable the verification of the complex nested protocols IKEv2-eap and PANA-AKA. Scyther verifies an abstraction of IKEv2-eap for up to 6 threads where it times out on the original even for 4 threads. More strikingly, it completes an unbounded verification of the simplified PANA-AKA in less than 1/10 second whereas it failed on the original already for 4 threads. Similarly, an unbounded verification of the simplified KSL takes 0.04 seconds whereas it times out on the original for 5 threads. We also achieve dramatic speedups for many other protocols, most notably for the IKEv1-pk-a22, ISO/IEC 9798-2-6, and ISO/IEC 9798-3-6-2 protocols. Moreover, the verification time for many abstracted protocols increases much more slowly than for their originals. For instance, we obtain almost constant verification times for the six ISO/IEC 9798 protocols, the PANA-AKA protocol, and the KSL protocol, whereas the time significantly increases on some originals, e.g., for ISO/IEC 9798-3-6-1 and KSL. However, for a few protocols, the speedup is more modest: the verification time drops from about 4 to 1.1 minutes for the IKEv2-sigtomac protocol and from about 8.7 seconds to 1.7 seconds for IKEv2-mac protocol. For the last ISO/IEC-9897 protocols, the time differences are subtle as only one abstract protocol is produced for each.

Falsification For rows marked by $\times$, line 2 corresponds to falsification time for the most abstract model. Falsification on the most abstract models is much faster than on the originals. For example, it takes 0.05 seconds to falsify the abstract model of IKEv1-pk-m for 8 threads, while it times out for the original one. In the unbounded case, the speedup factors are 7 for IKEv1-sig-m and 2180 for IKEv1-sig-m-perlman. We have manually analyzed the abstract attacks. Interestingly, none of them is spurious, suggesting that our measures to prevent them are effective. We expect that fast automatic detection of spurious attacks is feasible and will thus affect the current performance only negligibly.

Combination For the IKEv1-pk-m2 protocol, the tool verifies non-injective agreement for one role and falsifies it for the other one. Surprisingly, we obtain a remarkable speedup even though the analysis of this protocol is done three times (for two abstract and the original models). Our abstractions push the feasibility bound from 5 to 8 threads. As the property is verified very quickly for one role on the most abstract model, it needs to be analyzed only for the other role at lower abstraction levels. This explains the remarkable speedups we obtain and therefore illustrates an advantage of our abstraction mechanism in this case.

6.2. Experimental Results for the Avantssar Tools

The AVANTSSAR platform is an integrated toolset for the formal specification and **A**utomated **VA**lidatio**N** of **T**rust and **S**ecurity of **S**ervice-oriented **AR**chitectures. It provides three validation back-ends (CL-Atse, OFMC, and SATMC) which share the input languages for specifying protocols. The validators are based on two different techniques. SATMC reduces protocol insecurity problems to the satisfiability of propositional formulas which can then be checked by modern SAT solvers. CL-Atse and OFMC both use constraint solving techniques to search for attacks. However, they use different optimization strategies to reduce the search space. All these tools can verify protocols only for a bounded number of threads.

We have experimented with CL-Atse (version 2.5-21), OFMC (version 2013b), and SATMC (version 3.4) on several protocols from IKE and ISO/IEC 9798 families. Moreover, we have performed experiments on variants of the TLS and basic Kerberos protocols. For TLS, we distinguish two instances according to different security properties of interest. So far, we have not modelled IKE protocols for SATMC, as this requires substantial effort to encode oracles for Diffie-Hellman equations. We therefore defer extended experiments with SATMC to future work.

In our experiments, we measure the verification time for different numbers of sessions. Note that a session in CL-Atse, SATMC, and OFMC differs from a thread in Scyther. CL-Atse and SATMC specify a session as an instantiation of all protocol roles, not just a single role. For instance, a session of a protocol with three different roles results in three role instances (or three threads in Scyther) where a concrete agent is assigned to each role. In contrast, OFMC works with symbolic sessions where the agents executing the roles are not concretely specified but kept as variables.

For the AVANTSSAR tools, our experimental results generally exhibit smaller speedups than for Scyther. There is also a considerable variance between the different tools.

CL-Atse (Table 6.2) CL-Atse shows minor performance gains for the two IKEv1 protocols (pk2-a and pk-a2). However, abstraction enables the verifications of first three IKEv2 protocols (eap, eap2, and mac) for four sessions in less than 2 hours and dramatically speeds up the verification of three sessions of the eap and eap2 variants by factors greater than 690 and 900, respectively. For the last two IKEv2 protocols, the performance gains are still substantial: for four sessions we achieve a speedup factor of 7 for IKEv2-mactosig and of 107 for IKEv2-sigtomac. The best result in the ISO/IEC family is achieved for the ISO/IEC 9798-2-5 protocol where we can turn a timeout for 10 sessions into a time less than 0.2 seconds. The speedup for the 2-5 variant is less impressive and for the two 3-7 variants, we even observe an increase in verification time as we do for the basic Kerberos protocol. For TLS, the verification time of secrecy up to five sessions drops from 260 minutes to 6 minutes (factor 42), whereas that of authentication is sped up by a factor of 1.5 for four sessions.

OFMC (Table 6.3) Surprisingly, the experimental results for OFMC are almost dual to those for CL-Atse. In particular, for the two IKEv1 protocols, OFMC loses performance on the abstracted protocols compared to the originals. Nevertheless, the abstractions save a lot of effort for the remaining protocols. We are able to increase the number of tractable sessions for 8 protocols: for 2 out of 7 from the IKE family, 5 out of 6 from the ISO/IEC 9798 family, and for the basic Kerberos protocol. For TLS, the verification of authentication is 1.7 times faster (up to 3 sessions). For secrecy, the tool achieves a 20-fold speedup (up to 4 sessions). As a typical case, OFMC verifies an abstraction of ISO/IEC 9798-2-5 for 5 sessions within less than 4 seconds whereas it times out on the original for more than 2 sessions.

SATMC (Table 6.4) The abstractions enable the verification of the Kerberos and TLS protocols for 5 and even 10 sessions. In particular, the tool takes less than 21 seconds to verify the abstracted TLS protocol for 10 sessions whereas it times out for 5 sessions of the original protocol. On the negative side, SATMC loses performance for the protocols in the ISO/IEC family.

Apart from positive results, our experiments also provide an evidence that protocol abstractions are not always helpful. This is typically the case when an abstraction removes sensitive information. In particular, the performance degradation for the AVANTSSAR tools can possibly be attributed to an interference with the highly refined optimization techniques used in these tools. More precisely, an abstraction may get rid of data that is crucial to eliminate redundancies (for CL-Atse) or to limit the number of branching nodes in the symbolic search tree (for OFMC). As a result, the search space becomes larger in the abstracted protocols than in the originals. However, the influence of abstraction on the SATMC's performance is not clear. A further investigation is therefore desirable.

6.3. Experimental Results for ProVerif

ProVerif is an automated cryptographic protocol verifier in the standard Dolev-Yao model. It supports user-defined equational theories to model algebraic properties of cryptographic primitives. In contrast with Scyther, it uses approximations, e.g., translating protocol models in the applied pi calculus to a set of Horn clauses, to handle an unbounded number of sessions. These approximations are sound with respect to attacks, i.e., if the tool finds no attacks then the protocol is indeed secure.

We have validated our abstractions for ProVerif (version 1.88) on six protocols from the IKE and ISO/IEC 9798 families (see Table 6.5). For all these protocols, we observe good speedups. In particular, for the IKEv1-pk-a2 and the IKEv2-eap, the speedup factors are 6 and 5, respectively. The performance gains for the ISO/IEC 9798 protocols are less obvious than for the IKE ones. Concretely, the tool is roughly 1.5 times faster for these protocols.

protocol/prop./#sessions	S	A	W	N	3	4	5	10
IKEv1-pk2-a	✓			✓	0.06 0.05	0.11 0.08	0.53 0.32	TO TO
IKEv1-pk-a2	✓			✓	0.05 0.05	0.09 0.07	1.79 1.17	TO TO
IKEv2-eap	✓			✓	625.75 23.17	TO TO	TO TO	TO TO
IKEv2-eap2	✓			✓	1248.57 37.93	TO TO	TO TO	TO TO
IKEv2-mac	✓			✓	2.78 0.89	TO 5830.38	TO TO	TO TO
IKEv2-mactosig	✓			✓	0.24 0.12	1056.31 149.19	TO TO	TO TO
IKEv2-sigtomac	✓			✓	2.52 0.10	16710.31 155.63	TO TO	TO TO
ISO/IEC 9798-2-5	✓				20.05 0.52	TO 4064.93	TO 0.18	TO 0.17
ISO/IEC 9798-2-6	✓				1639.32 703.55	TO TO	TO TO	TO TO
ISO/IEC 9798-3-7-1		✓		✓	1.21 1973.78	4495.43 TO	TO TO	TO TO
ISO/IEC 9798-3-7-2		✓		✓	29.95 TO	TO TO	TO TO	TO TO
Kerb-basic			✓		0.30 0.18	0.29 0.18	22473.21 TO	TO TO
TLS-auth				✓	0.10 0.08	60.02 39.42	TO TO	TO TO
TLS-sec				✓	0.07 0.05	8.63 0.51	15551.63 369.57	TO TO

Table 6.2.: Experimental verification results for CL-Atse. The time is in seconds.

protocol/prop./#sessions	S	A	W	N	2	3	4	5
IKEv1-pk2-a	✓			✓	36.28 59.10	27745.29 TO	TO TO	TO TO
IKEv1-pk-a2	✓			✓	4.28 12.09	849.46 9192.14	TO TO	TO TO
IKEv2-eap	✓			✓	8920.57 10.07	TO 8942.94	TO TO	TO TO
IKEv2-eap2	✓			✓	5407.00 46.14	TO TO	TO TO	TO TO
IKEv2-mac	✓			✓	18.59 11.19	22547.87 16139.98	TO TO	TO TO
IKEv2-mactosig	✓			✓	22.08 9.27	15561.69 10605.58	TO 11782.39	TO TO
IKEv2-sigtomac	✓			✓	18.58 12.36	13617.91 12408.54	TO TO	TO TO
ISO/IEC 9798-2-5	✓				805.64 3.61	TO 3.43	TO 3.85	TO 3.59
ISO/IEC 9798-2-6	✓				7232.17 144.06	TO TO	TO TO	TO TO
ISO/IEC 9798-3-6-1		✓		✓	17941.80 27.92	TO 18019.32	TO TO	TO TO
ISO/IEC 9798-3-6-2		✓		✓	TO 12.97	TO 3673.20	TO TO	TO TO
ISO/IEC 9798-3-7-1		✓		✓	TO 50.52	TO TO	TO TO	TO TO
ISO/IEC 9798-3-7-2		✓		✓	TO 11.61	TO 4010.64	TO TO	TO TO
Kerb-basic			✓		20.63 8.07	TO 28699.72	TO TO	TO TO
TLS-auth				✓	9.12 8.88	6002.38 3549.25	TO TO	TO TO
TLS-sec				✓	0.27 0.15	13.62 1.97	1304.21 59.87	TO TO

Table 6.3.: Experimental verification results for OFMC. The time is in seconds.

number of sessions	S	A	W	N	3	4	5	10
ISO/IEC 9798-2-5	✓				0.44 0.58	0.42 0.64	0.45 0.90	0.50 3.70
ISO/IEC 9798-2-6	✓				0.45 35.36	0.46 247.67	0.48 2155.28	0.50 23740.06
ISO/IEC 9798-3-7-1		✓		✓	0.46 0.78	0.47 0.95	0.48 1.31	0.53 8.17
ISO/IEC 9798-3-7-2		✓		✓	0.47 2.64	0.47 5.83	0.64 11.61	0.60 121.17
Kerb-basic			✓		100.88 3.32	107.66 3.46	ME 51.15	TO 23396.15
TLS-auth				✓	163.51 1.52	4464.73 1.90	TO 2.65	TO 20.74
TLS-sec				✓	148.21 1.71	4002.34 1.90	TO 2.30	TO 8.85

Table 6.4.: Experimental verification results for SATMC. The time is in seconds.

protocol/prop./#threads	S	A	W	N	∞
IKEv1-pk2-a	✓			✓	43.53 15.11
IKEv1-pk-a2	$\star$				1.84 0.3
IKEv2-eap	✓			✓	22.27 4.22
IKEv2-mactosig	✓			✓	4.57 0.91
ISO/IEC 9798-2-5	✓				0.09 0.06
ISO/IEC 9798-3-7-1				✓	0.13 0.08

Table 6.5.: Experimental verification results for ProVerif. The time is in seconds. The $\star$ presents ProVerif verifies the property for one role and cannot prove it for the other.

Part III.

Related Work and Conclusion

7. Related Work

In this chapter, we present a more technical discussion of related works in comparison with ours.

7.1. Security Protocol Abstraction

Hui and Lowe [72] define several kinds of abstractions similar to ours with the aim of improving the performance of the CASPER/FDR verifier. They prove soundness of each kind of transformations based on general soundness criteria for secrecy and authentication. Their protocol model is restricted to atomic keys and they establish their results only for ground messages. Addressing these limitations, we proposed in [103] a rich class of protocol abstractions and proved its soundness for a wide range of security properties. We worked in a more general setting and discussed in detail the non-trivial issue of handling terms with variables as they appear in protocol specifications. We used a type system to uniformly transform all terms of a given type (e.g., a pattern in a protocol role and its instances during execution) whereas [72] only covers ground terms. Our class of sound security protocol abstractions strictly subsumes those defined in [72], e.g., abstractions that split an encryption into several ones are not covered by [72].

Nevertheless, our work [103] exhibits several drawbacks: (1) like [72], the theory is limited to the free algebra over a fixed signature; (2) all variables have strict (possibly structured) types, hence it is not possible to precisely model ticket forwarding or Diffie-Hellman exchanges. Furthermore, this approach did not include user-defined types and subtyping, which are commonly supported features of modern verifiers. Their absence could make our type system too fine-grained. For example, the type system can distinguish nonces generated by different roles, which appears unrealistic; and (3) some soundness conditions involve quantification over substitutions or terms and are thus hard to check in practice. In this thesis, we address all the limitations above. First, we work with a useful subclass of shallow subterm-convergent rewrite theories modulo a set of axioms to model cryptographic operations. Second, we support untyped variables, user-defined types, and subtyping. User-defined types enable the grouping of similar atomic types (e.g., session keys) and adjusting the granularity of matching in message abstraction. Third, we have separated the removal of variables, atomic messages, and redundancies (new untyped abstractions) from the transformation of the message structure (typed abstractions). This simplifies the specifications and soundness proof of typed abstractions. We have extended the scope of our prior work with additional typed abstractions, e.g., splitting a hash into several ones, or splitting a symmetric key in an encryption. Finally, we provide effectively checkable syntactic criteria for the conditions of the soundness theorem.

On the one hand, our protocol abstractions cover most of those in [103]. Note that unlike [72, 103], we do not need a separate abstraction class for redirecting messages. These abstractions are achieved by combining atom-and-variable removal and splitting abstractions. We have illustrated such a combination to simplify a core version of Kerberos IV protocol in Section 4.5. On the other hand, there are abstractions that may be expressible in [72, 103], but disallowed in our theory. In particular, we cannot remove essential variables from encryptions because (i) removing fields from encryptions is forbiden by typed abstractions, (ii) even if these variables are in clear positions, they cannot be removed by atom-and-variable removal abstractions (see condition (i) in Definition 21), and (iii) redundany removal abstractions cannot removed a variable completely (see condition (i) in Definition 24). We however do not regard this as a restriction, since these variables are often crucial to establish the considered properties, and thus removing them may lead to oversimplification or unsoundness.

Backes et al. [11] formalize authentication protocols and abstractions in the ρ-spi calculus [25]. This work aims at analyzing authentication properties by abstracting challenge-response messages in authentication protocols into challenges or responses with lower levels of security. Informally, levels of security are defined in terms of security labels which describe the capability of reading and writing messages. Security labels are derived from employed cryptographic operations, e.g., public key encryptions are writable by everyone, but readable only by the intended receiver. Their abstraction methodology is similar to our heuristic described in Section 5.1, i.e., both approaches rely on security labels to identify abstractions and these abstractions always weaken the security labels of the original messages. However, their approach is different from ours in several ways.

First, their concrete and abstract protocols are not specified in the same protocol specification language. More precisely, they abstract actual protocols specified in the ρ-spi calculus into abstract protocols specified in the CR calculus which is tailored to reasoning about challenges and responses. As a consequence, their abstractions are not composable. Furthermore, analyzing abstract protocols requires a specialized proof technique. In contrast, our abstraction framework transforms protocols to protocols specified in the same language. This allows us to combine different types of abstractions in different ways. The analysis of abstract protocols directly benefits from tools that are available for the concrete ones, and do not require any special treatment.

Second, their abstraction computations depend on not only the messages to be transformed, but also on the challenge-response interpretation of the messages. However, they do not provide a method for determining challenge/response components and abstraction functions for a given protocol. In their running example, they directly define an abstraction function without discussing how this function is computed. Our abstractions instead work on symbolic terms with clear descriptions of how the transformed terms should look like, which facilitates the automation of our abstraction procedure.

Finally, their soundness results require that session keys circulate only within responses in abstract protocols. However, there are protocols in which a role authenticates the other by sending a nonce encrypted with a session key shared between the roles, and receiving the nonce back in clear. Such mutual key confirmation mechanisms are not captured in their theory.

7.2. Security Protocol Refinement

Guttman [66, 65] studies the preservation of security properties for a rich class of protocol transformations in the strand space model. His approach to property preservation is based on the simulation of protocol analysis steps instead of execution steps. Each analysis step explains the origin of a message. Instead of working on the level of protocol messages, his protocol transformations are applied to strand nodes and then lifted to protocol specifications and security properties. On the one hand, his approach puts no restrictions on the shapes of a transformed protocol message in comparison with its original message, and therefore potentially covers protocol transformations that are not captured in our framework. For example, a transformation that maps a transmission node $\mathsf{send}(\mathsf{h}(\langle X, K\rangle))$ to $\mathsf{send}(\{\!|X|\!\}_K)$ falls into his transformation class, but cannot be justified in our theory since our abstractions generally do not produce new function symbols. On the other hand, his protocol transformations are required to preserve the originated values and the plaintext subterms of node messages. The former condition means that if a value x first occurs in a transmission node n then it also occurs in the corresponding transformed node of n in the first time. These conditions need not hold for our soundness results. For example, we can completely remove fresh values that are in clear or fields in a hash. Such transformations are not supported by his theory. Furthermore, his soundness conditions are highly non-trivial to justify. In particular, the progress preservation for the analysis steps in the original and the refined protocols and the simulation condition on these steps are hard to verify. Even though our approach is incomparable with Guttman's from the theoretical point of view, it is still interesting to investigate the relation between the trace simulation in our approach and the analysis step simulation in his work.

8. Conclusions

8.1. Summary

In this thesis, we propose a set of syntactic protocol abstractions that allows us to abstract realistic protocols and capture a large class of attacks. Unlike previous works in this area, e.g., [103, 72], our theory and soundness results accommodate equational theories, untyped variables, user-defined types, and subtyping. These features allow us to accurately model protocols, capture type-flaw attacks, and adapt to different verification tools, e.g., those supporting equational theories such as ProVerif and CL-atse.

Theoretically, our results allow us to compare different security protocols, e.g., protocol variants in standards, and therefore deepen our understanding of security protocols. Additionally, these results provide useful guidelines for the incremental construction of security protocols. Practically, we provide syntactic criteria for our soundness results that can be efficiently checked. This enables automation of the abstraction mechanism for security protocol verification tools.

From the refinement point of view, our syntactic criteria allows justifying the correctness of protocol refinements at the syntactic level. Thereby, we pave a way for a tool implementation that automatically justifies the correctness of protocol refinements. Such a tool takes as an input a security protocol and its refined specifications, security properties of interest, and a sequence of protocol abstractions that can be used to transform the refined protocol into its abstraction. The tool then checks whether the abstract protocol can be obtained from the refined one using these abstractions. If it is the case, it verifies the required soundness conditions and concludes the correctness of the refinement if these conditions hold. For one-step refinement, the difference between the abstract and the refined protocols is often small and therefore specifying the desired protocol abstractions should not be too hard.

From the abstraction point of view, we provide a heuristic-based abstraction mechanism that can be applied to a wide range of security protocols. Our abstraction method combines different protocol abstractions in order to maximally simplify these protocols for more efficient verification. We have implemented our abstraction mechanism for Scyther and validated its usefulness on various IKE and ISO/IEC 9798 protocols. We also tested our technique (with manually produced abstractions) on ProVerif, CL-atse, OFMC, and SATMC. Our experiments clearly show that modern protocol verifiers with various underlying verification techniques can substantially benefit from our abstractions, which in many cases either enable the completion of verification tasks that have previously hit a resource limit (e.g., for an unbounded number of sessions) or lead to dramatic speedups. Even though our abstraction has been only implemented for Scyther, the abstraction mechanism is not tailored to this particular tool. The notion of security labels and the ideas of generating different abstractions are general and can be applied to other verification tools such as ProVerif and OFMC which support equational theories. Of course, we still need to justify the relation between the protocol models used in different tools and the role-based protocol model for which we establish our soundness results.

8.2. Future Work

In this section, we identify a number of directions for future research.

Equational theories Our approach to handling equational theories relies on two restrictions. First, we only consider decomposable equational theories, i.e., those can be decomposed into a set of convergent rewrite rules R and a set of axioms Ax. Second, we restrict R to the class of shallow subterm-convergent rewrite rules. As a consequence, we cannot treat a theory that models blind signatures or Diffie-Hellman theories where the exponents constitute an abelian group such as the Diffie-Hellman theory defined in [115]. These theories do not satisfy the second restriction. In order to support such theories, a possible approach could be to establish soundness results for these theories separately. One could also relax the subterm-convergence and shallowness assumptions in order to support a larger class of rewrite rules. Our first restriction can also be weakened, e.g., requiring that the considered set of equations E can be partitioned into a set of rewrite rules R and a set of axioms Ax such that the relation $\rightarrow_{R/Ax}$ is convergent, but not necessarily coherent. Under this assumption, $\rightarrow_{R/Ax}$ instead of $\rightarrow_{R,Ax}$ can be used to decide equality modulo E, i.e., for given terms s and t, we have $s =_E t$ if and only if there are terms s' and t' such that $s =_{Ax} s'$, $t =_{Ax} t'$ and $s'{\downarrow_{R/Ax}} = t'{\downarrow_{R/Ax}}$. The main drawback of this approach is that we need to explore all elements of s's Ax-equivalence class to find a term that is reducible via $\rightarrow_R$ which may be infeasible in practice.

Adversary models Our work is currently restricted to the standard Dolev-Yao adversary model. It is therefore desirable to extend our theory to cover stronger adversary models such as compromising adversaries. Linking to this direction, Basin and Cremers [14] formalize different compromising adversaries in the symbolic model. Their security models capture various security properties and adversary capabilities in the literature such as perfect forward secrecy, key compromise impersonation, and adversaries capable of revealing the local state of agents. They model adversary capabilities by a set of adversary rules which are separate from the protocol execution rules. We believe that our soundness results can be extended to these security models by additionally establishing reachability preservation for these adversary rules. In order to record state information in protocol traces, we need to extend our set of protocol events. We essentially can still use our property language since the adversary models are already captured by the operational semantics.

Security protocol models In this thesis, we work with the role-based security protocol model. The main restriction of this model is that we cannot specify protocols that involve loops, e.g., the TESLA protocol [108], or non-monotonic state, e.g., contract signing protocol [7]. Multiset rewriting [30] has been successfully employed to model such protocols. The state-of-the art security protocol verification tool Tamarin [94] is based on multiset rewriting. This motivates a future extension of our soundness results to the multiset rewriting model.

Spurious attack checking As mentioned earlier, our abstraction tool does not check for spurious attacks. We envision the following approach for implementing this functionality and thus completing the automation of the abstraction-refinement process. First, from an attack on the considered security property ϕ in an abstract model, we construct the corresponding set Ev of (symbolic) protocol events occurring in this attack and a partial order on these events. Then, we ask Scyther to search for an attack in the original protocol such that this attack contains only events corresponding to those in Ev and respects their order. If such attack exists then the attack found in the abstract model corresponds to a real one. Thus, we conclude that the original protocol does not satisfy ϕ. Otherwise, the attack is spurious and we have to refine our model. Using the spurious attack checking procedure, we can replace the decision diamond at the bottom of Figure 5.1 by the check whether the attack found is spurious or not. If the attack is spurious then we try the next model in the stack. Otherwise, we conclude that the original protocol does not satisfy the required properties.

TO enable this approach, we need to extend Scyther to support specifying constraints for the attack searching.

Part IV.

Appendix

A. Proofs for Typed Abstractions

A.1. Basic Lemmas the Type System

The subtyping relation respects the types' structures.

Lemma 8. *Let $\tau, \tau' \in \mathcal{Y}$ be such that $\tau \preccurlyeq \tau'$ and $\tau' \neq \mathsf{msg}$. Then either*

(i) τ and τ' are atomic and $\tau \neq \mathsf{msg}$, or

(ii) τ and τ' are composed and there are $n \geq 1$ and $g \in \Sigma^n$ such that $\tau = g(\tau_1, \ldots, \tau_n)$, $\tau' = g(\tau'_1, \ldots, \tau'_n)$, and $\tau_i \preccurlyeq \tau'_i$ for $i \in \widetilde{n}$.

Proof. We prove this lemma by rule induction on the derivation of $\tau \preccurlyeq \tau'$, depending on the last rule R that has been applied.

- $R = \mathrm{S}(\mathsf{msg})$: we have $\tau \in \mathcal{Y}$ and $\tau' = \mathsf{msg}$, contradicting our assumption.
- $R = \mathrm{S}(\preccurlyeq_0)$: we have $\tau \preccurlyeq_0 \tau'$. Then it is clear that both τ and τ' are atomic and $\tau \neq \mathsf{msg}$ by the definition of $\preccurlyeq_0$.
- $R = \mathrm{S}(\mathit{refl})$: we have $\tau = \tau'$ and thus the conclusion holds trivially.
- $R = \mathrm{S}(\mathit{trans})$: here, there is a τ'' such that $\tau \preccurlyeq \tau''$ and $\tau'' \preccurlyeq \tau'$. Since $\tau' \neq \mathsf{msg}$, we derive (i) or (ii) from the induction hypothesis for $\tau'' \preccurlyeq \tau'$ to for τ'' and τ. In both cases, we have $\tau'' \neq \mathsf{msg}$. Therefore, we can also apply the induction hypothesis to $\tau \preccurlyeq \tau''$. Hence, we either have that τ, τ'', and τ' are all atomic and $\tau \neq \mathsf{msg}$ or they all have the same top-level constructor g and the arguments of τ and τ'' and of τ'' and τ' are in the subtyping relation and we conclude by applying $\mathrm{S}(\mathit{trans})$ on the argument types.
- $R = \mathrm{S}(\Sigma^n)$: In this case, the conclusion (ii) follows directly from the rules' premises and conclusions.

□

The following lemma states that well-typed substitutions respect types.

Lemma 9. *Let θ be an R,Ax-normal substitution that is well-typed. Then for all terms $t \in \mathcal{T}$, we have $\Gamma(t\theta) \preccurlyeq \Gamma(t)$.*

Proof. The proof is proceeded by induction on t.

- If t is an atom then $t\theta = t$ and thus the lemma holds trivially.

- If t is a variable X then we distinguish two cases. If $X \notin dom(\theta)$ then we have $X\theta = X$ and this case holds trivially. Otherwise, we have $\Gamma(X\theta) \preccurlyeq \Gamma(X)$, since θ is well-typed and R,Ax-normal.

- If $t = c(t_1, \ldots, t_n)$ for some $c \in \Sigma^n$ and $n \geq 1$ then we have $t\theta = c(t_1\theta, \ldots, t_n\theta)$. Moreover, by induction hypothesis, we have $\Gamma(t_i\theta) \preccurlyeq \Gamma(t_i)$ for all $i \in \widetilde{n}$. This yields $\Gamma(t\theta) \preccurlyeq \Gamma(t)$ as required.

□

A.2. Basic Properties of Typed Abstractions

In this section, we prove several properties of typed abstractions. First, we show that two terms whose types are in a subtyping relation must be transformed by the same clause. Second, we describe the shapes of transformed terms in different cases. At the end, we prove that type inference is preserved under abstractions.

A.2.1. Uniform Matching

The following lemma states that a term t matches a linear pattern p whenever t's type is a subtype of p's type.

Lemma 10. *Let $p \in \mathcal{P}$ be a linear pattern. Then, for all $t \in \mathcal{T}$ such that $\Gamma(t) \preccurlyeq \Gamma(p)$ there exists a substitution $\sigma : vars(p) \to \mathcal{T}$ such that $p\sigma = t$.*

Proof. We prove the lemma by induction on the structure of p. Below we use the abbreviations $\tau = \Gamma(t)$ and $\pi = \Gamma(p)$.

- If p is a pattern variable, then we define $\sigma = \{t/p\}$, hence $p\sigma = t$.

- If $p = g(p_1, \ldots, p_n)$ for $g \in \Sigma^n$, $n \geq 1$ then since $\Gamma(p) = \pi$, there exists $\pi_1, \ldots, \pi_n$ such that
$$\pi = g(\pi_1, \ldots, \pi_n) \text{ and } \Gamma(p_i) = \pi_i \text{ for } i \in \widetilde{n}.$$
Since $\tau \preccurlyeq \pi$, by Lemma 8, we have
$$\tau = g(\tau_1, \ldots, \tau_n) \text{ and } \tau_i \preccurlyeq \pi_i \text{ for } i \in \widetilde{n}.$$
Since $\Gamma(t) = \tau$ and τ is composed, t is not a variable. Therefore, we have
$$t = g(t_1, \ldots, t_n) \text{ and } \Gamma(t_i) = \tau_i \text{ for } i \in \widetilde{n}.$$
Hence, by induction hypothesis, there are $\sigma_i : vars(p_i) \to \mathcal{T}$ such that $t_i = p_i\sigma_i$ for $i \in \widetilde{n}$. Since p is linear, we can thus define $\sigma : vars(p) \to \mathcal{T}$ by $\sigma = \bigcup_{i=1}^{n} \sigma_i$. Hence, we obtain $p\sigma = t$.

This completes the proof of the lemma. □

Lemma 11 (Uniform matching)**.** *Let* $E_f = [f(p_1) = u_1, \ldots, f(p_n) = u_n]$ *and*

$$matches(t) = \{i \in \widetilde{n} \mid \exists\theta.\ t = p_i\theta \ \wedge\ \Gamma(t) \preccurlyeq \Gamma(p_i)\}.$$

Then, for all $t, t' \in \mathcal{T}$ *with* $\Gamma(t') \preccurlyeq \Gamma(t)$*, we have*

(i) $matches(t) \subseteq matches(t')$*,*

(ii) $matches(t) = matches(t') = \{i\}$ *for some* $i \in \widetilde{n}$ *if* $\Gamma(t) \in \Pi_f{\downarrow}$*.*

In particular, $matches(t) = matches(\Gamma(t))$ *for all terms* $t \in \mathcal{T}$*.*

Proof. Let $t, t' \in \mathcal{T}$, $t : \tau$, $t' : \tau'$, and $\tau' \preccurlyeq \tau$. To see (i), suppose $i \in matches(t)$, i.e., $t = p_i\theta$ and $\Gamma(t) \preccurlyeq \Gamma(p_i)$ for some substitution θ. Since $\Gamma(t') \preccurlyeq \Gamma(t)$, we also have $\Gamma(t') \preccurlyeq \Gamma(p_i)$ and hence $i \in matches(t')$. This shows (i).

To see (ii), we first derive $\tau' \in \Pi_f{\downarrow}$ from the assumptions $\tau \in \Pi_f{\downarrow}$ and $\tau' \preccurlyeq \tau$. Therefore, there are $i, j \in \widetilde{n}$ and $\{\pi_i, \pi_j\} \subseteq \Pi_f$ such that $\tau \preccurlyeq \pi_i$ and $\tau' \preccurlyeq \pi_j$. Moreover, i and j are unique since F_f is pattern-disjoint. By Lemma 10 there are substitutions θ and θ' (with domains $vars(p_i)$ and $vars(p_j)$) such that $t = p_i\theta$ and $t' = p_j\theta'$. Hence, $matches(t) = \{i\}$ and $matches(t') = \{j\}$. Using the result in (i) derive $i = j$ as required. □

In the following lemma, we show that typed abstractions respect well-typedness.

Lemma 12. *Let* σ *be an* R,Ax*-normal ground substitution that is well-typed. Then* $f(\sigma)$ *is well-typed.*

Proof. Let $X \in dom(f(\sigma))$. Then we have $X \in dom(\sigma)$ and $f(X) = X$. Since σ is R,Ax-normal, so is $X\sigma$. Let t be a term such that $t =_{Ax} (Xf(\theta)){\downarrow_{R,Ax}}$. We need to show that $\Gamma(t) \preccurlyeq \Gamma(X)$. We consider two cases.

- If $\Gamma(X) = \mathsf{msg}$ then it is trivial that $\Gamma(t) \preccurlyeq \Gamma(X)$.
- If $\Gamma(X) = \tau$ for an atomic type τ, then since σ is well-typed and $X\sigma$ is R,Ax-normal, it follows that $X\sigma$ is an atom. Thus, we have $f(X\sigma) = Xf(\sigma) = X\sigma$. Hence $t = X\sigma$. This implies $\Gamma(t) \preccurlyeq \Gamma(X)$ as required.

This completes the proof of the lemma. □

A.2.2. Shape Lemma and Termination

Lemma 13 (Shape lemma)**.** *If* $t \in \mathcal{T}$ *then the following holds*

(i) If t *is a variable or an atom, then* $f(t) = t$*.*

(ii) If $t = c(t_1, \ldots, t_n)$*,* $c \in \Sigma^n \setminus (\mathsf{C_{Ax}} \cup \mathsf{C_{Key}})$ *then we have*

$$f(c(t_1, \ldots, t_n)) = \langle u_1, \ldots, u_d \rangle$$

for some $d > 0$ *and for all* $i \in \widetilde{d}$*,* u_i *is one of the following forms:*

a) $u_i = c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_n}))$ *such that*

$$\begin{array}{l} \mathit{split}(\overline{v_j}) \subseteq \mathit{split}(t_j) \text{ for all } j \in \widetilde{n},\ c \neq \langle \cdot, \cdot \rangle, \text{ and} \\ c \in \mathsf{C_R} \Rightarrow \overline{v_n} = [t_n] \end{array}$$

b) $u_i = f(v)$ *such that* $\mathit{split}(v) \subseteq \mathit{split}(t_j)$ *for some* $j \in \widetilde{n}$.

Moreover, it holds that

$$\begin{array}{l} \forall j \in \mathit{pp}(c).\ \mathit{split}(t_j) \subseteq \mathit{split}(P(j)) \\ \forall j \in \widetilde{n}.\ P(j) \subseteq \mathit{subs}(t) \setminus \{t\} \end{array}$$

where

$$\begin{array}{rl} P(j) = & \bigcup\{\mathit{set}(\overline{v_j}) \mid c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_n})) \in \{u_1, \ldots, u_d\}\} \\ & \cup\{v \mid f(v) \in \{u_1, \ldots, u_d\}\} \end{array}$$

(iii) If $t = c(t_1, \ldots, t_n)$ *for* $c \in \Sigma^n \cap (\mathsf{C_{Ax}} \cup \mathsf{C_{Key}} \cup \mathsf{C_{R,Ax}})$ *and* $n \geq 1$ *then*

$$f(t) = c(f(t_1), \ldots, f(t_n)).$$

Proof. We prove this lemma by case distinction on the shape of the term $t \in \mathcal{T}$. We know that there exists the first clause $f(p) = u$ in the list E_f^1 such that $\Gamma(t) \preccurlyeq \Gamma(p)$. By Lemma 10, there is a substitution θ such that $p\theta = t$. Hence, by Program 1, we have

$$f(t) = u\theta. \tag{A.1}$$

Case (i) where t is a variable or an atom follows immediately from Program 1 and the definition of E_f^0. Suppose t is composed. We distinguish the following cases.

- $t = c(t_1, \ldots, t_n)$ and $c \in \Sigma^n \setminus (\mathsf{C_{Ax}} \cup \mathsf{C_{Key}} \cup \mathsf{C_{R,Ax}})$: since p is not a pattern variable and $t = p\theta$, we must have $p = c(p_1, \ldots, p_n)$. By Definition 5, we have $f(p) = \langle e_1, \ldots, e_d \rangle$ for some $d > 0$ and for all $i \in \widetilde{d}$, e_i is one of the following forms:
  1. $e_i = c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n}))$ such that

  $$\begin{array}{l} \mathit{set}(\overline{q_j}) \subseteq \mathit{split}(p_j) \text{ for all } j \in \widetilde{n},\ c \neq \langle \cdot, \cdot \rangle, \text{ and} \\ c \in \mathsf{C_R} \Rightarrow \overline{q_n} = [p_n] \end{array}$$

  2. $e_i = f(q)$ such that $q \in \mathit{split}(p_j)$ for some $j \in \widetilde{n}$.

  and it holds that $\forall j \in \mathit{pp}(c).\ \mathit{split}(p_j) \subseteq Q(j)$ where

  $$\begin{array}{rl} Q(j) = & \bigcup\{\mathit{set}(\overline{q_j}) \mid c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n})) \in \{e_1, \ldots, e_d\}\} \\ & \cup\{q \mid f(q) \in \{e_1, \ldots, e_d\}\}. \end{array}$$

  Let $u_i = e_i\theta$ for all $i \in \widetilde{d}$. For each $i \in \widetilde{d}$, we distinguish two cases depending on the shape of e_i.

– If $e_i = c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n}))$ for some vectors $\overline{q_1}, \ldots, \overline{q_n}$, then let $\overline{v_i} = \overline{q_i}\theta$. We obtain that $u_i = c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_n}))$. From the fact that

$$\begin{array}{l} set(\overline{q_j}) \subseteq split(p_j) \text{ for all } j \in \widetilde{n}, \\ c \in \mathsf{C_R} \Rightarrow \overline{q_n} = [p_n], \text{ and } p_j\theta = t_j \text{ for all } j \in \widetilde{n}. \end{array}$$

we derive that $c \in \mathsf{C_R} \Rightarrow \overline{v_n} = [t_n]$. Moreover, we also have

$$split(\overline{v_j}) \subseteq split(t_j) \text{ for all } j \in \widetilde{n}. \tag{A.2}$$

– If $e_i = f(q)$ and $q \in split(p_j)$ for some $j \in \widetilde{n}$, then let $v = q\theta$. We derive that $split(v) \subseteq split(p_j\theta)$. Since $p_j\theta = t_j$, we obtain that

$$split(v) \subseteq split(t_j) \tag{A.3}$$

It remains to show that

$$\begin{array}{l} \forall j \in pp(c).\ split(t_j) \subseteq split(P(j)), \\ \forall j \in \widetilde{n}.\ P(j) \subseteq subs(t) \setminus \{t\}. \end{array}$$

To see the first point, let $j \in pp(c)$. By Definition 5, we have $split(p_j) \subseteq Q(j)$. This implies $split(p_j\theta) \subseteq split(Q(j)\theta)$. Since $p_j\theta = t_j$ and $P_j = Q_j\theta$, we obtain $split(t_j) \subseteq P(j)$ as required. The second point follows immediately from (A.2) and (A.3).

- $t = c(t_1, \ldots, t_n)$ for $c \in \Sigma^n \cap (\mathsf{C_{Ax}} \cup \mathsf{C_{Key}} \cup \mathsf{C_{R,Ax}})$ and $n \geq 1$: Since $p\theta = t$ and p is not a pattern-variable, we must have $p = c(q_1, \ldots, q_n)$ and $q_i\theta = t_i$ for all $i \in \widetilde{n}$. By Definition 5, we have

$$f(p_i) = c(f(q_1), \ldots, f(q_n)).$$

Hence we derive $f(t) = u_i\theta = c(f(t_1), \ldots, f(t_n))$ as required.

This completes the proof of the lemma. □

Proposition 15 (Termination)**.** *The function f defined by Program 1 terminates on all terms $t \in \mathcal{T}$.*

Proof. We prove this by induction on the size of t. If $\Gamma(t)$ is an atom then the termination of $f(t)$ is immediate. If $\Gamma(t)$ is composed then from Lemma 13 we know that that f is called recursively on subterms of t. Hence, these calls terminate by the induction hypothesis. Therefore, $f(t)$ also terminates. This completes the proof of the proposition. □

A.2.3. Lemmas about Splitting and Intruder Deducibility

Lemma 14. *Let $t, u \in \mathcal{T}$ such that $split(u) \subseteq split(t)$. Then we have*

$$split(f(u)) \subseteq split(f(t)).$$

Proof. We proceed by induction on $|u| + |t|$.

- If $|split(u)| + |split(t)| = 2$ then $split(u) \subseteq split(t)$ implies that $u = t$. Thus the lemma holds for this case.
- Now we assume that $|split(u)| + |split(t)| > 2$. There are two cases.
  - If u is not a pair then $split(u) = \{u\}$. Hence we have

    $$u \in split(t) \tag{A.4}$$

    Since $|split(u)| + |split(t)| > 2$, we have $t = \langle u_1, u_2 \rangle$. Hence by Lemma 13, we have $f(t) = \widehat{f}(\overline{v})$ for some vector $\overline{v}$ such that $split(t) = split(\overline{v})$. By (A.4), there is $t' \in set(\overline{v})$ such that $u \in split(t')$. Moreover, we have $|t'| < |t|$. Thus by induction hypothesis, we have

    $$split(f(u)) \subseteq split(f(t')).$$

    Since $split(f(t')) \subseteq split(f(t))$, this implies that

    $$split(f(u)) \subseteq split(f(t)).$$

  - If $u = \langle u_1, u_2 \rangle$ then by Lemma 13, we have that $f(u) = \widehat{f}(\overline{r})$ for some vector $\overline{r}$ of length m such that $split(u) = split(\overline{r})$. Since $split(u) \subseteq split(t)$, we have $split(r_i) \subseteq split(t)$ for all $i \in \widetilde{m}$. Moreover, we also have that $|r_i| < |u|$. Hence by induction hypothesis, we have $split(f(r_i)) \subseteq split(f(t))$. Therefore, we obtain $split(f(u)) \subseteq split(f(t))$ as required.

This completes the proof of the lemma. □

The following lemma is an immediate corollary of Lemma 14.

Corollary 1. *Let $t \in \mathcal{T}$ and $u \in split(t)$. Then we have*

$$f(t) \vdash_E f(u).$$

Lemma 15. *For all $t, u \in \mathcal{T}(V, \Sigma_0)$ and all substitutions θ, $split(t) \subseteq split(u)$ implies that $split(t\theta) \subseteq split(u\theta)$*

Proof. Suppose that $split(t) \subseteq split(u)$ and $v \in split(t\theta)$. To show that $v \in split(u\theta)$ we distinguish two cases:

1. There is some $t' \in split(t)$ that is not a variable and $v = t'\theta$. Then $t' \in split(u)$ and thus t' is not a pair. Since t' is neither a variable nor a pair, we have $v \in split(u\theta)$.

2. There is a variable $X \in \mathit{split}(t)$ such that $v \in \mathit{split}(X\theta)$. Then $X \in \mathit{split}(u)$ and thus $v \in \mathit{split}(u\theta)$.

This completes the proof of the lemma. □

The following lemma shows that if the intruder learns all the transformed components of a term, he can also learn the transformed term.

Lemma 16. *Let $T \cup \{u\} \subseteq \mathcal{T}$. Suppose $T \vdash_E f(t)$ for all $t \in \mathit{split}(u)$. Then $T \vdash_E f(u)$.*

Proof. We prove the lemma by induction on the size of u. If u is not a pair then $\mathit{split}(u) = \{u\}$ and $T \vdash_E f(u)$ follows immediately from the assumption. Otherwise, $u = \langle u_1, u_2 \rangle$. Then, by Lemma 13, we derive that

$$f(u) = \widehat{f}(\overline{r}).$$

for some vector $\overline{r} = [r_1, \ldots, r_m]$ such that $\mathit{split}(\overline{r}) = \mathit{split}(u)$ and $\mathit{set}(\overline{r}) \subseteq \mathit{subs}(u) \setminus \{u\}$. Since u is R,Ax-normal, so are the r_i. Let $i \in \widetilde{m}$. By assumption and since $\mathit{split}(r_i) \subseteq \mathit{split}(u)$, we have $T \vdash_E f(t)$ for all $t \in \mathit{split}(r_i)$. Since $r_i \in \mathit{subs}(u) \setminus \{u\}$, we obtain $T \vdash_E f(r_i)$ from the induction hypothesis. Hence, the desired $T \vdash_E f(u)$ follows from $T \vdash_E f(r_i)$ for all $i \in \widetilde{m}$. □

The following lemma is a consequence of the two previous lemmas.

Lemma 17. *For all terms $t, u \in \mathcal{T}$, we have $\mathit{split}(t) \subseteq \mathit{split}(u)$ implies that $f(u) \vdash_E f(t)$.*

Proof. By Corollary 1, we have $f(u) \vdash_E f(p)$ for all $p \in \mathit{split}(u)$. Moreover, since $\mathit{split}(t) \subseteq \mathit{split}(u)$, we have $f(u) \vdash_E f(q)$ for all $q \in \mathit{split}(t)$. Hence, by Lemma 16, we have $f(u) \vdash_E f(t)$. □

A.3. Substitution Property

Theorem (Substitution property; Justification of Theorem 2)**.** *Let θ be a well-typed and R,Ax-normal substitution and $t \in \mathit{udom}(F_f, \mathit{dom}(\theta))$. Then $f(t\theta) = f(t)f(\theta)$.*

Proof. We prove the theorem by induction on the size of t. Suppose $E_f = [f(p_1) = u_1, \ldots, f(p_n) = u_n]$ and let t be a term such that $t \in \mathit{udom}(F_f, \mathit{dom}(\theta))$. We distinguish three cases.

- If t is a variable and thus $f(t) = t$ using the final identity fall-back clause of E_f^0. It follows that $f(t)f(\theta) = tf(\theta) = f(t\theta)$ as required.
- If $\mathit{vars}(t) \cap \mathit{dom}(\theta) = \emptyset$ then it is clear that $f(t\theta) = f(t) = f(t)f(\theta)$.
- If t is not a variable and $\mathit{vars}(t) \cap \mathit{dom}(\theta) \neq \emptyset$ then t must be composed. Let $t : \tau$ and $t\theta : \tau'$. Then, we have $\tau' \preccurlyeq \tau$ by Lemma 9. Since $t \in \mathit{udom}(F_f, \mathit{dom}(\theta))$ and $\mathit{vars}(t) \cap \mathit{dom}(\theta) \neq \emptyset$, we have $\tau \in \Pi_f{\downarrow}$. Hence, by Lemma 11, t and $t\theta$ are abstracted in the same way.

Thus, in both cases, there exists a unique clause $(f(p) = u) \in E_f$ and substitutions θ' and θ'' such that $p\theta' = t$ and $p\theta'' = t\theta$. Thus, we also have $t\theta = p\theta'' = p(\theta\theta')$. By Program 1 (modulo renamings), we have

$$f(t) = u[f/f_0]\theta' \quad \text{and} \quad f(t\theta) = u[f/f_0](\theta\theta').$$

We distinguish two base cases.

- $u = p$ and $t = a$ is an atom. Then we obtain that $f(a\theta) = f(a) = a = af(\theta) = f(a)f(\theta)$ as required.
- $u = p$ and $t = X$ is a variable. Then we have $f(X) = X$. Let us consider two cases:
  * If $X \in dom(\theta)$, then we have
  $$f(X\theta) = Xf(\theta) = f(X)f(\theta).$$
  * If $X \notin dom(\theta)$, then since $dom(f(\theta)) = dom(\theta)$, we have $X \notin dom(f(\theta))$. Hence, we have
  $$f(X\theta) = f(X) = X = Xf(\theta) = f(X)f(\theta).$$

For the inductive cases, note that recursive calls of f have subterms of t as arguments by Lemma 13. Moreover, since $t \in udom(F_f, dom(\theta))$, we also have $t' \in udom(F_f, dom(\theta))$ for each term t' occurring as the argument of a recursive call of f in the computation of $f(t)$. This enables the application of the induction hypotheses (IH) below. We distinguish the following cases.

- $p = c(p_1, \ldots, p_n)$ for $c \in \Sigma^n \setminus (\mathsf{C}_{\mathsf{Ax}} \cup \mathsf{C}_{\mathsf{Key}})$, $n \geq 1$. In this case, we have
  $$f(c(p_1, \ldots, p_n)) = \langle e_1, \ldots, e_d \rangle$$
  for some $d > 0$ and for all $i \in \widetilde{d}$, e_i is one of the following forms:
  1. $e_i = c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n}))$ such that
  $$\begin{array}{l} set(\overline{q_j}) \subseteq split(p_j) \text{ for all } j \in \widetilde{n},\ c \neq \langle \cdot, \cdot \rangle, \text{ and} \\ c \in \mathsf{C}_{\mathsf{R}} \Rightarrow \overline{q_n} = [p_n] \end{array}$$
  2. $e_i = f(q)$ such that $q \in split(p_i)$ for some $j \in \widetilde{n}$.

  and it holds that $\forall j \in pp(c).\ split(p_j) \subseteq Q(j)$ where
  $$\begin{array}{rl} Q(j) = & \bigcup\{set(\overline{q_j}) \mid c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n})) \in \{e_1, \ldots, e_d\}\} \\ & \cup\{q \mid f(q) \in \{e_1, \ldots, e_d\}\}. \end{array}$$
  Therefore, we have
  $$\begin{array}{rcl} f(t) & = & \langle e_1\theta', \ldots, e_d\theta' \rangle, \\ f(t\theta) & = & \langle e_1(\theta\theta'), \ldots, e_d(\theta\theta')) \rangle. \end{array}$$
  This implies $f(t)f(\theta) = \langle e_1\theta' f(\theta), \ldots, e_d\theta' f(\theta) \rangle$. To see that $f(t\theta) = f(t)f(\theta)$, it is sufficient to show that $e_i(\theta\theta') = e_i\theta' f(\theta)$ for all $i \in \widetilde{d}$. Let $i \in \widetilde{d}$, we distinguish two cases.

* If $e_i = c(\widehat{f}(\overline{q_1}), \dots, \widehat{f}(\overline{q_n}))$ then we have

$$e_i(\theta\theta') = c(\widehat{f}(\overline{q_1}(\theta\theta')), \dots, \widehat{f}(\overline{q_n}(\theta\theta'))).$$

Since $set(\overline{q_j}) \subseteq split(p_j)$ for all $j \in \widetilde{n}$, we derive that $set(\overline{q_j}\theta') \subseteq subs(p_j\theta')$ for all $j \in \widetilde{n}$. Moreover, we have $p_j\theta' \in subs(t) \setminus \{t\}$. Therefore, by induction hypothesis, we derive that $\widehat{f}(\overline{q_j}(\theta\theta')) = \widehat{f}(\overline{q_j}\theta')f(\theta)$ for all $j \in \widetilde{n}$. Note that for all $j \in \widetilde{n}$, we have $\widehat{f}(\overline{q_j}\theta') = \widehat{f}(\overline{q_j})\theta'$. Hence, we conclude that $e_i(\theta\theta') = e_i\theta' f(\theta)$ as desired.

– $p = c(p_1, \dots, p_n)$ for $c \in \Sigma^n \cap (\mathsf{C}_{\mathsf{Ax}} \cup \mathsf{C}_{\mathsf{Key}})$ and $n \geq 1$. In this case, we have

$$\begin{array}{rcl} f(p) & = & c(f(p_1), \dots, f(p_n)), \\ f(t) & = & c(f(p_1\theta'), \dots, f(p_n\theta')). \end{array}$$

Let $v_j = p_j\theta'$ for $j \in \widetilde{n}$. We prove this case by the following calculation:

$$\begin{array}{lll} f(t\theta) & = g(f(v_1\theta), \dots, f(v_n\theta)) & \\ & = g(f(v_1)f(\theta), \dots, f(v_n)f(\theta)) & \text{by IH} \\ & = g(f(v_1), \dots, f(v_n))f(\theta) & \\ & = f(t)f(\theta) & \end{array}$$

This completes the proof of the theorem. □

A.4. Deducibility Preservation

Notation. For the sake of a lighter notation, we will omit set braces in intruder derivations and write, e.g., $T, t \vdash_E u$ instead of $T \cup \{t\} \vdash_E u$ for a set of terms T and individual terms t and u. We also write $T \vdash_E U$ for a set of terms U to mean that all terms in U are derivable from those in T. In the remains of this section, we assume that F_f is Ax-closed.

A.4.1. Preservation Results for Equality and Reduction

Proposition 16 (Ax-equality preservation)**.** *Let $t, u \in \mathcal{T}$. Then $t =_{Ax} u$ implies that $f(t) =_{Ax} f(u)$.*

Proof. We prove this by induction on the size of the derivation $t =_{Ax} u$ depending on the last rule that has been applied.

- Reflexivity: In this case, we have $t = u$. Thus the lemma holds trivially.
- Axiom: In this case, there are a pair $\{s_1, s_2\} \in Ax$ and a substitution σ such that $t = s_1\sigma$ and $u = s_2\sigma$. By point (ii) in Definition 5, we know that f is homomorphic for $ct(s_1) \cup ct(s_2)$. Therefore, we derive

$$\begin{array}{l} f(t) = f(s_1)f(\sigma) = s_1 f(\sigma), \\ f(u) = f(s_2)f(\sigma) = s_2 f(\sigma). \end{array}$$

Thus by rule Axiom, we obtain that $s_1 f(\sigma) =_{Ax} s_2 f(\sigma)$. Therefore, we have $f(t) =_{Ax} f(u)$.

- Congruence: Suppose that $t = c(t_1, \ldots, t_n)$ for some $c \in \Sigma^n$. Then we have $u = c(u_1, \ldots, u_n)$ for some terms u_i, $i \in \widetilde{n}$ and

$$t_i =_{Ax} u_i \text{ for all } i \in \widetilde{n}. \tag{A.5}$$

Moreover, since F_f is Ax-closed, we know that t and u match the same clauses $f(p) = q$ in E_f^+. Hence there are substitutions θ, θ' such that $t = p\theta$ and $u = p\theta'$. We consider different shapes of p.

  - $p = c(p_1, \ldots, p_n)$ for $c \in \Sigma^n \setminus (\mathsf{C}_{\mathsf{Ax}} \cup \mathsf{C}_{\mathsf{Key}})$ and $n \geq 1$. In this case, we have

$$f(p) = \langle e_1, \ldots, e_d \rangle$$

for some $d > 0$ for all $i \in \widetilde{d}$, e_i is one of the following forms:

1. $e_i = c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n}))$ such that

$$\begin{array}{l} set(\overline{q_j}) \subseteq split(p_j) \text{ for all } j \in \widetilde{n},\ c \neq \langle \cdot, \cdot \rangle, \text{ and} \\ c \in \mathsf{C}_{\mathsf{R}} \Rightarrow \overline{q_n} = [p_n]. \end{array}$$

2. $e_i = f(q)$ such that $q \in split(p_i)$ for some $j \in \widetilde{n}$.

and it holds that $\forall j \in pp(c).\ split(p_j) \subseteq Q(j)$ where

$$\begin{array}{rl} Q(j) = & \bigcup \{ set(\overline{q_j}) \mid c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n})) \in \{e_1, \ldots, e_d\} \} \\ & \cup \{ q \mid f(q) \in \{e_1, \ldots, e_d\} \}. \end{array}$$

Hence we have

$$\begin{array}{rcl} f(t) & = & \langle e_1\theta, \ldots, e_d\theta \rangle, \\ f(u) & = & \langle e_1\theta', \ldots, e_d\theta' \rangle. \end{array}$$

To see that $f(t) =_{Ax} f(u)$, it is sufficient to show that $e_i\theta =_{Ax} e_i\theta'$ for all $i \in \widetilde{d}$. Let $i \in \widetilde{d}$. We distinguish two cases depending on the shape of e_i.

    * If $e_i = c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n}))$ then let $j \in \widetilde{n}$ and $e \in split(p_j)$. By (A.5), we know that $t_j =_{Ax} u_j$. Hence, we have $p_j\theta =_{Ax} p_j\theta'$. By Definition 1, we derive that there must be a sub-derivation $e\theta =_{Ax} e\theta'$. Hence, by induction hypothesis, we know that $f(e\theta) =_{Ax} f(e\theta')$. Moreover, we have $set(\overline{q_j}) \subseteq split(p_j)$. Therefore, we obtain $e_i\theta =_{Ax} e_i\theta'$ as desired.
    * If $e_i = f(q)$ where $q \in split(p_j)$ for some $j \in \widetilde{n}$, then by a similar reasoning as before, we derive that there is a sub-derivation $q\theta =_{Ax} q\theta'$. By induction hypothesis, we have $f(q\theta) =_{Ax} f(q\theta')$ which implies $e_i\theta =_{Ax} e_i\theta'$ as desired.

– $p = c(p_1, \ldots, p_n)$ for $c \in \Sigma^n \cap (\mathsf{C_{Ax}} \cup \mathsf{C_{Key}})$, $n \geq 1$. Thus, we have

$$\begin{array}{rcl} f(p) & = & c(f(p_1), \ldots, f(p_n)), \\ f(t) & = & c(f(t_1), \ldots, f(t_n)), \\ f(u) & = & c(f(u_1), \ldots, f(u_n)). \end{array}$$

By the induction hypothesis, we have $f(t_i) =_{Ax} f(u_i)$ for all $i \in \widetilde{n}$. It follows that $f(t) =_{Ax} f(u)$ as required.

- Transitivity: In this case, there is a term t' such that $t =_{Ax} t'$ and $t' =_{Ax} u$. By induction hypothesis, we have $f(t) =_{Ax} f(t')$ and $f(t') =_{Ax} f(u)$. Therefore, we have $f(t) =_{Ax} f(u)$.

This completes the proof of the lemma. $\square$

Proposition 17 (Reduction preservation)**.** *Let $t \in rdom(F_f)$ be a ground term. Suppose $t \rightarrow_{R,Ax} u$ for some term u. Then $f(t) \rightarrow^*_{R,Ax} f(u)$.*

Proof. Suppose $t \in rdom(F_f)$ and $t \rightarrow_{R,Ax} u$ for some term u. Then there are a position p in t, a rewrite rule $l \rightarrow r \in R$, and substitution σ such that $t|_p =_{Ax} l\sigma$ and $t \rightarrow_{R,Ax} t[r\sigma]_p = u$. Let $t' = t[Z]_p$ for some fresh variable Z such that $Z\colon \mathsf{msg}$ and $Z \notin vars(t)$. Then we can write the rewriting step as follows.

$$t = t'\{\, t|_p \,/Z\} =_{Ax} t'\{l\sigma/Z\} \quad \rightarrow_{R,Ax} \quad t'\{r\sigma/Z\} = u$$

By Proposition 16, we have $f(t|_p) =_{Ax} f(l\sigma)$. Assuming without loss of generality that $t|_p$ and l have disjoint sets of variables, σ is also a R, Ax-unifier of $t|_p$ and l. Therefore, $t|_p$ is l-reducible. Since $t \in rdom(F_f)$, we know from point (i) of Definition 10 that f is homomorphic for $ct(l)$ and from point (ii) that $t' \in udom(F_f, \{Z\})$. Moreover, since r is either a variable or a constant, we have $f(r\sigma) = rf(\sigma)$. Using Theorem 2, we justify the following rewriting step.

$$f(t) = f(t')\{f(t|_p)/Z\} =_{Ax} f(t')\{lf(\sigma)/Z\} \quad \rightarrow^*_{R,Ax} \quad f(t')\{rf(\sigma)/Z\} = f(u)$$

Hence, we have established $f(t) \rightarrow^*_{R,Ax} f(u)$. $\square$

Lemma 18. *Let θ be a well-typed substitution, V be a set of variables, and t be a term such that $vars(t) \cap V \neq \emptyset$ and $t \in udom(F_f, V)$. Let u be a term such that $u \in Rec(F_f, V, t)$ and $vars(u) \cap V \nsubseteq dom(\theta)$. Then, we have $u\theta \in Rec(F_f, V \setminus dom(\theta), t\theta)$.*

Proof. We prove this lemma by induction on the computation of $f(t)$.

- If t is a variable or an atom then since $u \in Rec(F_f, V, t)$, we have $u = t$ and $u \in V$. Since $vars(u) \cap V \nsubseteq dom(\theta)$, we have $u \notin dom(\theta)$. Hence, we have $u \in V \setminus dom(\theta)$. This implies that $u\theta \in Rec(F_f, V \setminus dom(\theta), t\theta)$.
- If t is composed then we consider two cases:
  – If $u = t$ then the lemma holds trivially.

– If $u \neq t$ then since $t \in \mathit{udom}(F_f, V)$ and $\mathit{vars}(t) \cap V \neq \emptyset$, there is a clause $f(p) = q \in E_f$ such that $\Gamma(t) \preccurlyeq \Gamma(p)$. Since $u \in \mathit{Rec}(F_f, V, t)$, there must be a strict subterm t' of t such that $f(t') \in \mathit{subs}(q)$ and $u \in \mathit{Rec}(F_f, V, t')$. Since θ is well-typed, we know that $\Gamma(t\theta) \preccurlyeq \Gamma(t)$. Hence, we derive that $t\theta$ is transformed by the clause $f(p) = q$. Since $\mathit{vars}(u) \cap V \not\subseteq \mathit{dom}(\theta)$ and $\mathit{vars}(u) \subseteq \mathit{vars}(t')$, we derive that $\mathit{vars}(t') \cap V \not\subseteq \mathit{dom}(\theta)$. It follows that

$$t'\theta \in \mathit{Rec}(F_f, V \setminus \mathit{dom}(\theta), t\theta). \tag{A.6}$$

Since $\mathit{vars}(t') \cap V \not\subseteq \mathit{dom}(\theta)$, we have $\mathit{vars}(t') \cap V \neq \emptyset$. Moreover, since $t \in \mathit{udom}(F_f, V)$, we derive that $t' \in \mathit{udom}(F_f, V)$. Hence by induction hypothesis, we have

$$u\theta \in \mathit{Rec}(F_f, V \setminus \mathit{dom}(\theta), t'\theta). \tag{A.7}$$

By (A.6) and (A.7), we obtain that $u\theta \in \mathit{Rec}(F_f, V \setminus \mathit{dom}(\theta), t\theta)$ as required.

This completes the proof of the lemma. □

Lemma 19. *Let t be a term, θ be a well-typed ground substitution, and V be a set of variables such that $V \cap \mathit{vars}(t) \neq \emptyset$ and $t \in \mathit{udom}(F_f, V)$. Then, we have $t\theta \in \mathit{udom}(F_f, V \setminus \mathit{dom}(\theta))$.*

Proof. Let $u \in \mathit{Rec}(F_f, V \setminus \mathit{dom}(\theta), t\theta)$. Then, we have $\mathit{vars}(u) \cap V \not\subseteq \mathit{dom}(\theta)$. We need to show that $\Gamma(u) \in \Pi_f{\downarrow} \cup \mathcal{Y}_{at}$. Since $t \in \mathit{udom}(F_f, V)$ and $\mathit{vars}(t) \cap V \neq \emptyset$, there must exist the minimal subterm t' of t such that $t' \in \mathit{Rec}(F_f, V, t)$ and $u \in \mathit{Rec}(F_f, V \setminus \mathit{dom}(\theta), t'\theta)$. Since $\mathit{vars}(u) \cap V \not\subseteq \mathit{dom}(\theta)$, we derive that $\mathit{vars}(t') \cap V \not\subseteq \mathit{dom}(\theta)$. Hence, by Lemma 18, we have $t'\theta \in \mathit{Rec}(F_f, V \setminus \mathit{dom}(\theta), t\theta)$. We consider the following cases.

- If $t'\theta = u$ then since $t' \in \mathit{Rec}(F_f, V, t)$, we know that $\Gamma(t') \in \Pi_f{\downarrow} \cup \mathcal{Y}_{at}$. This together with the well-typedness of θ implies that $\Gamma(u) \in \Pi_f{\downarrow} \cup \mathcal{Y}_{at}$.
- If $t'\theta \neq u$ then let $f(p) = q$ be the clause that transforms t' and σ be a substitution such that $p\sigma = t'$. Since $u \in \mathit{Rec}(F_f, V \setminus \mathit{dom}(\theta), t'\theta)$, there must be $f(q') \in \mathit{subs}(q)$ such that $u \in \mathit{Rec}(F_f, V \setminus \mathit{dom}(\theta), q'(\theta \uplus \sigma))$. Then, we derive that

$$\mathit{vars}(q'(\theta \uplus \sigma)) \cap V \not\subseteq \mathit{dom}(\theta).$$

  This means $\mathit{vars}(q'\sigma) \cap V \neq \emptyset$ and thus $q'\sigma \in \mathit{Rec}(F_f, V, t)$. Note that $q'\sigma$ is a strict subterm of t'. This contradicts the minimality of t' and therefore concludes this case.

□

Lemma 20. *Let $t \in \mathit{rdom}(F_f)$ be a ground term and p be a position such that all strict subterms of $t|_p$ are R,Ax-normal. Suppose $t =_{Ax} t[l\sigma]_p \to_{R,Ax} t[r\sigma]_p = u$ for some rewrite rule $l \to r \in R$ and some substitution σ. Then $u \in \mathit{rdom}(F_f)$.*

Proof. To show that $u \in \mathit{rdom}(F_f)$, we need to show two points.

(i) for all non-variable positions $p' \in \mathcal{P}os(u)$ such that $u|_{p'}$ is l'-reducible, we have that f is homomorphic for $ct(l')$,

(ii) for all non-empty sets of non-variable positions $\{p_1, \ldots, p_n\} \subseteq \mathcal{P}os(u)$ such that
- p_i and p_j are incomparable with respect to $\sqsubseteq$ for all $i, j \in \widetilde{n}$ and $i \neq j$,
- $u|_{p_i}$ is l_i-reducible for all $i \in \widetilde{n}$,

we have that $u[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n} \in \mathit{udom}(F_f, \{Z_1, \ldots, Z_n\})$ for some fresh variables $Z_1, \ldots, Z_n$ such that $Z_i \notin \mathit{vars}(u) \cup \bigcup_{1 \leq j \leq n} \mathit{vars}(l_j)$ and $Z_i\colon \mathsf{msg}$ for all $i \in \widetilde{n}$.

To show these points, we first prove the following claim.

Claim 1. *For all non-variable positions $p' \in \mathcal{P}os(u)$ such that $u|_{p'}$ is l'-reducible for some rule $l' \to r' \in R$, then we have that*

(a) p is not a prefix of p' and $p' \in \mathcal{P}os(t)$, and

(b) $t|_{p'}$ is l'-reducible.

Let $u' = u|_{p'}$ and suppose that $p \sqsubseteq p'$. Then since u is ground and u' is l'-reducible, we know that there is a term $v \in \mathit{subs}(u')$ such that v can be rewritten at the root using some rule $l' \to r'$. Note that $v \in \mathit{subs}(t|_p)$ since R is subterm-convergent. This means $t|_p$ is not R,Ax-normal which contradicts our assumption. Hence, we have that p is not a prefix of p'. This implies $p' \in \mathcal{P}os(t)$ and thus (a) holds. We now show (b) by induction on $|u'|$:

- If p and p' are incomparable with respect to $\sqsubseteq$ then we have $u' = t|_{p'}$. Since u' is l'-reducible, so is $t|_{p'}$. The lemma therefore holds for this case.
- If $p' \sqsubset p$ then since u' is l'-reducible, there are two possibilities.
  - There is a well-typed Ax-unifier σ' of l' and u'. Then let Z be a fresh variable such that $Z \notin \mathit{vars}(u) \cup \mathit{vars}(l') \cup \mathit{vars}(l)$ and $Z : \mathsf{msg}$. Then, we have $u'\sigma' =_{Ax} l'\sigma'$. Let p'' be such that $p = p' \cdot p''$. Thus, we obtain $u'[Z]_{p''}\theta =_{Ax} l'\theta$, where $\theta = \sigma' \cup \{u|_p\ /Z\}$. Note that $u'[Z]_{p''} = (t|_{p'})[Z]_{p''}$. Hence, we have $(t|_{p'})[Z]_{p''}\theta =_{Ax} l'\theta$. By Definition 9(ii), we derive that $t|_{p'}$ is l'-reducible.
  - There are a non-empty set of non-variable positions $\{p_1, \ldots, p_n\} \subseteq \mathcal{P}os(u') \setminus \{\epsilon\}$ such that they are incomparable with respect to $\sqsubseteq$ and $u'|_{p_i}$ is l_i-reducible, a well-typed substitution σ', and some fresh variables $Z_1, \ldots, Z_n$ such that

$$Z_i \notin \mathit{vars}(u') \cup \bigcup_{1 \leq j \leq n} \mathit{vars}(l_j) \cup \mathit{vars}(l') \text{ and } Z_i\colon \mathsf{msg} \text{ for all } i \in \widetilde{n}, \text{ and}$$

$$u'[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n}\sigma' =_{Ax} l'\sigma'. \tag{A.8}$$

  Then by (a), we have p is not a prefix of $p' \cdot p_i$ for all $i \in \widetilde{n}$. We therefore have two cases:

- $*$ There is some $p_i \in \{p_1, \ldots, p_n\}$ such that $p' \cdot p_i \sqsubseteq p$. Then since $p_1, \ldots, p_n$ are pairwise incomparable with respect to $\sqsubseteq$, we know that i is unique. Thus we have

$$u'[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n} = (t|_{p'})[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n}.$$

Moreover, by induction hypothesis, we know that $t|_{p' \cdot p_i}$ is l_i-reducible. This together with (A.8) and Definition 9(ii), we derive that $t|_{p'}$ is l'-reducible.

- $*$ $p' \cdot p_i$ and p are incomparable with respect to $\sqsubseteq$ for all $i \in \widetilde{n}$. Let $Z_{n+1} \notin vars(u') \cup \bigcup_{1 \leq j \leq n} vars(l_j) \cup vars(l')$ be a fresh variable such that $Z_{n+1} \notin \{Z_1, \ldots, Z_n\}$. Let $p'' \in \mathcal{P}os(u')$ such that $p = p' \cdot p''$. We define $\theta = \sigma' \cup \{u|_p / Z_{n+1}\}$ and $p_{n+1} = p''$. From (A.8), we derive that

$$u'[Z_1, \ldots, Z_{n+1}]_{p_1,\ldots,p_{n+1}}\theta =_{Ax} l'\theta. \tag{A.9}$$

Note that $p' \cdot p_i \in \mathcal{P}os(t)$ for all $i \in \widetilde{n+1}$. Therefore, we have

$$u'[Z_1, \ldots, Z_{n+1}]_{p_1,\ldots,p_{n+1}} = (t|_{p'})[Z_1, \ldots, Z_{n+1}]_{p_1,\ldots,p_{n+1}}.$$

This by (A.9) and Definition 9(ii) implies that $t|_{p'}$ is l'-reducible.

Thus, we have shown Claim 1. As a consequence, point (i) immediately follows from Claim 1 and the fact that $t \in rdom(F_f)$.

To show point (ii), let $\{p_1, \ldots, p_n\} \subseteq \mathcal{P}os(u)$ be a non-empty set of non-variable positions such that

- p_i and p_j are incomparable with respect to $\sqsubseteq$ for all $i, j \in \widetilde{n}$ and $i \neq j$,
- $u|_{p_i}$ is l_i-reducible for all $i \in \widetilde{n}$,

Let $Z_1, \ldots, Z_n$ be fresh variables such that

$$Z_i \notin vars(u) \cup \bigcup_{1 \leq j \leq n} vars(l_j) \text{ and } Z_i\colon \mathsf{msg} \text{ for all } i \in \widetilde{n}.$$

We need to show that $u[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n} \in udom(F_f, \{Z_1, \ldots, Z_n\})$. By Claim 1, we know that

$$t|_{p_i} \text{ is } l_i\text{-reducible for all } i \in \widetilde{n}. \tag{A.10}$$

Moreover, we also know that p is not a prefix of p_i for all $i \in \widetilde{n}$. We therefore consider the following cases:

- There is some $p_i \in \{p_1, \ldots, p_n\}$ such that $p_i \sqsubseteq p$. Then since $p_1, \ldots, p_n$ are pairwise incomparable with respect to $\sqsubseteq$, we know that i is unique. Therefore, we have

$$u[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n} = t[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n}.$$

By (A.10) and $t \in rdom(F_f)$, we derive that

$$t[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n} \in udom(F_f, \{Z_1, \ldots, Z_n\}).$$

This implies $u[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n} \in udom(F_f, \{Z_1, \ldots, Z_n\})$ as required.

- p_i and p are incomparable with respect to $\sqsubseteq$ for all $i \in \widetilde{n}$. Then let $Z_{n+1} \notin \mathit{vars}(t) \cup \bigcup_{1 \leq j \leq n} \mathit{vars}(l_j) \cup \mathit{vars}(l)$ be a fresh variable such that $Z_{n+1} \notin \{Z_1, \dots, Z_n\}$ and $p_{n+1} = p$. Since $t|_p$ is l-reducible and $t \in \mathit{rdom}(F_f)$, we derive from (A.10) that
$$t[Z_1, \dots, Z_{n+1}]_{p_1,\dots,p_{n+1}} \in \mathit{udom}(F_f, \{Z_1, \dots, Z_{n+1}\}). \tag{A.11}$$
Let $\theta = \{u|_p \,/Z_{n+1}\}$. Then by Lemma 19, we have that
$$t[Z_1, \dots, Z_{n+1}]_{p_1,\dots,p_{n+1}}\theta \in \mathit{udom}(F_f, \{Z_1, \dots, Z_n\}).$$
This means $u[Z_1, \dots, Z_n]_{p_1,\dots,p_n} \in \mathit{udom}(F_f, \{Z_1, \dots, Z_n\})$, i.e., point (ii) holds.

We have established points (i) and (ii) and thus complete the proof of the lemma. □

Lemma 21. *Let $t \in \mathit{rdom}(F_f)$ be ground. Then $f(t) =_E f(t{\downarrow_{R,Ax}})$.*

Proof. Suppose that $t \rightarrow^*_{R,Ax} u =_{Ax} t\downarrow_{R,Ax}$. By R, Ax-convergence, we can assume without loss of generality that the rewriting in the sequence $t \rightarrow^*_{R,Ax} u$ is done inside out, i.e., all strict subterms of the redexes in the sequence are R,Ax-normal. Hence, using Lemma 20 and Proposition 17 in a routine induction on the length of the rewriting sequence, we can show that $f(t) \rightarrow^*_{R,Ax} f(u)$. Moreover, we have $f(u) =_{Ax} f(t{\downarrow_{R,Ax}})$ by Proposition 16. Hence, we obtain $f(t) =_E f(t{\downarrow_{R,Ax}})$ as required. □

Theorem (Equality preservation; Justification of Theorem 3)**.** *Let t and u be ground terms such that $t, u \in \mathit{rdom}(F_f)$. Then $t =_E u$ implies $f(t) =_E f(u)$.*

Proof. Suppose $t =_E u$. Then $t\downarrow_{R,Ax} =_{Ax} u\downarrow_{R,Ax}$. By Proposition 16, we have that $f(t\downarrow_{R,Ax}) =_{Ax} f(u\downarrow_{R,Ax})$. From Lemma 21, we also have $f(t) =_E f(t\downarrow_{R,Ax})$ and $f(u) =_E f(u{\downarrow_{R,Ax}})$. Hence, we obtain $f(t) =_E f(u)$ as required. □

A.4.2. Preservation Results for Deducibility

Theorem (Deducibility preservation; Justification of Theorem 1)**.** *Let $T \cup \{t\}$ be a set of ground terms such that $\mathcal{C} \subseteq T$ and T is R,Ax-normal. Then $T \vdash_E t$ implies $f(T) \vdash_E f(t{\downarrow_{R,Ax}})$.*

Proof. By induction on the derivation D of $T \vdash_E t$, depending on the last rule that has been applied in this derivation.

- Ax: In this case, we have $t \in T$. Therefore $f(t) \in f(T)$ and thus $f(T) \vdash_E f(t)$. Moreover, since T is R,Ax-normal, so is t. Thus by Proposition 16, we have $f(t) =_{Ax} f(t{\downarrow_{R,Ax}})$. Hence, we obtain $f(T) \vdash_E f(t{\downarrow_{R,Ax}})$ as required.
- Eq: We have $T \vdash_E t'$ and $t' =_E t$. By induction hypothesis, we derive that $f(T) \vdash_E f(t'\downarrow_{R,Ax})$. Moreover, since $t' =_E t$, we derive $t'\downarrow_{R,Ax} =_{Ax} t\downarrow_{R,Ax}$. By Proposition 16, we have $f(t'\downarrow_{R,Ax}) =_{Ax} f(t\downarrow_{R,Ax})$. It follows that $f(T) \vdash_E f(t{\downarrow_{R,Ax}})$ as required.

- Comp: In this case, $t = g(u_1, \ldots, u_m)$ for $g \in \Sigma^m$ and the rule's premises are $T \vdash_E u_i$ for $i \in \widetilde{m}$. The induction hypotheses are $f(T) \vdash_E f(u_i \downarrow_{R,Ax})$ for $i \in \widetilde{m}$. Note that $t \downarrow_{R,Ax} =_{Ax} (g(u_1 \downarrow_{R,Ax}, \ldots, u_m \downarrow_{R,Ax})) \downarrow_{R,Ax}$. Hence without loss of generality, we can assume that u_i are R,Ax-normal for all $i \in \widetilde{m}$. We distinguish two cases depending on whether t is R,Ax-normal or not.

Case 1: t is not R,Ax-normal. Since u_i are R,Ax-normal for all $i \in \widetilde{m}$, some rewrite rule $l \to r \in R$ can be applied at the root position of t, i.e., there is a substitution σ such that $t =_{Ax} l\sigma$. This rule is of one of the following types.

- R4: In this case, we have $t\downarrow_{R,Ax} = a$ for some constant a. Hence $f(t) = f(a) = a$. Since $\mathcal{C} \subseteq T$ and $\mathcal{C} = f(\mathcal{C})$, we have $f(T) \vdash_E f(t\downarrow_{R,Ax})$.
- R3: We have $t \downarrow_{R,Ax} =_{Ax} u_j$ for some $j \in \widetilde{m}$. By Proposition 16, we have $f(t \downarrow_{R,Ax}) =_{Ax} f(u_j)$ which implies $f(T) \vdash_E f(t \downarrow_{R,Ax})$ by the induction hypothesis.
- R1: We have $m = 2$ and

$$\begin{array}{rcl} l & = & g(c(x_1, \ldots, x_{n-1}, t'), u') \to x_j, \\ r & = & x_j \text{ for some } j, 1 \le j \le n-1, \end{array}$$

  where $x_1, \ldots, x_{n-1}$ are variables and t', u' are terms.

  Let $t_i = x_i\sigma$ for all $i, 1 \le i \le n-1$ and $t_n = t'\sigma$. We consider two sub-cases.

  - If $g \in \mathit{top}(Ax)$ then by Definition 5(ii), we know that f is homomorphic for $\mathit{ct}(l)$. Note that t is l-reducible. Since u_i is R,Ax-normal for all $i \in \{1, 2\}$, we derive that $t \in \mathit{rdom}(F_f)$ by Definition 10. Moreover, we have $f(t) = g(f(u_1), f(u_2))$. Let $i \in \{1, 2\}$. We have $f(T) \vdash_E f(u_i \downarrow_{R,Ax})$. Since $u_i \downarrow_{R,Ax} =_{Ax} u_i$, by Proposition 16, we know that $f(u_i \downarrow_{R,Ax}) =_{Ax} f(u_i)$. Therefore, we obtain $f(T) \vdash_E f(u_i)$. This implies

$$f(T) \vdash_E f(t). \tag{A.12}$$

    Since $t \in \mathit{rdom}(F_f)$, by Lemma 21, we have $f(t) =_E f(t \downarrow_{R,Ax})$. This together with (A.12) implies $f(T) \vdash_E f(t\downarrow_{R,Ax})$.
  - If $g \notin \mathit{top}(Ax)$ then let $r = c(t_1, \ldots, t_{n-1}, t_n)$. Since $t =_{Ax} l\sigma$, we derive that

$$\begin{array}{rcl} u_1 & =_{Ax} & r, \\ u_2 & =_{Ax} & u'\sigma. \end{array}$$

    Moreover, since u_1 is R,Ax-normal, so is t_j. Therefore, we have $t\downarrow_{R,Ax} =_{Ax} t_j$. By Proposition 16, we know that $f(t\downarrow_{R,Ax}) =_{Ax} f(t_j)$. Therefore, to see that $f(T) \vdash_E f(t\downarrow_{R,Ax})$, it is sufficient to show that $f(T) \vdash_E f(t_j)$.

    From the induction hypothesis, we know that $f(T) \vdash_E f(u_1)$. Since $u_1 =_{Ax} r$, by Proposition 16, we have $f(u_1) =_{Ax} f(r)$. Hence, we derive that $f(T) \vdash_E f(r)$. By Lemma 13, we have

$$f(r) = \langle w_1, \ldots, w_d \rangle$$

for some $d > 0$ for all $i \in \widetilde{d}$, w_i is one of the following forms:

1. $w_i = c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_n}))$ such that

$$\begin{array}{l} split(\overline{v_k}) \subseteq split(t_k) \text{ for all } k \in \widetilde{n},\ c \neq \langle\cdot,\cdot\rangle, \text{ and} \\ c \in \mathsf{C_R} \Rightarrow \overline{v_n} = [t_n], \end{array}$$

2. $w_i = f(v)$ where $split(v) \subseteq split(t_k)$ for some $k \in \widetilde{n}$

and it holds that

$$\begin{array}{l} \forall k \in pp(c).\ split(t_k) \subseteq split(P(k)) \\ \forall k \in \widetilde{n}.\ P(k) \subseteq subs(r) \setminus \{r\} \end{array} \tag{A.13}$$

where

$$\begin{array}{rl} P(k) = & \bigcup\{set(\overline{v_k}) \mid c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_n})) \in \{w_1, \ldots, w_d\}\} \\ & \cup\{v \mid f(v) \in \{w_1, \ldots, w_d\}\}. \end{array}$$

By (A.13) and Lemma 17, it is sufficient to establish $f(T) \vdash_E f(e)$ for all $e \in P(j)$ in order to conclude the desired $f(T) \vdash_E f(t_j)$.

Let $e \in P(j)$. Note that there is $k \in \widetilde{d}$ such that one of the following holds.

$$\begin{array}{rcl} w_k & = & c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_n})) \text{ and } e \in set(\overline{v_j}), \\ w_k & = & f(v) \text{ and } e = v. \end{array}$$

Clearly, we can derive $f(T) \vdash_E w_k$ from $f(T) \vdash_E f(r)$ by projections. We have

$$f(T) \vdash_E c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_n})). \tag{A.14}$$

In the second case, we immediately obtain $f(T) \vdash_E f(e)$. In the first case, we know that $\overline{v_n} = [t_n]$ by Lemma 13. Hence, we have

$$\widehat{f}(\overline{v_n}) = f(t_n) = f(t'\sigma). \tag{A.15}$$

Since $u_2 =_{Ax} u'\sigma$, by Proposition 16, we have $f(u_2) =_{Ax} f(u'\sigma)$. By induction hypothesis, we know that $f(T) \vdash_E f(u_2)$. Thus, we derive $f(T) \vdash_E f(u'\sigma)$. Since $ct(t') \cup ct(u') \subseteq \mathsf{C_{Key}}$, we know that f is homomorphic for all $c \in ct(t') \cup ct(u')$. As before, we derive

$$\begin{array}{rcl} f(t_n) & = & t'f(\sigma), \\ f(u'\sigma) & = & u'f(\sigma). \end{array} \tag{A.16}$$

Since $f(T) \vdash_E f(u'\sigma)$, by (A.14), (A.15), and (A.16) we obtain

$$f(T) \vdash_E g(c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_{n-1}}), t'f(\sigma)), u'f(\sigma))).$$

Since $(vars(t') \cup vars(u')) \cap \bigcup_{k=1}^{n-1} x_k = \emptyset$ and $x_p \neq x_q$ for all p, q such that $1 \leq p \neq q \leq n-1$, we can define a substitution θ as follows.

$$\begin{array}{rcl} x_q\theta & = & \widehat{f}(\overline{v_q}) \text{ for } 1 \leq q \leq n-1, \\ x\theta & = & xf(\sigma) \text{ for all } x \in vars(u') \cup vars(t'). \end{array}$$

Then we have $g(c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_{n-1}}), t'f(\sigma)), u'f(\sigma))) = l\theta$. Thus, we derive $g(c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_{n-1}}), t'f(\sigma)), u'f(\sigma))) \to_{R,Ax} \widehat{f}(\overline{v_j})$. This implies that $f(T) \vdash_E \widehat{f}(\overline{v_j})$. Together with $e \in set(\overline{v_j})$ and Lemma 17, we derive that $f(T) \vdash_E f(e)$.

– $\mathsf{R2}$: This case is treated similarly as the case for $\mathsf{R1}$.

Case 2: t is R,Ax-normal. By Proposition 16, we have $f(t) =_{Ax} f(t\downarrow_{R,Ax})$. Hence, it is sufficient to show that $f(T) \vdash_E f(t)$. We do case analysis on g.

– If $g \in \Sigma^n \setminus (\mathsf{C_{Ax}} \cup \mathsf{C_{Key}})$ then by Lemma 13, we know that

$$f(t) = \langle w_1, \ldots, w_d \rangle$$

for some $d > 0$ and for all $i \in \widetilde{d}$, w_i is one of the following forms:

1. $w_i = g(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_m}))$ such that

$$\begin{array}{l} split(\overline{v_k}) \subseteq split(u_k) \text{ for all } k \in \widetilde{m},\ g \neq \langle \cdot, \cdot \rangle, \\ \text{and } g \in \mathsf{C_R} \Rightarrow \overline{v_m} = [u_m]. \end{array}$$

2. $w_i = f(v)$ such that $split(v) \subseteq split(u_k)$ for some $k \in \widetilde{m}$

and it holds that

$$\begin{array}{l} \forall k \in pp(g).\ split(u_k) \subseteq split(P(k)) \\ \forall k \in \widetilde{m}.\ P(k) \subseteq subs(t) \setminus \{t\} \end{array}$$

where

$$\begin{array}{rl} P(k) = & \bigcup\{set(\overline{v_k}) \mid g(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_m})) \in \{w_1, \ldots, w_d\}\} \\ & \cup\{v \mid f(v) \in \{w_1, \ldots, w_d\}\}. \end{array}$$

To see that $f(T) \vdash_E f(t)$, it is sufficient to show that $f(T) \vdash_E w_i$ for all $i \in \widetilde{d}$. Let $i \in \widetilde{d}$. We do a case distinction on the shape of w_i.

* If $w_i = c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_m}))$ then since $split(\overline{v_j}) \subseteq split(u_j)$ for all $j \in \widetilde{m}$, by Lemma 17, we derive that $f(u_j \vdash_E \widehat{f}(\overline{v_j})$ for all $j \in \widetilde{m}$. Moreover, by induction hypothesis, we have $f(T) \vdash_E f(u_j)$ for all $j \in \widetilde{m}$. Therefore, we derive that $f(T) \vdash_E \widehat{f}(\overline{v_j})$ for all $j \in \widetilde{m}$. This implies $f(T) \vdash_E w_i$ as desired.
* If $w_i = f(v)$ for some term v such that $split(v) \subseteq split(u_k)$ for some $k \in \widetilde{m}$. By a similar reasoning as above, we conclude that $f(T) \vdash_E w_i$.

– For the remaining cases, we have $f(t) = g(f(u_1), \ldots, f(u_m))$. Then, we obtain $f(T) \vdash_E f(t)$ immediately from the induction hypothesis.

This completes the proof of the theorem. $\square$

Lemma 22. *Let θ be a well-typed R,Ax-normal ground substitution and $t \in rdom(F_f)$ such that $vars(t) \subseteq dom(\theta)$. Then $t\theta \in rdom(F_f)$.*

Proof. Let $u = t\theta$. To show that $u \in rdom(F_f)$, we need to show two points.

(i) for all non-variable positions $p \in \mathcal{P}os(u)$ such that $u|_p$ is l-reducible, we have that f is homomorphic for $ct(l)$,

(ii) for all non-empty sets of non-variable positions $\{p_1, \ldots, p_n\} \subseteq \mathcal{P}os(u)$ such that
   - p_i and p_j are incomparable with respect to $\sqsubseteq$ for all $i, j \in \widetilde{n}$ and $i \neq j$,
   - $u|_{p_i}$ is l_i-reducible for all $i \in \widetilde{n}$ and some $l_i \to r_i \in R$,

   we have that $u[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n} \in udom(F_f, \{Z_1, \ldots, Z_n\})$ for some fresh variables $Z_1, \ldots, Z_n$ such that $Z_i \notin vars(u) \cup \bigcup_{1 \leq j \leq n} vars(l_j)$ and $Z_i \colon \mathsf{msg}$ for all $i \in \widetilde{n}$.

To show these points, we first prove the following claim.

Claim 2. *For all non-variable positions $p \in \mathcal{P}os(u)$ such that $u|_p$ is l-reducible, we have that $p \in \mathcal{P}os(t)$ and $t|_p$ is l-reducible.*

Let p be a non-variable position in u such that $u|_p$ is l-reducible and $u' = u|_p$. First, we show that p is a non-variable position in t. Since $u|_p \in subs(t\theta)$, there are positions $q \in \mathcal{P}os(t)$ and $q' \in \mathcal{P}os(t|_q\, \theta)$ such that $t|_q$ is a variable and $p = q \cdot q'$. Since θ is R,Ax-normal, so is $t|_q\, \theta$. Moreover, we have that $u' = (t|_q\, \theta)|_{q'}$ and thus $(t|_q\, \theta)|_{q'}$ is l-reducible. Since $t|_q\, \theta$ is ground, there is a term $v \in subs((t|_q\, \theta)|_{q'})$ such that v can be rewritten at the root using some rule $l \to r$. This means $t|_q\, \theta$ is not R,Ax-normal which is a contradiction. Thus we have shown that p is a non-variable position in t. We now show by induction on $|u'|$ that $t|_p$ is l-reducible. By Definition 9, we consider two cases:

- There is a well-typed Ax-unifier σ of u' and l then we have $u'\sigma =_{Ax} l\sigma$. Since u' is ground, we have $u' =_{Ax} l\sigma$. Note that $u|_q = t|_q\, \theta$ for all positions q in t. Hence, we have $t|_p\, \theta =_{Ax} l\sigma$. Assuming without loss of generality that $t|_{p'}$ and l have disjoint sets of variables, we have $\theta \cup \sigma$ is a well-typed Ax-unifier of $t|_p$ and l. This implies $t|_p$ is l-reducible.

- There are a non-empty set of non-variable positions $\{p_1, \ldots, p_n\} \subseteq \mathcal{P}os(u') \setminus \{\epsilon\}$ such that they are pairwise incomparable with respect to $\sqsubseteq$ and $u'|_{p_i}$ is l_i-reducible for all $i \in \widetilde{n}$, a well-typed substitution σ, and some fresh variables $Z_1, \ldots, Z_n$ such that

$$Z_i \notin vars(u') \cup \bigcup_{1 \leq j \leq n} vars(l_j) \cup vars(l) \text{ and } Z_i \colon \mathsf{msg} \text{ for all } i \in \widetilde{n},$$

  and $u'[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n}\sigma =_{Ax} l\sigma$. Without loss of generality, we assume that $dom(\sigma) \cap dom(\theta) = \emptyset$ and $Z_i \notin vars(t)$ for all $i \in \widetilde{n}$. Let $t' = t\,|_p$. Then by induction hypothesis, we know that $p_i \in \mathcal{P}os(t')$ and $t'\,|_{p_i}$ is l_i-reducible for all $i \in \widetilde{n}$. Moreover, we have

$$t'[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n}(\theta \cup \sigma) = u'[Z_1, \ldots, Z_n]_{p_1,\ldots,p_n}\sigma =_{Ax} l\sigma = l(\theta \cup \sigma).$$

  By Definition 9(ii), we derive that t' is l-reducible.

Thus we have proven Claim 2. From this claim and the assumption that $t \in \mathit{rdom}(F_f)$, we immediately obtain point (i).

To see point (ii), let $\{p_1, \ldots, p_n\} \subseteq \mathcal{P}os(u)$ be a non-empty set of non-variables positions such that

- p_i and p_j are incomparable with respect to $\sqsubseteq$ for all $i, j \in \widetilde{n}$ and $i \neq j$,
- $u|_{p_i}$ is l_i-reducible for all $i \in \widetilde{n}$,

By Claim 2, we know that $p_i \in \mathcal{P}os(t)$ and $t|_{p_i}$ is l_i-reducible. Now, let $Z_1, \ldots, Z_n$ be fresh variables such that

$$Z_i \notin \mathit{vars}(t) \cup \mathit{dom}(\theta) \cup \bigcup_{1 \leq j \leq n} \mathit{vars}(l_j) \text{ and } Z_i \colon \mathsf{msg} \text{ for all } i \in \widetilde{n}.$$

Since $t \in \mathit{rdom}(F_f)$, we derive that $t[Z_1, \ldots, Z_n]_{p_1, \ldots, p_n} \in \mathit{udom}(F_f, \{Z_1, \ldots, Z_n\})$. By Lemma 19, we have that $t[Z_1, \ldots, Z_n]_{p_1, \ldots, p_n}\theta \in \mathit{udom}(F_f, \{Z_1, \ldots, Z_n\})$. Since p_i is a non-variable position for all $i \in \widetilde{n}$, we have $u[Z_1, \ldots, Z_n]_{p_1, \ldots, p_n} = t[Z_1, \ldots, Z_n]_{p_1, \ldots, p_n}\theta$. Therefore, we derive that $u[Z_1, \ldots, Z_n]_{p_1, \ldots, p_n} \in \mathit{udom}(F_f, \{Z_1, \ldots, Z_n\})$. This implies point (ii) and thus completes the proof of the lemma. □

Lemma. *Justification of Lemma 1 Let θ be a well-typed R,Ax-normal ground substitution and t be a term such that $t \in \mathit{rdom}(F_f)$. Then, we have*

(i) $t\theta \in \mathit{rdom}(F_f)$,

(ii) *t is ground and $t \rightarrow_{R,Ax} u$ imply $u \in \mathit{rdom}(F_f)$.*

Proof. The proof of this lemma immediately follows from Lemma 22 and Lemma 20. □

A.5. Soundness of Typed Abstractions

Using the deducibility preservation for open terms, we show that each reachable state in the original protocol can be simulated by one in the abstracted protocol.

Theorem 8 (Reachability preservation)**.** *Suppose that*

(i) $f(IK_0) \subseteq IK_0'$,

(ii) $\mathcal{M}_P \subseteq \mathit{udom}(F_f) \cap \mathit{rdom}(F_f)$.

Let (tr, th, σ) be a reachable state of P such that σ is R,Ax-normal. Then we have that $(f(tr), f(th), f(\sigma))$ is a reachable state of $f(P)$.

Proof. Note that $f(\sigma)$ is a well-typed ground substitution by Lemma 12. We now show that $(f(tr), f(th), f(\sigma))$ is reachable in $f(P)$ by induction on the number n of transitions leading to a state (tr, th, σ).

- Base case ($n = 0$): For all $i \in dom(th)$, there exists $R \in dom(P)$ such that $th(i) = (R, P(R))$. Hence we have

$$f(th)(i) = (R, f(P(R))) = (R, f(P)(R)). \tag{A.17}$$

  Since (ϵ, th, σ) is reachable, for all $v \in dom(P)$ and $i \in TID$ we have $v^{\sharp i}\sigma \in \mathcal{A}$. Moreover, we have $v^{\sharp i} f(\sigma) = v^{\sharp i}\sigma$, we also have

$$v^{\sharp i} f(\sigma) \in \mathcal{A}. \tag{A.18}$$

  By (A.17), (A.18) and $f(\epsilon) = \epsilon$, it is obvious that $(f(\epsilon), f(th), f(\sigma))$ is reachable in $f(P)$.

- Inductive case ($n = k + 1$): Suppose (tr', th', σ) is reachable in k steps and there is a transition $(tr', th', \sigma) \to (tr, th, \sigma)$. By induction hypothesis, we have

$$(f(tr'), f(th'), f(\sigma)) \text{ is reachable in } f(P) \tag{A.19}$$

  We consider two cases according to the rule r that has been applied in step $k + 1$.

  - If $r = SEND$ then there exists $i \in TID$ and $R \in dom(P)$ such that

$$\begin{aligned} th'(i) &= (R, \mathsf{send}(pt).tl), \\ tr &= tr' \cdot (i, \mathsf{send}(pt)), \\ th &= th'[i \mapsto (R, tl)]. \end{aligned} \tag{A.20}$$

    By (A.20) we have

$$\begin{aligned} f(tr) &= f(tr') \cdot (i, \mathsf{send}(f(pt))), \\ f(th) &= f(th')[i \mapsto (R, f(tl))]. \end{aligned} \tag{A.21}$$

    By (A.20) we have

$$f(th')(i) = (R, \mathsf{send}(f(pt)) \cdot f(tl)). \tag{A.22}$$

    By (A.22), (A.20), (A.21) and rule $SEND$, we have

$$(f(tr'), f(th'), f(\sigma)) \to (f(tr), f(th), f(\sigma)).$$

    Together with (A.19) this implies that $(f(tr), f(th), f(\sigma))$ is reachable in $f(P)$.

  - If $r = RECV$ then there exists $i \in TID$ and $R \in dom(P)$ such that

$$\begin{aligned} &th'(i) = (R, \mathsf{rcv}(u) \cdot tl), \\ &IK(tr')\sigma, IK_0 \vdash_E u\sigma, \end{aligned} \tag{A.23}$$

    and

$$\begin{aligned} tr &= tr' \cdot (i, \mathsf{rcv}(u)), \\ th &= th'[i \mapsto (R, tl)]. \end{aligned} \tag{A.24}$$

By (A.23) and (A.24) we have

$$f(tr) = f(tr') \cdot (i, \mathsf{rcv}(f(u))),$$
$$f(th) = f(th')[i \mapsto (R, f(tl))].$$

To justify $(f(tr'), f(th'), f(\sigma)) \to (f(tr), f(th), f(\sigma))$, it is sufficient to establish the following two premises of rule $RECV$:

1. $f(th')(i) = (R, \mathsf{rcv}(f(u)) \cdot f(tl))$, which follows from (A.23), and
2. $IK(f(tr'))f(\sigma), IK_0' \vdash_E f(u)f(\sigma)$. This follows from (A.23), Proposition 3, and the fact that $f(IK(tr')) = IK(f(tr'))$.

Together with (A.19), this implies $(f(tr), f(th), f(\sigma))$ is reachable in $f(P)$.

This completes the proof of the theorem. □

Theorem (Soundness; Justification of Theorem 4)**.** *Suppose P, ϕ, and F_f satisfy*

- $f(IK_0) \subseteq IK_0'$,
- *F_f is Ax-closed and pattern-disjoint,*
- $\mathcal{M}_P \subseteq udom(F_f) \cap rdom(F_f)$, *and*
- *ϕ is safe for P and f.*

Then, for all states (tr, th, σ) reachable in P, we have

(i) $(f(tr), f(th), f(\sigma))$ *is a reachable state of* $f(P)$,

(ii) $(tr, th, \sigma) \not\models \phi$ *implies* $(f(tr), f(th), f(\sigma)) \not\models f(\phi)$.

Proof. Suppose that (tr, th, σ) is a reachable state of P. Then point (i) follows from Theorem 8. We establish point (ii) by proving the following generalized statement by induction on the structure of ϕ (which may now contain free thread-id variables).

$$\forall \vartheta.\ (tr, th, \sigma, \vartheta) \not\models \phi \Rightarrow (f(tr), f(th), f(\sigma), \vartheta) \not\models f(\phi).$$

Note that a formula is safe if and only if all its subformulas are safe. The literals form the base cases of the induction. We cover all atoms and their negations (except $secret(m)$) in a single equivalence-based argument, where the right-to-left direction covers the positive literal and the other direction the corresponding negative literal. We remark that $(tr, th, \sigma, \vartheta) \not\models A$ is equivalent to $(tr, th, \sigma, \vartheta) \models \neg A$ for all atoms A (but not for all formulas, since $\mathcal{L}_P$ is not closed under negation).

- $\phi \equiv i = j$ or $\phi \equiv \neg(i = j)$.

$$\begin{array}{ll} & (tr, th, \sigma, \vartheta) \models i = j \\ \Leftrightarrow & \vartheta(i) = \vartheta(j) \\ \Leftrightarrow & (f(tr), f(th), f(\sigma), \vartheta) \models f(i = j). \end{array}$$

- $\phi \equiv m = m'$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \vDash m = m' & \\ \Rightarrow & m\sigma =_E m'\sigma & \\ \Rightarrow & f(m\sigma) =_E f(m'\sigma) & \text{by Theorem 3} \\ \Rightarrow & f(m)f(\sigma) = f(m')f(\sigma) & \text{by Theorem 2} \\ \Leftrightarrow & (f(tr), f(th), f(\sigma), \vartheta) \vDash f(m) = f(m'). & \end{array}$$

- $\phi \equiv \neg(m = m')$.

$$\begin{array}{lll} & (f(tr), f(th), f(\sigma), \vartheta) \vDash f(m) = f(m') & \\ \Rightarrow & f(m)f(\sigma) =_E f(m')f(\sigma) & \\ \Leftrightarrow & f(m)f(\sigma) =_E f(m')f(\sigma) & \\ \Rightarrow & f(m\sigma) =_E f(m'\sigma) & \text{by Theorem 2} \\ \Rightarrow & m\sigma =_E m'\sigma & \text{since } \phi \text{ is safe} \\ \Rightarrow & (tr, th, \sigma, \vartheta) \vDash m = m'. & \end{array}$$

- $\phi \equiv \mathit{role}(i, R)$ or $\phi \equiv \neg\mathit{role}(i, R)$.

$$\begin{array}{ll} & (tr, th, \sigma, \vartheta) \vDash \mathit{role}(i, R) \\ \Leftrightarrow & \exists \mathit{seq} \in \mathit{Evt}^*.\ th(\vartheta(i)) = (R, \mathit{seq}) \\ \Leftrightarrow & \exists \mathit{seq} \in \mathit{Evt}^*.\ f(th)(\vartheta(i)) = (R, f(\mathit{seq})) \\ \Leftrightarrow & (f(tr), f(th), f(\sigma), \vartheta) \vDash \mathit{role}(i, R). \end{array}$$

- $\phi \equiv \mathit{honest}(i, R)$ or $\phi \equiv \neg\mathit{honest}(i, R)$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \vDash \mathit{honest}(i, R) & \\ \Leftrightarrow & R^{\vartheta(i)}\sigma \in \mathcal{A}_H & \\ \Leftrightarrow & R^{\vartheta(i)}f(\sigma) \in \mathcal{A}_H & f \text{ is the identity on } \mathcal{A} \\ \Leftrightarrow & R^{\vartheta(i)}\sigma' \in \mathcal{A}_H & \text{since } R^{\vartheta(i)}f(\sigma) = R^{\vartheta(i)}f(\sigma) \\ \Leftrightarrow & (f(tr), f(th), f(\sigma), \vartheta) \vDash \mathit{honest}(i, R). & \end{array}$$

- $\phi \equiv \mathit{steps}(i, s(m))$ or $\phi \equiv \neg\mathit{steps}(i, s(m))$, where $s \in \{\mathsf{send}, \mathsf{rcv}\}$. We have that

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \vDash \mathit{steps}(i, s(m)) & \\ \Leftrightarrow & (\vartheta(i), s(m^{\sharp\vartheta(i)})) \in tr & \\ \Leftrightarrow & (\vartheta(i), s(f(m)^{\sharp\vartheta(i)})) \in f(tr) & \text{justified below} \\ \Leftrightarrow & (f(tr), f(th), f(\sigma), \vartheta) \vDash \mathit{steps}(i, s(f(m))). & \end{array}$$

  We show the second equivalence. The left-to-right implication holds, since ϕ is safe. For the inverse direction (covering the positive literal $\phi \equiv \mathit{steps}(i, s(m))$), suppose that
$$(\vartheta(i), s(f(m)^{\sharp\vartheta(i)})) \in f(tr).$$
  Then there exists $s(m') \in \mathit{Evt}(\mathcal{M}_P)$ such that $(\vartheta(i), s(m'^{\sharp\vartheta(i)})) \in tr$ and $f(m') = f(m)$. Since ϕ is safe, this implies $m = m'$ and hence $(\vartheta(i), s(m^{\sharp\vartheta(i)})) \in tr$.

- $\phi \equiv (i, s(m)) \prec (j, s'(m'))$ or $\phi \equiv \neg((i, s(m)) \prec (j, s'(m')))$, where $s, s' \in \{\mathsf{send}, \mathsf{rcv}\}$.

$$
\begin{array}{lll}
& (tr, th, \sigma, \vartheta) \vDash (i, s(m)) \prec (j, s'(m')) & \\
\Leftrightarrow & (\vartheta(i), s(m^{\sharp\vartheta(i)})) \prec_{tr} (\vartheta(j), s'(m'^{\sharp\vartheta(j)})) & \\
\Leftrightarrow & (\vartheta(i), s(f(m)^{\sharp\vartheta(i)})) \prec_{f(tr)} (\vartheta(j), s'(f(m')^{\sharp\vartheta(j)})) & \text{justified below} \\
\Leftrightarrow & (f(tr), f(th), f(\sigma), \vartheta) \vDash (i, s(f(m))) \prec (j, s'(f(m'))). &
\end{array}
$$

We show the second equivalence. Note that, the if-direction immediately follows, since ϕ is safe and f is order-preserving for events.

For the only-if direction (covering the case that $\phi \equiv (i, s(m)) \prec (j, s'(m'))$), suppose $(\vartheta(i), s(f(m)^{\sharp\vartheta(i)})) \prec_{f(tr)} (\vartheta(j), s'(f(m')^{\sharp\vartheta(j)}))$. Since f is order-preserving for events, there are $s(u), s'(u') \in Evt(\mathcal{M}_P)$ such that

$$(\vartheta(i), s(u^{\sharp\vartheta(i)})) \prec_{tr} (\vartheta(j), s'(u'^{\sharp\vartheta(j)}))$$

with $f(u) = f(m)$ and $f(u') = f(m')$. Since ϕ is safe, we have $u = m$ and $u' = m'$, completing the proof of this direction.

- $\phi \equiv secret(m)$.

$$
\begin{array}{lll}
& (tr, th, \sigma, \vartheta) \nvDash secret(m) & \\
\Leftrightarrow & IK(tr)\sigma, IK_0 \vdash_E m\sigma & \\
\Rightarrow & f(IK(tr))f(\sigma), IK'_0 \vdash_E f(m)f(\sigma) & \text{by Proposition 3} \\
\Rightarrow & IK(f(tr))f(\sigma), IK'_0 \vdash_E f(m)f(\sigma) & f(IK(tr)) = IK(f(tr))) \\
\Leftrightarrow & (f(tr), f(th), f(\sigma), \vartheta) \nvDash secret(f(m)). &
\end{array}
$$

The inductive cases are routines. This concludes the proof of the theorem. $\square$

A.6. Proof of Well-formedness Preservation

Lemma 23. *Let t be a term, X be a variable, and $F_f = (f, E_f)$ be a typed abstraction such that X is extractable from t. Then X is extractable from $f(t)$.*

Proof. We prove this lemma by induction on $|t|$.

- If t is not composed then $f(t) = t$ and thus the lemma holds in this case.
- If t is composed then since X is extractable from t, we know that there is $k \in pp(top(t))$ such that X is extractable from $t|_k$. Suppose that $t = c(t_1, \ldots, t_n)$ for some $n \geq 1$ and terms $t_1, \ldots, t_n$. By Lemma 13, we consider two cases:
  - $f(t) = \langle u_1, \ldots, u_d \rangle$ for some $d > 0$ and for all $i \in \widetilde{d}$, u_i is one of the following forms:
    1. $u_i = c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_n}))$ such that

$$
\begin{array}{l}
split(\overline{v_j}) \subseteq split(t_j) \text{ for all } j \in \widetilde{n},\ c \neq \langle \cdot, \cdot \rangle, \text{ and} \\
c \in \mathsf{C_R} \Rightarrow \overline{v_n} = [t_n]
\end{array}
$$

2. $u_i = f(v)$ such that $split(v) \subseteq split(t_j)$ for some $j \in \widetilde{n}$.

Moreover, it holds that

$$\begin{array}{l} \forall j \in pp(c).\ split(t_j) \subseteq split(P(j)) \\ \forall j \in \widetilde{n}.\ P(j) \subseteq subs(t) \setminus \{t\} \end{array} \tag{A.25}$$

where

$$\begin{array}{rl} P(j) = & \bigcup\{set(\overline{v_j}) \mid c(\widehat{f}(\overline{v_1}), \ldots, \widehat{f}(\overline{v_n})) \in \{u_1, \ldots, u_d\}\} \\ & \cup\{v \mid f(v) \in \{u_1, \ldots, u_d\}\} \end{array}$$

Since $k \in pp(c)$ and $X \in vars(t_k)$, by (A.25), we derive that there is a term $w \in P(k)$ such that X is extractable from w. From the definition of $P(k)$, we derive that $|w| < |t|$. Hence, by induction hypothesis, we know that X is extractable from $f(w)$. Moreover, we also have that $f(w)$ is extractable from $f(t)$. Therefore X is extractable from $f(t)$.

– $f(t) = c(f(t_1), \ldots, f(t_n))$. Since X is extractable from $f(t_k)$ and $k \in pp(c)$, we derive that X is extractable from $f(t)$.

This completes the proof of the lemma. □

The following proposition states that all typed abstractions preserve well-formedness.

Proposition (Well-formedness preservation; Justification of Proposition 11)**.** *Let P be a well-formed protocol and $F_f = (f, E_f)$ be a typed abstraction. Then $f(P)$ is well-formed.*

Proof. Let e be an event in a role $P(R)$ and $X \in vars(term(f(e)))$ such that $\Gamma(X) \neq \alpha$. Then, we have $X \in vars(term(e))$. Since P is well-formed, there is an event $\mathsf{rcv}(t)$ in $P(R)$ such that $\mathsf{rcv}(t)$ equals or precedes e in $P(R)$ and X is extractable from t. Then, we have that $f(\mathsf{rcv}(t))$ equals or precedes $f(e)$ in $f(P)(R)$. Moreover, we also have that X is extractable from $f(t)$ by Lemma 23. This completes the proof of the proposition. □

B. Proofs for the Syntactic Criteria

Lemma 24. *Let u, t, v be ground terms and a be an atom such that $a \notin subs(u) \cup subs(t)$. Suppose that $u[a/v]_{\simeq_{Ax}} =_{Ax} t[a/v]_{\simeq_{Ax}}$. Then $u \in dom([a/v]_{\simeq_{Ax}})$ if and only if $t \in dom([a/v]_{\simeq_{Ax}})$.*

Proof. By symmetry, it is sufficient to show that $u \in dom([a/v]_{\simeq_{Ax}})$ implies that $t \in dom([a/v]_{\simeq_{Ax}})$. Suppose that $u \in dom([a/v]_{\simeq_{Ax}})$. Then we have $u[a/v]_{\simeq_{Ax}} = a$. Hence, we obtain $t[a/v]_{\simeq_{Ax}} =_{Ax} a$. If $t \in dom([a/v]_{\simeq_{Ax}})$ then we are done. Otherwise, since $a \notin subs(t) \cup subs(Ax)$, there must be a strict subterm t' of t such that $t' \in dom([a/v]_{\simeq_{Ax}})$. This implies a is a strict subterm of $t[a/v]_{\simeq_{Ax}}$. But from $a =_{Ax} t[a/v]_{\simeq_{Ax}}$ and the assumption that $|s| = |s'|$ and s, s' are linear for all $\{s, s'\} \in Ax$, we derive that $t[a/v]_{\simeq_{Ax}}$ must not be composed. This is a contradiction and thus completes the proof of the lemma. $\square$

Lemma 25. *Let u, t, v be ground terms and a be an atom such that $a \notin subs(u) \cup subs(t) \cup subs(Ax)$. Then $u[a/v]_{\simeq_{Ax}} = t[a/v]_{\simeq_{Ax}}$ implies $u =_{Ax} t$.*

Proof. We show this lemma by induction on u.

- If u is an atom we consider two cases.
  - If $u \in dom([a/v]_{\simeq_{Ax}})$ then by Lemma 24, we have that $t \in dom([a/v]_{\simeq_{Ax}})$. This implies $u =_{Ax} v$ and $t =_{Ax} v$. Hence, we obtain $u =_{Ax} t$.
  - If $u \notin dom([a/v]_{\simeq_{Ax}})$ then by Lemma 24, we have that $t \notin dom([a/v]_{\simeq_{Ax}})$. Thus, we have $u[a/v]_{\simeq_{Ax}} = u$ and $t[a/v]_{\simeq_{Ax}} = t$. Therefore, we obtain $u = t$ and hence $u =_{Ax} t$.
- If $u = g(u_1, \ldots, u_n)$ for some $g \in \Sigma^n$ then we consider two cases.
  - If $u \in dom([a/v]_{\simeq_{Ax}})$ then by Lemma 24, we have $t \in dom([a/v]_{\simeq_{Ax}})$. It follows that $t[a/v]_{\simeq_{Ax}} = a$. Together with the assumption that $a \notin subs(t)$, this yields $t \in dom([a/v]_{\simeq_{Ax}})$. Hence, we have $u =_{Ax} t$ as required.
  - If $u \notin dom([a/v]_{\simeq_{Ax}})$ then by Lemma 24, we also have $t \notin dom([a/v]_{\simeq_{Ax}})$. Since $top(t[a/v]_{\simeq_{Ax}}) = g$ and $t \notin dom([a/v]_{\simeq_{Ax}})$, we must have $top(t) = g$. Therefore, we have $t = g(t_1, \ldots, t_n)$. We derive that

    $$\begin{array}{rcl} u[a/v]_{\simeq_{Ax}} & = & g(u_1[a/v]_{\simeq_{Ax}}, \ldots, u_n[a/v]_{\simeq_{Ax}}), \\ t[a/v]_{\simeq_{Ax}} & = & g(t_1[a/v]_{\simeq_{Ax}}, \ldots, t_n[a/v]_{\simeq_{Ax}}). \end{array}$$

    Since $u[a/v]_{\simeq_{Ax}} = t[a/v]_{\simeq_{Ax}}$, we have $u_i[a/v]_{\simeq_{Ax}} = t_i[a/v]_{\simeq_{Ax}}$ for all $i \in \widetilde{n}$. By induction hypothesis, we know that $u_i =_{Ax} t_i$. Therefore, by the Congruence rule, we conclude that $u =_{Ax} t$ as required.

This completes the proof of the lemma. □

Lemma 26. *Let t, u, v be terms and a be an atom such that t, v are ground and $a \notin subs(t) \cup subs(u) \cup subs(Ax)$. Let σ be a ground substitution such that $dom(\sigma) = vars(u)$ and $t[a/v]_{\simeq_{Ax}} = u\sigma$. Then there is a ground substitution σ' such that the following holds.*

(i) $dom(\sigma') = vars(u)$,

(ii) $a \notin subs(ran(\sigma'))$,

(iii) $u\sigma' =_{Ax} t$, *and*

(iv) $\sigma'[a/v]_{\simeq_{Ax}} = \sigma$.

Proof. We prove this lemma by induction on u.

- If u is an atom then we have $u\sigma = u$ and thus $t[a/v]_{\simeq_{Ax}} = u$. Since $a \notin subs(u)$, we must have $t[a/v]_{\simeq_{Ax}} = t = u$. Thus we set σ' to the empty substitution and obtain $u\sigma' = t$. Moreover, we also have $dom(\sigma') = \emptyset = dom(\sigma)$. Hence, it is clear that $\sigma'[a/v]_{\simeq_{Ax}} = \sigma$.

- If $u = X$ is a variable then let σ' be such that $dom(\sigma') = \{X\}$ and $X\sigma' = t$. Then we have $u\sigma' = t$. By assumption, we have $a \notin subs(X\sigma) = subs(ran(\sigma))$. Moreover, since $t[a/v]_{\simeq_{Ax}} = X\sigma$, it follows that $(X\sigma')[a/v]_{\simeq_{Ax}} = X\sigma$. Thus, we also have $\sigma'[a/v]_{\simeq_{Ax}} = \sigma$.

- If $u = g(u_1, \ldots, u_n)$ for some $g \in \Sigma^n$ then since $t[a/v]_{\simeq_{Ax}} = u\sigma$ and a is an atom, there must be terms $t_1, \ldots, t_n$ such that

  $$\begin{array}{rcl} t & = & g(t_1, \ldots, t_n), \\ t[a/v]_{\simeq_{Ax}} & = & g(t_1[a/v]_{\simeq_{Ax}}, \ldots, t_n[a/v]_{\simeq_{Ax}}). \end{array}$$

  Therefore, we have $t_i[a/v]_{\simeq_{Ax}} = u_i\sigma$ for all $i \in \widetilde{n}$. By induction hypothesis, there are ground substitutions $\sigma_1, \ldots, \sigma_n$ such that for all $i \in \widetilde{n}$, we have

  $$\begin{array}{l} a \notin subs(ran(\sigma_i)), \\ dom(\sigma_i) = vars(u_i), \\ \sigma_i[a/v]_{\simeq_{Ax}} = \sigma|_{vars(u_i)}, \text{ and} \\ u_i\sigma_i =_{Ax} t_i. \end{array}$$

  We define σ' such that $dom(\sigma') = vars(u)$ and for all $X \in dom(\sigma')$, $X\sigma' = X\sigma_i$ where $i \in \widetilde{n}$ is the smallest index such that $X \in dom(\sigma_i)$. It is clear that $\sigma'[a/v]_{\simeq_{Ax}} = \sigma$ and $a \notin subs(ran(\sigma'))$. To see that $u\sigma' =_{Ax} t$, it is sufficient to show that for all $i, j \in \widetilde{n}$ and all $X \in dom(\sigma_i) \cap dom(\sigma_j)$, it holds that $X\sigma_i =_{Ax} X\sigma_j$. Let $i, j \in \widetilde{n}$ and $X \in dom(\sigma_i) \cap dom(\sigma_j)$. From the induction hypothesis, we know that $(X\sigma_i)[a/v]_{\simeq_{Ax}} = X\sigma$ and $(X\sigma_j)[a/v]_{\simeq_{Ax}} = X\sigma$. Thus, we have $(X\sigma_i)[a/v]_{\simeq_{Ax}} = (X\sigma_j)[a/v]_{\simeq_{Ax}}$. This by Lemma 25 implies that $X\sigma_i =_{Ax} X\sigma_j$. We therefore conclude this case.

This completes the proof of the lemma. □

Lemma 27. *Let t, u, v be terms and a be an atom such that t, v are ground and $a \notin subs(t) \cup subs(u) \cup subs(Ax)$. Suppose that*

(i) $t[a/v]_{\simeq_{Ax}} =_{Ax} u$, *and*

(ii) $top(v) \notin ct(Ax)$.

Then there is a ground term t' such that $t =_{Ax} t'$ and $t'[a/v]_{\simeq_{Ax}} = u$.

Proof. We prove this lemma by induction on the derivation of $t[a/v]_{\simeq_{Ax}} =_{Ax} u$ depending on the last rule that has been applied.

- Reflexivity: We have $t[a/v]_{\simeq_{Ax}} = u$. Thus, we can pick $t' = t$.
- Axiom: In this case, there are $\{s, s'\} \in Ax$ and a ground substitution σ such that $t[a/v]_{\simeq_{Ax}} = s\sigma$ and $u = s'\sigma$. By Lemma 26, there exists a ground substitutions σ' such that
$$\begin{aligned}&dom(\sigma') = vars(s),\\ &a \notin subs(ran(\sigma')),\\ &s\sigma' =_{Ax} t, \text{ and}\\ &\sigma'[a/v]_{\simeq_{Ax}} = \sigma.\end{aligned}$$
We pick $t' = s'\sigma'$. Since $vars(s) = vars(s')$, we have $s'\sigma'$ is ground. Moreover, we have $t' =_{Ax} t$. By assumption (ii), we derive that $s'\sigma = s'(\sigma'[a/v]_{\simeq_{Ax}}) = (s'\sigma')[a/v]_{\simeq_{Ax}} = t'[a/v]_{\simeq_{Ax}}$. Hence, we have $u = t'[a/v]_{\simeq_{Ax}}$ as required.
- Congruence: In this case, there are $c \in \Sigma^n$ and terms $t_1, \ldots, t_n$, $u_1, \ldots, u_n$ such that
$$\begin{aligned}&t = c(t_1, \ldots, t_n),\\ &u = c(u_1, \ldots, u_n),\\ &t_i[a/v]_{\simeq_{Ax}} =_{Ax} u_i \text{ for all } i \in \widetilde{n}.\end{aligned}$$
By induction hypothesis, there is a term t'_i such that $t_i =_{Ax} t'_i$ and $u_i = t'_i[a/v]_{\simeq_{Ax}}$ for all $i \in \widetilde{n}$. We define $t' = c(t'_1, \ldots, t'_n)$ and derive that $t' =_{Ax} t$ and $u = t'[a/v]_{\simeq_{Ax}}$ as required.
- Transitivity: In this case, there is a term w such that $t[a/v]_{\simeq_{Ax}} =_{Ax} w$ and $w =_{Ax} u$. By induction hypothesis, there is a term w' such that $t =_{Ax} w'$ and $w = w'[a/v]_{\simeq_{Ax}}$. Thus, we have $w'[a/v]_{\simeq_{Ax}} =_{Ax} u$. Using the induction hypothesis, we derive that there exists a term t' such that $w' =_{Ax} t'$ and $u = t'[a/v]_{\simeq_{Ax}}$. It follows that $t =_{Ax} t'$ which concludes this case.

□

Lemma 28. *Let u, t, v be ground terms and a be an atom. Suppose that $top(v) \notin ct(Ax)$. Then $u =_{Ax} t$ implies $u[a/v]_{\simeq_{Ax}} =_{Ax} t[a/v]_{\simeq_{Ax}}$.*

Proof. We prove this lemma by induction on the derivation $u =_{Ax} t$ depending on the last rule that has been applied.

- Reflexivity: In this case, we have $u = t$. Thus it is obvious that $u[a/v]_{\simeq_{Ax}} = t[a/v]_{\simeq_{Ax}}$.
- Axiom: In this case, then there are a pair $\{s, s'\} \in Ax$ and a substitution σ such that $u = s\sigma$ and $t = s'\sigma$. Let $\sigma' = \sigma[a/v]_{\simeq_{Ax}}$. Since $top(v) \notin ct(Ax)$, we have

$$\begin{aligned} u[a/v]_{\simeq_{Ax}} &= (s\sigma)[a/v]_{\simeq_{Ax}} = s\sigma', \\ t[a/v]_{\simeq_{Ax}} &= (s'\sigma)[a/v]_{\simeq_{Ax}} = s'\sigma'. \end{aligned}$$

  Using the Axiom rule, we derive that $u[a/v]_{\simeq_{Ax}} =_{Ax} t[a/v]_{\simeq_{Ax}}$ as required.
- Congruence: In this case, we have that $u = g(u_1, \ldots, u_n)$ and $t = g(t_1, \ldots, t_n)$ for some $g \in \Sigma^n$, $n \geq 1$. Moreover, we have $u_i =_{Ax} t_i$ for all $i \in \widetilde{n}$. Since $u =_{Ax} t$, it is clear that $u \in dom([a/v]_{\simeq_{Ax}})$ if and only if $t \in dom([a/v]_{\simeq_{Ax}})$. We consider two cases.
  - If $u \in dom([a/v]_{\simeq_{Ax}})$ then $t \in dom([a/v]_{\simeq_{Ax}})$. Thus, we have $u[a/v]_{\simeq_{Ax}} = a = t[a/v]_{\simeq_{Ax}}$.
  - If $u \notin dom([a/v]_{\simeq_{Ax}})$ then $t \notin dom([a/v]_{\simeq_{Ax}})$. Therefore, we have

$$\begin{aligned} u[a/v]_{\simeq_{Ax}} &= g(u_1[a/v]_{\simeq_{Ax}}, \ldots, u_n[a/v]_{\simeq_{Ax}}), \\ t[a/v]_{\simeq_{Ax}} &= g(t_1[a/v]_{\simeq_{Ax}}, \ldots, t_n[a/v]_{\simeq_{Ax}}). \end{aligned}$$

    Moreover, by induction hypothesis, we know that $u_i[a/v]_{\simeq_{Ax}} =_{Ax} t_i[a/v]_{\simeq_{Ax}}$ for all $i \in \widetilde{n}$. Hence, we obtain $u[a/v]_{\simeq_{Ax}} =_{Ax} t[a/v]_{\simeq_{Ax}}$ as required.
- Transitivity: In this case, there is a term w such that $u =_{Ax} w$ and $w =_{Ax} t$. By induction hypothesis, we have

$$\begin{aligned} u[a/v]_{\simeq_{Ax}} &=_{Ax} w[a/v]_{\simeq_{Ax}}, \\ w[a/v]_{\simeq_{Ax}} &=_{Ax} t[a/v]_{\simeq_{Ax}}. \end{aligned}$$

  It follows that $u[a/v]_{\simeq_{Ax}} =_{Ax} t[a/v]_{\simeq_{Ax}}$ as required.

This completes the proof of the lemma. □

Lemma 29. *Let u, t, v be ground terms and a be an atom such that $a \notin subs(u) \cup subs(t) \cup subs(Ax)$ and $top(v) \notin ct(Ax)$. Then $u[a/v]_{\simeq_{Ax}} =_{Ax} t[a/v]_{\simeq_{Ax}}$ implies $u =_{Ax} t$.*

Proof. We prove this lemma by induction on the derivation $u[a/v]_{\simeq_{Ax}} =_{Ax} t[a/v]_{\simeq_{Ax}}$.

- Reflexivity: In this case, we have $u[a/v]_{\simeq_{Ax}} = t[a/v]_{\simeq_{Ax}}$. By Lemma 25, we have $u =_{Ax} t$ as required.

- Axiom: In this case, there are a pair $\{s, s'\} \in Ax$ and a ground substitution σ such that $dom(\sigma) = vars(s) \cup vars(s')$ and

$$\begin{array}{rcl} u[a/v]_{\simeq_{Ax}} & = & s\sigma, \\ t[a/v]_{\simeq_{Ax}} & = & s'\sigma. \end{array}$$

  Moreover, by Lemma 26, there is a ground substitution σ' such that $dom(\sigma') = vars(s)$ and

$$\begin{array}{rcl} u & =_{Ax} & s\sigma', \\ \sigma'[a/v]_{\simeq_{Ax}} & = & \sigma|_{vars(s)} \,. \end{array}$$

  Since $vars(s) = vars(s')$, we derive that $\sigma|_{vars(s)}= \sigma$. Hence, we have

$$s'\sigma = s'(\sigma'[a/v]_{\simeq_{Ax}}).$$

  By assumption, for all terms $w \in dom([a/v]_{\simeq_{Ax}}) \cap subs(s'\sigma)$, we have that $w \in subs(ran(\sigma))$. Therefore, we must have $s'\sigma = (s'\sigma')[a/v]_{\simeq_{Ax}}$. Hence, we obtain

$$t[a/v]_{\simeq_{Ax}} = (s'\sigma')[a/v]_{\simeq_{Ax}}.$$

  By Lemma 25, we derive that $t =_{Ax} s'\sigma'$. Together with $u =_{Ax} s\sigma'$, we have $u =_{Ax} t$ as required.

- Congruence: In this case, there is $g \in \Sigma^n$ such that

$$\begin{array}{rcl} u[a/v]_{\simeq_{Ax}} & = & g(u_1, \ldots, u_n), \\ t[a/v]_{\simeq_{Ax}} & = & g(t_1, \ldots, t_n). \end{array}$$

  Moreover, we have $u_i =_{Ax} t_i$ for all $i \in \widetilde{n}$. We consider two cases.

  - If $u \in dom([a/v]_{\simeq_{Ax}})$ then by Lemma 24, we have $t \in dom([a/v]_{\simeq_{Ax}})$. Therefore, we have $u =_{Ax} t$ as required.
  - If $u \notin dom([a/v]_{\simeq_{Ax}})$ then by Lemma 24, we also have $t \notin dom([a/v]_{\simeq_{Ax}})$. Thus, there must be terms $u'_1, \ldots, u'_n$ and terms $t'_1, \ldots, t'_n$ such that

$$\begin{array}{lcl} u & = & g(u'_1, \ldots, u'_n), \\ t & = & g(t'_1, \ldots, t'_n), \\ u'_i[a/v]_{\simeq_{Ax}} & = & u_i \text{ for all } i \in \widetilde{n}, \\ t'_i[a/v]_{\simeq_{Ax}} & = & t_i \text{ for all } i \in \widetilde{n}. \end{array}$$

    Hence, we have $u'_i[a/v]_{\simeq_{Ax}} =_{Ax} t'_i[a/v]_{\simeq_{Ax}}$ for all $i \in \widetilde{n}$. By induction hypothesis, we know that $u'_i =_{Ax} t'_i$ for all $i \in \widetilde{n}$. This implies $u =_{Ax} t$ as required.

- Transitivity: In this case, there is a term w such that $u[a/v]_{\simeq_{Ax}} =_{Ax} w$ and $w =_{Ax} t[a/v]_{\simeq_{Ax}}$. We consider two cases.

- If $a \in subs(u[a/v]_{\simeq_{Ax}}) \cup subs(t[a/v]_{\simeq_{Ax}})$ then since $a \notin subs(Ax)$, we derive that $a \in subs(w)$. Hence, there is a term w' such that $w = w'[a/v]_{\simeq_{Ax}}$. By induction hypothesis, we have $u =_{Ax} w'$ and $w' =_{Ax} t$. This implies $u =_{Ax} t$ as required.
- If $a \notin subs(u[a/v]_{\simeq_{Ax}}) \cup subs(t[a/v]_{\simeq_{Ax}})$ then we have $u[a/v]_{\simeq_{Ax}} = u$ and $t[a/v]_{\simeq_{Ax}} = t$. Thus, we obtain $u =_{Ax} t$ as required.

This completes the proof of the lemma. $\square$

Lemma 30. *Let t, u, v be terms and a be an atom such that t, v are ground and $a \notin subs(t) \cup subs(u) \cup subs(Ax)$ and $top(v) \notin ct(Ax)$. Let σ be a ground substitution such that $dom(\sigma) = vars(u)$ and $t[a/v]_{\simeq_{Ax}} =_{Ax} u\sigma$. Then there is a ground substitution σ' such that the following holds.*

(i) $dom(\sigma') = vars(u)$,

(ii) $a \notin subs(ran(\sigma'))$,

(iii) $u\sigma' =_{Ax} t$, *and*

(iv) $\sigma'[a/v]_{\simeq_{Ax}} = \sigma$.

Proof. We prove this lemma by induction on the derivation $t[a/v]_{\simeq_{Ax}} =_{Ax} u\sigma$.

- Reflexivity: We have $t[a/v]_{\simeq_{Ax}} = u\sigma$ and thus the conclusion follows immediately from Lemma 26.
- Axiom: Suppose that there are a pair $\{s, s'\} \in Ax$ and a ground substitution θ such that $dom(\theta) = vars(s) \cup vars(s')$ and

$$\begin{array}{rcl} t[a/v]_{\simeq_{Ax}} & = & s\theta, \\ u\sigma & = & s'\theta. \end{array}$$

  By Lemma 26, there is a ground substitution σ'' such that

$$\begin{array}{l} dom(\sigma'') = vars(s), \\ a \notin subs(ran(\sigma'')), \\ \sigma''[a/v]_{\simeq_{Ax}} = \theta|_{vars(s)}, \text{ and} \\ t =_{Ax} s\sigma''. \end{array}$$

  Since $vars(s) = vars(s')$, we have $\theta\,|_{vars(s)} = \theta$. Hence, we derive that $s'\theta = s'(\sigma''[a/v]_{\simeq_{Ax}})$. This by assumption implies that $s'\theta = (s'\sigma'')[a/v]_{\simeq_{Ax}}$. Since $a \notin subs(ran(\sigma''))$, from assumption (ii), we have $a \notin subs(s'\sigma'')$. Moreover, we have $u\sigma = s'\theta = (s'\sigma'')[a/v]_{\simeq_{Ax}}$. By Lemma 26, there is a ground substitution σ' such that

$$\begin{array}{l} dom(\sigma') = vars(u), \\ a \notin subs(ran(\sigma')), \\ \sigma'[a/v]_{\simeq_{Ax}} = \sigma, \text{ and} \\ s'\sigma'' =_{Ax} u\sigma'. \end{array}$$

  Together with $t =_{Ax} s\sigma''$, we derive that $u\sigma' =_{Ax} t$.

- Congruence: We have $t = c(t_1, \ldots, t_n)$ and $u = c(u_1, \ldots, u_n)$ for $c \in \Sigma^n$. Moreover, we have $t_i[a/v]_{\simeq_{Ax}} =_{Ax} u_i\sigma$ for all $i \in \widetilde{n}$. By induction hypothesis, there are ground substitutions σ_i for all $i \in \widetilde{n}$ such that

$$\begin{array}{l} dom(\sigma_i) = vars(u_i), \\ a \notin subs(ran(\sigma_i)), \\ u_i\sigma_i =_{Ax} t_i, \text{ and} \\ \sigma_i[a/v]_{\simeq_{Ax}} = \sigma. \end{array}$$

  We can pick $\sigma' = \bigcup_{i=1}^{n} \sigma_i$ that satisfies the desired properties.

- Transitivity: In this case, there is a term w such that $t[a/v]_{\simeq_{Ax}} =_{Ax} w$ and $w =_{Ax} u\sigma$. By Lemma 27, there is a ground term t' such that $t =_{Ax} t'$ and $w = t'[a/v]_{\simeq_{Ax}}$. Hence, we have $t'[a/v]_{\simeq_{Ax}} =_{Ax} u\sigma$. By induction hypothesis, there exists a ground substitution σ' such that

$$\begin{array}{l} dom(\sigma') = vars(u), \\ a \notin subs(ran(\sigma')), \\ u\sigma_i =_{Ax} t', \text{ and} \\ \sigma'[a/v]_{\simeq_{Ax}} = \sigma. \end{array}$$

  Since $t' =_{Ax} t$, we derive that σ' satisfies the desired properties.

This completes the proof of the lemma. □

Lemma 31. *Let t, v be ground terms and a be an atom. Suppose that*

(i) t is R,Ax-normal,

(ii) $a \notin subs(Ax) \cup subs(R) \cup subs(t)$, and

(iii) $top(v) \notin ct(Ax)$.

Then $t[a/v]_{\simeq_{Ax}}$ is R,Ax-normal.

Proof. We prove this lemma by induction on t.

- If t is an atom then we have $t[a/v]_{\simeq_{Ax}} = t$ and thus $t[a/v]_{\simeq_{Ax}}$ is R,Ax-normal.
- If $t = g(t_1, \ldots, t_n)$ for some $g \in \Sigma^n$ then we consider two cases.
  - If $t \in dom([a/v]_{\simeq_{Ax}})$ then we have $t[a/v]_{\simeq_{Ax}} = a$ and thus $t[a/v]_{\simeq_{Ax}}$ is R,Ax-normal.
  - If $t \notin dom([a/v]_{\simeq_{Ax}})$ then we have

$$t[a/v]_{\simeq_{Ax}} = g(t_1[a/v]_{\simeq_{Ax}}, \ldots, t_n[a/v]_{\simeq_{Ax}})$$

    Since t is R,Ax-normal, so is t_i for all $i \in \widetilde{n}$. By induction hypothesis, we have $t_i[a/v]_{\simeq_{Ax}}$ is R,Ax-normal for all $i \in \widetilde{n}$. There are two sub-cases.

* If there is no rule $l \to r \in R$ that is applicable to $t[a/v]_{\simeq_{Ax}}$ at the root, then since $t_i[a/v]_{\simeq_{Ax}}$ is R,Ax-normal for all $i \in \widetilde{n}$, we derive that $t[a/v]_{\simeq_{Ax}}$ is R,Ax-normal.
* If there is a rule $l \to r \in R$ that is applicable to $t[a/v]_{\simeq_{Ax}}$ at the root, then there is a ground substitution σ such that $dom(\sigma) = vars(l)$ and $t[a/v]_{\simeq_{Ax}} =_{Ax} l\sigma$. By Lemma 30, there is a ground substitution σ' such that $t =_{Ax} l\sigma'$. This means t is not R,Ax-normal which is a contradiction.

This completes the proof of the lemma. □

Lemma 32. *Let t, v be ground terms such that v is R,Ax-normal and a be an atom. Suppose that*

(i) $a \notin subs(t) \cup subs(Ax) \cup subs(R)$,

(ii) $top(v) \notin ct(Ax)$, *and*

(iii) $top(v) \neq top(l|_p)$ *for all* $l \in lhs(R)$ *and non-variable positiions* $p \in \mathcal{P}os(l) \setminus \{\epsilon\}$.

Then $(t[a/v]_{\simeq_{Ax}})\downarrow_{R,Ax} =_{Ax} t\downarrow_{R,Ax} [a/v]_{\simeq_{Ax}}$.

Proof. We prove this lemma by induction on the length ℓ of the derivation $t \to_{R,Ax} t_1 \to_{R,Ax} \cdots \to_{R,Ax} t_{\ell-1} \to_{R,Ax} t\downarrow_{R,Ax}$.

- If $\ell = 0$ then t is R,Ax-normal. By Lemma 31, $t[a/v]_{\simeq_{Ax}}$ is R,Ax-normal. Therefore, we have $t[a/v]_{\simeq_{Ax}} =_{Ax} (t[a/v]_{\simeq_{Ax}})\downarrow_{R,Ax}$. Moreover, by Lemma 28, we have $t[a/v]_{\simeq_{Ax}} =_{Ax} t\downarrow_{R,Ax} [a/v]_{\simeq_{Ax}}$. Hence $(t[a/v]_{\simeq_{Ax}})\downarrow_{R,Ax} =_{Ax} t\downarrow_{R,Ax} [a/v]_{\simeq_{Ax}}$.

- If $\ell > 0$ then there are a position k, a rule $l \to r \in R$, and a substitution σ such that $t|_k =_{Ax} l\sigma$ and $(t[r\sigma]_k)\downarrow_{R,Ax} =_{Ax} t\downarrow_{R,Ax}$. We will show that

$$t[a/v]_{\simeq_{Ax}} \to (t[r\sigma]_k)[a/v]_{\simeq_{Ax}}.$$

By Lemma 28, we have $t|_k\ [a/v]_{\simeq_{Ax}} =_{Ax} (l\sigma)[a/v]_{\simeq_{Ax}}$. Let $\sigma' = \sigma[a/v]_{\simeq_{Ax}}$ and k' be an arbitrary prefix of k in t. We show that

$$t|_{k'} \notin dom([a/v]_{\simeq_{Ax}}). \tag{B.1}$$

Suppose that it is not the case, then we have $t\,|_{k'} =_{Ax} v$. We also have that $t|_k \in subs(t|_{k'})$ and $t|_k \to_{R,Ax} r\sigma$. Together with the fact that $\to_{R,Ax}$ is coherent, we derive that v is not R,Ax-normal. This however contradicts our assumption. Therefore, we have established (B.1). In particular, we have $l\sigma \notin dom([a/v]_{\simeq_{Ax}})$. Moreover, for all non-variable terms $u \in subs(l) \setminus \{l\}$, we have

$$u\sigma \neq_{Ax} v. \tag{B.2}$$

Let us by contradiction assume that $u\sigma =_{Ax} v$ for some $u \in subs(l) \setminus \{l\}$. Then by point (iii) in Definition 1, we know that $u\sigma =_{Ax} v$ implies $top(u\sigma) = top(v)$. This means $top(u) = top(v)$ which contradicts assumption (iii). Therefore, we have $(l\sigma)[a/v]_{\simeq_{Ax}} = l(\sigma[a/v]_{\simeq_{Ax}}) = l\sigma'$. It follows that $t|_k\ [a/v]_{\simeq_{Ax}} =_{Ax} l\sigma'$. Hence, we obtain $t[t|_k\ [a/v]_{\simeq_{Ax}}]_k =_{Ax} t[l\sigma']_k$. This yields

$$\begin{array}{rcll} t[a/v]_{\simeq_{Ax}} & = & (t[t|_k\ [a/v]_{\simeq_{Ax}}]_k)[a/v]_{\simeq_{Ax}} & \text{by (B.1)} \\ & =_{Ax} & (t[l\sigma']_k)[a/v]_{\simeq_{Ax}} & \text{by Lemma 28.} \end{array}$$

By assumption (i) and (B.2), we know that $subs(l\sigma') \cap dom([a/v]_{\simeq_{Ax}}) = \emptyset$. Hence, we derive that $(t[l\sigma']_k)[a/v]_{\simeq_{Ax}} \rightarrow_{R,Ax} (t[r\sigma']_k)[a/v]_{\simeq_{Ax}}$. Thus, we obtain

$$(t[a/v]_{\simeq_{Ax}})\downarrow_{R,Ax} =_{Ax} ((t[r\sigma']_k)[a/v]_{\simeq_{Ax}})\downarrow_{R,Ax}\ .$$

Moreover, from (B.2) and the fact that $r \in subs(l) \cup \mathcal{C}$, we derive that

$$r\sigma' = r(\sigma[a/v]_{\simeq_{Ax}}) = (r\sigma)[a/v]_{\simeq_{Ax}}.$$

Therefore, we have

$$\begin{array}{rcl} (t[r\sigma']_k)[a/v]_{\simeq_{Ax}} & = & (t[(r\sigma)[a/v]_{\simeq_{Ax}}]_k)[a/v]_{\simeq_{Ax}} \\ & = & (t[r\sigma]_k)[a/v]_{\simeq_{Ax}}. \end{array}$$

Hence, we obtain $t[a/v]_{\simeq_{Ax}} \rightarrow (t[r\sigma]_k)[a/v]_{\simeq_{Ax}}$ and therefore

$$(t[a/v]_{\simeq_{Ax}})\downarrow_{R,Ax} =_{Ax} ((t[r\sigma]_k)[a/v]_{\simeq_{Ax}})\downarrow_{R,Ax} \tag{B.3}$$

Since $(t[r\sigma]_k)\downarrow_{R,Ax} =_{Ax} t\downarrow_{R,Ax}$, by Lemma 28, we have

$$(t[r\sigma]_k)\downarrow_{R,Ax}\ [a/v]_{\simeq_{Ax}} =_{Ax} t\downarrow_{R,Ax}\ [a/v]_{\simeq_{Ax}}. \tag{B.4}$$

By induction hypothesis, we have

$$((t[r\sigma]_k)[a/v]_{\simeq_{Ax}})\downarrow_{R,Ax} =_{Ax} (t[r\sigma]_k)\downarrow_{R,Ax}\ [a/v]_{\simeq_{Ax}}.$$

This by (B.3) and (B.4) yields $(t[a/v]_{\simeq_{Ax}})\downarrow_{R,Ax} =_{Ax} t\downarrow_{R,Ax}\ [a/v]_{\simeq_{Ax}}$ as required.

This completes the proof of the lemma. □

Proposition 18. *Let $\{t, u, a, v\}$ be ground terms such that*

(i) a is an atom and $a \notin subs(t) \cup subs(u) \cup subs(v) \cup subs(Ax) \cup subs(R)$,

(ii) v is R,Ax-normal and $top(v) \notin ct(Ax)$ and v is R,Ax-stable, and

(iii) $top(v) \neq top(l|_p)$ for all $l \in lhs(R)$ and non-variable positions $p \in \mathcal{P}os(l) \setminus \{\epsilon\}$.

Then $t =_E u$ if and only if $t[a/v]_{\simeq_{Ax}} =_E u[a/v]_{\simeq_{Ax}}$.

Proof. We present all the derivation steps as follows.

$$\begin{array}{llll}
 & t & =_E & u \\
\Leftrightarrow & t{\downarrow_{R,Ax}} & =_{Ax} & u{\downarrow_{R,Ax}} \\
\Leftrightarrow & t{\downarrow_{R,Ax}}\,[a/v]_{\simeq_{Ax}} & =_{Ax} & u{\downarrow_{R,Ax}}\,[a/v]_{\simeq_{Ax}} & \text{by Lemmas 28, 29} \\
\Leftrightarrow & t[a/v]_{\simeq_{Ax}}{\downarrow_{R,Ax}} & =_{Ax} & u[a/v]_{\simeq_{Ax}}{\downarrow_{R,Ax}} & \text{by Lemma 32} \\
\Leftrightarrow & t[a/v]_{\simeq_{Ax}} & =_E & u[a/v]_{\simeq_{Ax}}.
\end{array}$$

This completes the proof of the proposition. □

Lemma (Justification of Lemma 2)**.** *Let $T \cup \{t, v\}$ be a set of ground terms such that v is R,Ax-normal and a is an atom. Suppose that*

(i) $a \notin \mathit{subs}(t) \cup \mathit{subs}(v) \cup \mathit{subs}(Ax) \cup \mathit{subs}(R)$,

(ii) $\mathit{top}(v) \notin \mathit{ct}(Ax)$, *and*

(iii) $\mathit{top}(v) \neq \mathit{top}(l|_p)$ *for all* $l \in \mathit{lhs}(R)$ *and non-variable positions* $p \in \mathcal{P}os(l) \setminus \{\epsilon\}$.

Then $T \vdash_E t$ implies $T[a/v]_{\simeq_{Ax}}, a \vdash_E t[a/v]_{\simeq_{Ax}}$.

Proof. We prove this lemma induction on the derivation of $T \vdash_E t$ depending on the last rule that has been applied.

- Ax: We have $t \in T$ and thus $t[a/v]_{\simeq_{Ax}} \in T[a/v]_{\simeq_{Ax}}$. Therefore, we have that $T[a/v]_{\simeq_{Ax}}, a \vdash_E t[a/v]_{\simeq_{Ax}}$.
- Comp: We have $t = g(u_1, \ldots, t_n)$ and $T \vdash_E u_i$ for $i \in \widetilde{n}$. There are two cases.
  - If $t =_{Ax} v$ then we have $t[a/v]_{\simeq_{Ax}} = a$. Hence, we obtain that
  $$T[a/v]_{\simeq_{Ax}}, a \vdash_E t[a/v]_{\simeq_{Ax}}.$$
  - If $t \neq_{Ax} v$ then we have
  $$t[a/v]_{\simeq_{Ax}} = g(u_1[a/v]_{\simeq_{Ax}}, \ldots, u_n[a/v]_{\simeq_{Ax}}).$$
  Moreover, by induction hypothesis, for all $i \in \widetilde{n}$ we have
  $$T[a/v]_{\simeq_{Ax}}, a \vdash_E u_i[a/v]_{\simeq_{Ax}}.$$
  Hence, we obtain $T[a/v]_{\simeq_{Ax}}, a \vdash_E t[a/v]_{\simeq_{Ax}}$ as required.
- Eq: In this case, there is a term t' such that $T \vdash_E t'$ and $t' =_E t$. From assumption (i), we derive that $a \notin \mathit{subs}(t')$. Hence, we can apply the induction hypothesis and obtain $T[a/v]_{\simeq_{Ax}} \vdash_E t'[a/v]_{\simeq_{Ax}}$. By Proposition 18, we have that $t'[a/v]_{\simeq_{Ax}} =_E t[a/v]_{\simeq_{Ax}}$. Thus, we obtain $T[a/v]_{\simeq_{Ax}}, a \vdash_E t[a/v]_{\simeq_{Ax}}$ as desired.

This completes the proof of the lemma. □

Lemma (Justification of Lemma 3)**.** *Let t, v be ground term, and a be an atom. Assume that*

(i) t *is* R,Ax*-normal,*

(ii) v *is composed and not a pair,*

(iii) for all $\tau \in \Pi(E_f^+)$ *and all types* $\tau' \in subs(\tau) \setminus \{\tau\}$*, we have*
  - *(a)* $top(\tau') \neq top(v)$,
  - *(b) if* $\gamma_a \preccurlyeq \tau'$ *then* $\tau' = \mathsf{msg}$.

(iv) for all terms $u \in Rec(F_f, t)$, $u \neq_{Ax} v$,

(v) $a \notin subs(Ax) \cup subs(R)$,

(vi) $top(v) \notin ct(Ax)$.

Then $f(t) = f(t[a/v]_{\simeq_{Ax}})$.

Proof. We show that $f(t) = f(t[a/v]_{\simeq_{Ax}})$ by induction on the size of t.

- If t is an atom then since v is composed, we have $t[a/v]_{\simeq_{Ax}} = t$. Thus we obtain $f(t) = f(t[a/v]_{\simeq_{Ax}})$.
- If $t = c(t_1, \dots, t_n)$ for some $c \in \Sigma^n$ and $n \geq 1$, then let $f(p) = q$ be the pattern in E_f^+ that is chosen for t. Let $p : \tau$. Then we have $\Gamma(t) \preccurlyeq \tau$. We show that $\Gamma(t[a/v]_{\simeq_{Ax}}) \preccurlyeq \tau$.

  Suppose there is a position k such that $t|_k =_{Ax} v$. It is sufficient to show that $\Gamma(t[a]_k) \preccurlyeq \tau$. Let us consider two cases.

  - If $k \in \mathcal{P}os(\tau)$ then we have $\Gamma(t|_k) \preccurlyeq \tau|_k$. Since $t|_k =_{Ax} v$ and v is composed, we derive that $t|_k$ is composed and

    $$top(t|_k) = top(v). \tag{B.5}$$

    Note that $t \in Rec(F_f, t)$. By assumption (iv), we have $t \neq_{Ax} v$. Since $t|_k =_{Ax} v$ and $t \neq_{Ax} v$, we know that k is not the root of t. This means $\tau|_k$ is a strict subterm of τ. Hence, by assumption (iii.a), we derive that $top(\tau|_k) \neq top(v)$. This by (B.5) implies that $top(\tau|_k) \neq top(t|_k)$. Moreover, we know that $t|_k$ is composed and $\Gamma(t|_k) \preccurlyeq \tau|_k$. Therefore, we must have that $\tau|_k = \mathsf{msg}$ and obtain $\Gamma(t[a]_k) \preccurlyeq \tau$ as desired.
  - If $k \notin \mathcal{P}os(\tau)$ then there must be strict prefix k' of k such that $\tau|_{k'} = \mathsf{msg}$. This also yields $\Gamma(t[a]_k) \preccurlyeq \tau$.

  Hence, we have shown that $\Gamma(t[a/v]_{\simeq_{Ax}}) \preccurlyeq \tau$. Similarly, we show that whenever a pattern p' matches $t[a/v]_{\simeq_{Ax}}$, it also matches t. Indeed, suppose that $\Gamma(t[a/v]_{\simeq_{Ax}}) \preccurlyeq \tau$ and there is a position $k \in \mathcal{P}os(t[a/v]_{\simeq_{Ax}})$ such that $t[a/v]_{\simeq_{Ax}}|_k = a$. We consider two cases.

- If $k \in \mathcal{P}os(\tau)$ then we have $\gamma_a \preccurlyeq \tau|_k$. This by assumption (iii.b) implies $\tau|_k = \mathsf{msg}$. Hence, we obtain that $\Gamma(t[v]_k) \preccurlyeq \tau$.
- If $k \notin \mathcal{P}os(\tau)$ then by a similar reasoning as before, we conclude that $\Gamma(t[v]_k) \preccurlyeq \tau$.

Hence, we derive that $\Gamma(t) \preccurlyeq \tau$. Therefore t and $t[a/v]_{\simeq_{Ax}}$ are abstracted under f by the same clause. Let θ and θ' be substitutions such that $t = p\theta$ and $t[a/v]_{\simeq_{Ax}} = p\theta'$. We perform a case distinction on p.

- $p = c(p_1, \ldots, p_n)$ for $c \in \Sigma^n \setminus \{\mathsf{C}_{\mathsf{Ax}} \cup \mathsf{C}_{\mathsf{Key}} \cup \mathsf{C}_{\mathsf{R,Ax}}\}$. In this case, we have

$$f(c(p_1, \ldots, p_n)) = \langle e_1, \ldots, e_d \rangle$$

for some $d > 0$ and for all $i \in \widetilde{d}$, e_i is one of the following forms:

1. $e_i = c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n}))$ such that

$$\begin{array}{l} set(\overline{q_j}) \subseteq split(p_j) \text{ for all } j \in \widetilde{n},\ c \neq \langle \cdot, \cdot \rangle, \text{ and} \\ c \in \mathsf{C}_{\mathsf{R}} \Rightarrow \overline{q_n} = [p_n]. \end{array}$$

2. $e_i = f(q)$ such that $q \in split(p_j)$ for some $j \in \widetilde{n}$.

and it holds that $\forall j \in pp(c).\ split(p_j) \subseteq Q(j)$ where

$$\begin{array}{rl} Q(j) = & \bigcup\{set(\overline{q_j}) \mid c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n})) \in \{e_1, \ldots, e_d\}\} \\ & \bigcup\{q \mid f(q) \in \{e_1, \ldots, e_d\}\}. \end{array}$$

Hence, we have

$$\begin{array}{rcl} f(t) & = & \langle e_1\theta, \ldots, e_d\theta \rangle, \\ f(t[a/v]_{\simeq_{Ax}}) & = & \langle e_1\theta', \ldots, e_d\theta' \rangle. \end{array}$$

To see that $f(t[a/v]_{\simeq_{Ax}}) = f(t)$, it is sufficient to show that $e_i\theta = e_i\theta'$ for all $i \in \widetilde{d}$. Let $i \in \widetilde{d}$. We consider two cases.

  * $e_i = c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n}))$. To show that $e_i\theta = e_i\theta'$, it is sufficient to show that $\widehat{f}(\overline{q_j}\theta) = \widehat{f}(\overline{q_j}\theta')$ for all $j' \in \widetilde{n}$. Let $j \in \widetilde{n}$. Since v is not a pair and the fact that $(p\theta)[a/v]_{\simeq_{Ax}} = p\theta'$, we have $(w\theta)[a/v]_{\simeq_{Ax}} = w\theta'$ for all $w \in set(\overline{q_j})$. Moreover, by assumption (iv), we have $w\theta \neq_{Ax} v$. Note that $w\theta \in subs(t) \setminus \{t\}$. Hence, by induction hypothesis, we have $f(w\theta') = f(w\theta)$. This yields $\widehat{f}(\overline{q_j}\theta) = \widehat{f}(\overline{q_j}\theta')$ as desired.
  * $e_i = f(q)$ with $q \in split(p_j)$ for some $j \in \widetilde{n}$. Since $(p\theta)[a/v]_{\simeq_{Ax}} = p\theta'$, we derive that $q\theta' = (q\theta)[a/v]_{\simeq_{Ax}}$. Moreover, $q \in split(p_j)$ implies that $q\theta \in subs(t) \setminus \{t\}$. By induction hypothesis, we know that $f(q\theta') = f(q\theta)$. Therefore, we obtain $e_i\theta = e_i\theta'$ as required.

- $p = c(p_1, \ldots, p_n)$ for $c \in \Sigma^n \cap (\mathsf{C}_{\mathsf{Ax}} \cup \mathsf{C}_{\mathsf{Key}} \cup \mathsf{C}_{\mathsf{R,Ax}})$. In this case, we have

$$\begin{array}{rcl} f(p) & = & c(f(p_1), \ldots, f(p_n)), \\ f(t) & = & c(f(t_1), \ldots, f(t_n)), \\ f(t[a/v]_{\simeq_{Ax}}) & = & c(f(t_1[a/v]_{\simeq_{Ax}}), \ldots, f(t_n[a/v]_{\simeq_{Ax}})). \end{array}$$

From assumption (iv), we have $t_i \neq_{Ax} v$ for all $i \in \widetilde{n}$. Thus by the induction hypothesis, we have $f(t_i) = f(t_i[a/v]_{\simeq_{Ax}})$. Thus we obtain $f(t) = f(t[a/v]_{\simeq_{Ax}})$ as required.

This completes the proof of the lemma. $\square$

Lemma 33. *Let t be a term and f be composite-preserving and homomorphic for $top(t)$. Suppose that there exists a rewrite rule $l \to r \in R$ and a substitution σ such that*

(i) $f(t) =_{Ax} l\sigma$,

(ii) l is Ax-stable, and

(iii) $l \in lhs(\mathsf{R5})$.

Then t is not R,Ax-normal.

Proof. Suppose that $t = c(t_1, \ldots, t_n)$ for some $c \in \Sigma^n$. Since f is homomorphic for c, we have

$$f(t) = c(f(t_1), \ldots, f(t_n)).$$

By assumption (i), we obtain that $c(f(t_1), \ldots, f(t_n)) =_{Ax} l\sigma$. Moreover, by assumption (ii), we derive that $l = c(u_1, \ldots, u_n)$ for some terms $u_1, \ldots, u_n$ and $f(t_i) =_{Ax} u_i\sigma$ for all $i \in \widetilde{n}$. From assumption (iii), we also know that u_i is either a variable or an atom. Let us define the substitution σ' such that

$$\sigma' = \{t_i/u_i \mid i \in \widetilde{n} \wedge u_i \in \mathcal{V}\}.$$

We show that $t =_{Ax} l\sigma'$. To see this, it is sufficient to show that u_i is an atom implies $u_i =_{Ax} t_i$. Indeed, let $i \in \widetilde{n}$ such that u_i is an atom. Since $f(t_i) =_{Ax} u_i\sigma$, we derive that $f(t_i) =_{Ax} u_i$. Note that $|s| = |s'|$ and s, s' are linear for all $\{s, s'\} \in Ax$. Therefore, we have that $f(t_i)$ is not composed. This together with the assumption that f is composite-preserving implies t_i is not composed. Therefore, we have $f(t_i) = t_i$. It follows that $t_i =_{Ax} u_i$. Hence, we have shown that $t =_{Ax} l\sigma'$. This contradicts our assumption that t is R,Ax-normal and thus completes the proof of the lemma. $\square$

In the following lemma, we define the set of terms $Dec(\sigma)$ for a given substitution σ as follows.

$$Dec(\sigma) = \{t \mid t \in subs(ran(\sigma)) \wedge top(t) \in top(lhs(R)) \wedge f(t) \text{ is a redex}\}.$$

Intuitively, $Dec(\sigma)$ is the set of subterms of terms in $ran(\sigma)$ whose top function symbols are reducible.

Lemma 34. *Let $\phi \in \mathcal{L}_P$, and (tr, th, σ) be a reachable state of P. Suppose that*

(i) (Σ, Ax, R) is top-restricted,

(ii) (F_f, P, ϕ) is compatible with (Σ, Ax, R).

Suppose that $Dec(\sigma) \neq \emptyset$ *and let* $v_0 \in Dec(\sigma)$, a_0 *be a constant that does not occur in* $\mathcal{M}_P$, ϕ, $ran(\sigma)$, R, *and* Ax, *and* $\sigma_0 = \sigma[a_0/v_0]_{\simeq_{Ax}}$. *Then the following holds:*

- $|Dec(\sigma_0)| < |Dec(\sigma)|$,
- (tr, th, σ_0) *is a reachable state in* P, *and*
- *if* $(tr, th, \sigma) \nvDash \phi$ *then* $(tr, th, \sigma_0) \nvDash \phi$.

Proof. First, we show that $top(v_0) \notin ct(Ax)$. Since $f(v_0)$ is a redex, there exists a rewrite rule $l \rightarrow r \in R$ and a substitution θ such that $f(v_0) =_{Ax} l\theta$. By Definition 16(i), we have $top(v_0) = top(f(v_0))$. Since all equations $s \simeq s'$ in Ax satisfy that $top(s) = top(s')$, we derive that $top(f(v_0)) = top(l\theta)$. As l is not a variable, we also have $top(l) = top(l\theta)$. Therefore, we obtain that $top(v_0) = top(l)$. If $l \in lhs(\mathsf{R5})$ and l is Ax-stable then by Lemma 33, we derive that v_0 is not R,Ax-normal which is a contradiction. Otherwise, by Definition 15(ii), we derive that $top(v_0) \notin ct(Ax)$.

Second, we define $\sigma_0 = \sigma[a_0/v_0]_{\simeq_{Ax}}$ and show that σ_0 is well-typed. Note that σ is well-typed by assumption. Let $X \in dom(\sigma_0)$ and suppose that $subs(X\sigma) \cap dom([a_0/v_0]_{\simeq_{Ax}}) \neq \emptyset$. Then $X\sigma$ is composed. Since σ is well-typed, we must have $X : \mathsf{msg}$. Therefore, we have $\Gamma((X\sigma_0)\downarrow_{R,Ax}) \preccurlyeq \Gamma(X)$. Hence σ_0 is well-typed.

Third, we show that σ_0 is R,Ax-normal. Let $X \in dom(\sigma_0)$. Since σ is R,Ax-normal, so is $X\sigma$. By Lemma 31, we have $(X\sigma)[a_0/v_0]_{\simeq_{Ax}}$ is R,Ax-normal. Thus $X\sigma_0$ is R,Ax-normal. Hence σ_0 is R,Ax-normal.

We now show that $|Dec(\sigma_0)| < |Dec(\sigma)|$. For this purpose, it is sufficient to show that for all terms $t \in subs(ran(\sigma))$ such that $f(t)$ is R,Ax-normal, we also have $f(t[a_0/v_0]_{\simeq_{Ax}})$ is R,Ax-normal. Let $t \in subs(ran(\sigma))$ such that $f(t)$ is R,Ax-normal. We claim that for all $u \in Rec(F_f, t)$, it holds that $u \neq_{Ax} v_0$. To see that, let us pick an arbitrary term $u \in Rec(F_f, t)$. Since $f(t)$ is R,Ax-normal, so is $f(u)$. Suppose that $u =_{Ax} v_0$. By Proposition 16, we know that $f(u) =_{Ax} f(v_0)$. Since $f(v_0)$ is not R,Ax-normal, neither is $f(u)$. This, together with $u \in Rec(F_f, t)$, implies that $f(t)$ is not R,Ax-normal which is a contradiction. Therefore, we must have $u \neq_{Ax} v_0$. Hence, we have established that $u \neq_{Ax} v_0$ for all $u \in Rec(F_f, t)$. Note that for all $\tau \in \Pi(E_f^+)$ and all types $\tau' \in subs(\tau) \setminus \{\tau\}$, we derive that $top(\tau') \neq top(v_0)$ from Definition 16(ii.a). Moreover, we also have that $\gamma_{a_0} \preccurlyeq \tau'$ implies $\tau' = \mathsf{msg}$ from assumption (ii.b). Thus by Lemma 3, we have $f(t) = f(t[a_0/v_0]_{\simeq_{Ax}})$. Since $f(t)$ is R,Ax-normal, so is $f(t[a_0/v_0]_{\simeq_{Ax}})$. Hence, we have just proved that

$$|Dec(\sigma_0)| < |Dec(\sigma)|. \tag{B.6}$$

Next, we show that (tr, th, σ_0) is reachable in P. We prove this by induction on the number n of transitions leading to (tr, th, σ).

- Base case ($n = 0$): Since (ϵ, th, σ) is reachable, so is (ϵ, th, σ_0).
- Inductive case ($n = k + 1$): Suppose (tr', th', σ) is reachable in k steps and there is a transition $(tr', th', \sigma) \rightarrow (tr, th, \sigma)$. By induction hypothesis, we have

$$(tr', th', \sigma) \text{ is reachable in } P. \tag{B.7}$$

We consider two non-trivial cases according to the rule r that has been applied in step $k+1$.

- If $r = \mathit{SEND}$ then it is obvious that $(tr', th', \sigma_0) \rightarrow (tr, th, \sigma_0)$. This by (B.7) yields that (tr, th, σ') is reachable in P.
- If $r = \mathit{RECV}$ then there exists $i \in \mathit{TID}$ and $R \in \mathit{dom}(P)$ such that $th'(i) = (R, \mathsf{rcv}(u) \cdot tl)$, $tr = tr' \cdot (i, \mathsf{rcv}(u))$, $th = th'[i \mapsto (R, tl)]$, and

$$\mathit{IK}(tr')\sigma, \mathit{IK}_0 \vdash_E u\sigma.$$

Let $T = \mathit{subs}(\mathit{IK}(tr) \cup \mathit{Sec}_\phi \cup \mathit{EqTerm}_\phi)$. We show the following result.

$$\forall t' \in T \setminus \mathit{vars}(T).\ t'\sigma_0 \neq_{Ax} v_0. \tag{B.8}$$

Suppose that it is not the case, then there is a term $t' \in T$ such that $t'\sigma =_{Ax} v_0$. By Proposition 16, we have $f(t'\sigma_0) =_{Ax} f(v_0)$. By Definition 16(iv), we can apply Theorem 2 and obtain $f(t'\sigma_0) = f(t')f(\sigma_0)$. Note that $f(v_0) = l\theta$. Hence, we obtain that $f(t')f(\sigma_0) =_{Ax} l\theta$. Without loss of generality, we can assume that $\mathit{vars}(l) \cap \mathit{dom}(\sigma_0) = \emptyset$. Thus, we derive that $f(t')\eta =_{Ax} l\eta$, where $\eta = f(\sigma_0) \uplus \theta$. This contradicts assumption (ii). Hence, we have shown (B.8). By (B.8) and Proposition 7, we derive that

$$\mathit{IK}(tr')\sigma_0, \mathit{IK}_0 \vdash_E u\sigma_0.$$

Thus the reachability of (tr, th, σ_0) in P follows immediately.

Then, we show attack preservation for σ_0, i.e., we need to show that

$$\forall \vartheta.\ (tr, th, \sigma, \vartheta) \nvDash \phi \Rightarrow (tr, th, \sigma_0, \vartheta) \nvDash \phi.$$

We prove this by induction on ϕ. It is enough to consider the following cases.

- $\phi \equiv m = m'$ or $\phi \equiv \neg(m = m')$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \vDash m = m' & \\ \Leftrightarrow & m\sigma =_E m'\sigma & \\ \Leftrightarrow & (m\sigma)[a_0/v_0]_{\simeq_{Ax}} =_E (m'\sigma)[a_0/v_0]_{\simeq_{Ax}} & \text{by Proposition 18} \\ \Leftrightarrow & m(\sigma[a_0/v_0]_{\simeq_{Ax}}) =_E m'(\sigma[a_0/v_0]_{\simeq_{Ax}}) & \text{by (B.8)} \\ \Leftrightarrow & m\sigma_0 =_E m\sigma_0 & \\ \Leftrightarrow & (tr, th, \sigma_0, \vartheta) \vDash m = m'. & \end{array}$$

- $\phi \equiv \mathit{honest}(i, R)$ or $\phi \equiv \neg\mathit{honest}(i, R)$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \vDash \mathit{honest}(i, R) & \\ \Leftrightarrow & R^{\vartheta(i)}\sigma \in \mathcal{A}_H & \\ \Leftrightarrow & R^{\vartheta(i)}\sigma' \in \mathcal{A}_H & \text{since } R^{\vartheta(i)}\sigma = R^{\vartheta(i)}\sigma_0 \\ \Leftrightarrow & (tr, th, \sigma_0, \vartheta) \vDash \mathit{honest}(i, R). & \end{array}$$

- $\phi \equiv secret(m)$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \not\models secret(m) & \\ \Leftrightarrow & IK(tr)\sigma, IK_0 \vdash_E m\sigma & \\ \Rightarrow & IK(tr)\sigma_0, IK_0 \vdash_E m\sigma_0 & \text{by (B.8) and Proposition 7} \\ \Leftrightarrow & (tr, th, \sigma_0, \vartheta) \not\models secret(m). & \end{array}$$

The inductive cases are routine. □

Lemma (Justification of Lemma 4)**.** *Let $\phi \in \mathcal{L}_P$. Suppose that*

(i) (Σ, Ax, R) *is top-restricted,*

(ii) (F_f, P, ϕ) *is compatible with* (Σ, Ax, R).

Then for all reachable states (tr, th, σ) of P, there is an R,Ax-normal ground well-typed substitution σ' such that $f(\sigma')$ is R,Ax-normal and the following holds:

- *(tr, th, σ') is a reachable state in P, and*
- *if $(tr, th, \sigma) \not\models \phi$ then $(tr, th, \sigma') \not\models \phi$.*

Proof. By Lemma 34, we know that there is an R,Ax-normal ground well-typed substitution σ_0 such that $|Dec(\sigma)| > |Dec(\sigma_0)|$ and

- (tr, th, σ_0) is a reachable state in P, and
- if $(tr, th, \sigma) \not\models \phi$ then $(tr, th, \sigma_0) \not\models \phi$.

We keep applying Lemma 34 to construct a sequence of R,Ax-normal ground well-typed substitution $\sigma_0, \sigma_1, \ldots, \sigma_n$ for some $n \geq 0$ such that $|Dec(\sigma_n)| = 0$ and

- (tr, th, σ_n) is a reachable state in P, and
- if $(tr, th, \sigma) \not\models \phi$ then $(tr, th, \sigma_n) \not\models \phi$.

Since $|Dec(\sigma_n)| = 0$, we have that $f(\sigma_n)$ is R,Ax-normal. By setting $\sigma' = \sigma_n$, we complete the proof of the lemma. □

C. Proofs for Untyped Abstractions

C.1. Soundness of Atom Removal Abstractions

Given a substitution σ and terms t and u, we define the substitution $\sigma[t/u]$ such that $\sigma[t/u](X) = \sigma(X)[t/u]$ for all $X \in dom(\sigma)$.

Lemma 35. *Let t, u be terms, a be an atom, and σ be a substitution such that $vars(t) \cap dom(\sigma) = \emptyset$. Then $(u\sigma)[t/a] = (u[t/a])(\sigma[t/a])$.*

Proof. We prove this lemma by induction on u.

- If u is an atom then $(u\sigma)[t/a] = u[t/a]$. By assumption, it follows that

  $$(u[t/a])(\sigma[t/a]) = u[t/a].$$

  Thus the lemma holds for this case.

- If u is a variable then by we have $u[t/a] = u$. Therefore, we have $(u\sigma)[t/a] = u(\sigma[t/a]) = (u[t/a])(\sigma[t/a])$.

- If $u = g(u_1, \ldots, u_n)$ for $g \in \Sigma^n$, $n \geq 1$, then we have

$$\begin{array}{rcll} (u\sigma)[t/a] & = & g((u_1\sigma)[t/a], \ldots, (u_n\sigma)[t/a]) & \text{since } a \text{ is an atom} \\ & = & g((u_1[t/a])(\sigma[t/a]), \ldots, (u_n[t/a])(\sigma[t/a])) & \text{by IH} \\ & = & (g(u_1, \ldots, u_n)[t/a])(\sigma[t/a]) & \text{since } a \text{ is an atom} \\ & = & (u[t/a])(\sigma[t/a]) & \end{array}$$

This completes the proof of the lemma. □

To establish soundness for an atom removal abstraction rem_{At} with a set of atoms At, we take the same approach as we did for typed abstractions, i.e., we show that whenever the original protocol P admits an attack (tr, th, σ) on a security property ϕ then there is a substitution θ such that $(rem_{At}(tr), rem_{At}(th), rem_{At}(\theta))$ represents an attack on $rem_{At}(\phi)$ in $rem_{At}(P)$. Analogous to typed abstractions, we can try to prove attack preservation for $\theta = rem_{At}(\sigma)$. Unfortunately, this does not work. In particular, it fails to achieve reachability preservation. We illustrate this issue in the following example.

Example 38. Consider a protocol Q with $dom(Q) = \{A, B\}$. Let na, na' : Nonce and h be a hash function. We assume that the set of rewrite rules consists of the rules for projections and that the set of axioms is empty. We define the initiator role A and the responder role B as follows.

$$\begin{array}{rcl} Q(A) & = & \mathsf{send}(na, na') \\ Q(B) & = & \mathsf{rcv}(X, Y) \end{array}$$

We consider the following scenario. In thread 1, an agent a playing role A generates two nonces na, na' and sends the pair $\langle na, na' \rangle$ to another agent b playing role B. The adversary intercepts this message and sends $\langle \mathsf{h}(na), na' \rangle$ to b. The responder b receives the message sent by the intruder in thread 2. The final state is (tr, th, σ), where

$$\begin{array}{rcl} tr & = & (1, \mathsf{send}(na^{\sharp 1}, na'^{\sharp 1})) \cdot (2, \mathsf{rcv}(X^{\sharp 2}, Y^{\sharp 2})) \\ \sigma & = & \{\mathsf{h}(na^{\sharp 1})/X^{\sharp 2},\ na'^{\sharp 1}/Y^{\sharp 2}\} \end{array}$$

Let us assume that $IK_0 = \mathcal{C} \cup \mathcal{A} \cup \mathcal{F}^\bullet$. This state is reachable because the second premise of the $RECV$ rule is satisfied, i.e., it holds that $\langle na^{\sharp 1}, na'^{\sharp 1} \rangle, IK_0 \vdash_E \langle X^{\sharp 2}, Y^{\sharp 2} \rangle \sigma$. Now, we consider the atom removal rem_{At} with $At = \{na\}$. Applying rem_{At} to tr and σ, we obtain the trace $rem_{At}(tr) = (1, \mathsf{send}(na'^{\sharp 1})) \cdot (2, \mathsf{rcv}(X^{\sharp 2}, Y^{\sharp 2}))$ and the substitution $rem_{At}(\sigma) = \sigma$. However, $(rem_{At}(tr), rem_{At}(th), rem_{At}(\sigma))$ is not reachable in $rem_{At}(Q)$, because $\langle X^{\sharp 2}, Y^{\sharp 2} \rangle rem_{At}(\sigma) = \langle \mathsf{h}(na^{\sharp 1}), na'^{\sharp 1} \rangle$ is not deducible from $na'^{\sharp 1}$ and IK_0. The problem is that we have removed the nonce $na^{\sharp 1}$ which is needed to deduce $\mathsf{h}(na^{\sharp 1})$.

To overcome this issue, we construct a different substitution θ such that $\langle X^{\sharp 2}, Y^{\sharp 2} \rangle \theta$ is derivable from $na'^{\sharp 1}$ and IK_0. It is sufficient to find a different substitution for $X^{\sharp 2}$ and keep the substitution for $Y^{\sharp 2}$ as $Y^{\sharp 2}\sigma = na'^{\sharp 1}$ is clearly deducible from $na'^{\sharp 1}$. Recall that the attacker has a countably infinite choice of nonces $n_i^\bullet \in \mathcal{F}^\bullet$ for each type β_n. We therefore can replace all occurrences of the nonce $na^{\sharp 1}$ in $ran(\sigma)$ by the intruder generated nonce $na_i^\bullet$ and obtain the substitution $\theta = \{\mathsf{h}(na_i^\bullet)/X^{\sharp 2},\ na'^{\sharp 1}/Y^{\sharp 2}\}$. Since $na_i^\bullet \in \mathcal{F}^\bullet \subseteq IK_0$, it is clear that $\langle X^{\sharp 2}, Y^{\sharp 2} \rangle \theta = \langle \mathsf{h}(na_i^\bullet), na'^{\sharp 1} \rangle$ is deducible from $na'^{\sharp 1}$ and IK_0. Hence $(rem_{At}(tr), rem_{At}(th), \theta)$ is reachable in $rem_{At}(Q)$. ♠

In the following, we use the idea sketched above to establish our attack preservation result for atom removal abstractions. First, we introduce an additional definition. Given a set of atoms $At \subseteq \mathcal{A} \cup \mathcal{C} \cup \mathcal{F}$ and a set of terms T, we define $imap(At, T)$ as the set of injective mappings $\eta : tID(At, T) \to \mathbb{N}$ where

$$tID(At, T) = \{i \in TID \mid f^i \in fresh(At^\sharp) \cap split(T)\}.$$

Let $irep(At, T)$ be the set of replacements $\rho : \mathcal{F}^\sharp \rightharpoonup \mathcal{F}^\bullet$ such that for all $f^i \in fresh(At^\sharp) \cap split(T)$, we have that $\rho(f^i) = f^\bullet_{\eta(i)}$ where $\eta \in imap(At, T)$. Intuitively, a replacement $\rho \in irep(At, T)$ replaces each fresh value in $At^\sharp$ that is a field of some tuple in T with an intruder generated fresh value of the same type. We extend ρ homomorphically to all terms as expected. The application of ρ to a substitution σ is defined by applying ρ to the range of σ.

In the following lemma, we show that atom removal abstractions preserve deducibility.

Lemma 36. *Let T be set of terms, t be a term, At a set of atoms such that* *(i)* $At^\sharp$ *is clear in* $T \cup \{t\}$, *and*

(ii) $At^\sharp \cap \mathit{fresh}(IK_0) = \emptyset$.

Let σ *be a substitution such that* $T\sigma, IK_0 \vdash_E t\sigma$. *Let* θ *be a substitution such that* $\theta(X) = \sigma(X)\rho$ *for all* $X \in \mathit{dom}(\sigma)$. *Then, for all* $\rho \in \mathit{irep}(At, T \cup \{t\})$, *we have*

$$\mathit{rem}_{At}(T)\theta, IK_0 \vdash_E \mathit{rem}_{At}(t)\theta.$$

Proof. We start by observing that under assumption (i) we have

$$\mathit{rem}_{At}(u), \mathcal{F}^\bullet \vdash_E u\rho \quad \text{and} \quad u\rho \vdash_E \mathit{rem}_{At}(u). \tag{C.1}$$

for all terms $u \in T \cup \{t\}$ and $\rho \in \mathit{irep}(At, T \cup \{t\})$.

Suppose $T\sigma, IK_0 \vdash_E t\sigma$. We show that $\mathit{rem}_{At}(T)\theta, IK_0 \vdash_E \mathit{rem}_{At}(t)\theta$. Together with $\mathcal{F}^\bullet\theta = \mathcal{F}^\bullet$ and $\mathcal{F}^\bullet \subseteq IK_0$, we can use Lemma 2 to derive the following from (C.1):

$$\mathit{rem}_{At}(T)\theta, IK_0 \vdash_E (T\rho)\theta \quad \text{and} \quad (t\rho)\theta \vdash_E \mathit{rem}_{At}(t)\theta. \tag{C.2}$$

Since $IK_0\rho = IK_0$ by assumption (ii), we can also apply Lemma 2 to the assumption $T\sigma, IK_0 \vdash_E t\sigma$ and deduce $(T\sigma)\rho, IK_0 \vdash_E (t\sigma)\rho$. Next, we use Lemma 35 to deduce $(T\rho)\theta, IK_0 \vdash_E (t\rho)\theta$. Combining this with (C.2) above yields the desired result $\mathit{rem}_{At}(T)\theta, IK_0 \vdash_E \mathit{rem}_{At}(t)\theta$. □

Theorem 9 (Reachability preservation for atom removal abstractions)**.** *Suppose that* $At \subseteq \mathit{atoms}(\mathcal{M}_P)$ *is a set of atoms such that*

(i) At *is clear in* $\mathcal{M}_P$, *and*

(ii) $At^\sharp \cap \mathit{fresh}(IK_0) = \emptyset$, *and*

(iii) $IK_0 \subseteq IK_0'$.

Let $\rho \in \mathit{irep}(At, \mathcal{M}_P^\sharp)$ *and* (tr, th, σ) *be a reachable state of* P. *Let* θ *be a substitution such that* $\theta(X) = \sigma(X)\rho$ *for all* $X \in \mathit{dom}(\sigma)$. *Then we have that* $(\mathit{rem}_{At}(tr), \mathit{rem}_{At}(th), \theta)$ *is a reachable state of* $\mathit{rem}_{At}(P)$.

Proof. It is clear that θ is well-typed. We now prove the first conclusion by induction on the number n of transitions leading to a state (tr, th, σ). For the empty trace ($n = 0$), the theorem holds trivially. For the inductive case ($n = k+1$), assume that (tr', th', σ) is reachable in k steps and there is a transition $(tr', th', \sigma) \to (tr, th, \sigma)$. By induction hypothesis, $(\mathit{rem}_{At}(tr'), \mathit{rem}_{At}(th'), \theta)$ is reachable in $\mathit{rem}_{At}(P)$.

There are two cases according to the rule r that has been applied in the step $k+1$ in P. The case of the *SEND* and *SIGNAL* rules is easy. We consider the case of receive rule. The rule's premises require that there are $i \in \mathit{TID}$, $R \in \mathit{dom}(P)$, and a suffix tl of the role $P(R)^{\sharp i}$ such that

$$th'(i) = (R, \mathsf{rcv}(t) \cdot tl) \quad \text{and} \quad IK(tr')\sigma, IK_0 \vdash t\sigma. \tag{C.3}$$

If $rem_{At}(t) = \mathsf{nil}$ then we have $rem_{At}(tr) = rem_{At}(tr')$ and $rem_{At}(th) = rem_{At}(th')$. Thus it immediately follows from the induction hypothesis that $(rem_{At}(tr), rem_{At}(th), \theta)$ is reachable in $rem_{At}(P)$. Otherwise, the rule's conclusion implies that $tr = tr' \cdot (i, \mathsf{rcv}(t))$ and $th = th'[i \mapsto tl]$. In order to apply the $RECV$ rule in $(rem_{At}(tr'), rem_{At}(th'), \theta)$, we must show the following two premises

$$\begin{array}{l} rem_{At}(th')(i) = (R, \mathsf{rcv}(rem_{At}(t)) \cdot rem_{At}(tl)), \text{ and} \\ IK(rem_{At}(tr')))\theta, IK_0' \vdash_E rem_{At}(t)\theta. \end{array}$$

Clearly, the first premise is satisfied by application of rem_{At} to th'. We now show the second one. Using Lemma 36, we deduce from (C.3) that

$$rem_{At}(IK(tr'))\theta, IK_0 \vdash_E rem_{At}(t)\theta$$

Since $rem_{At}(IK(tr')) = IK(rem_{At}(tr'))$ and $IK_0 \subseteq IK_0'$, we obtain

$$IK(rem_{At}(tr'))\theta, IK_0' \vdash_E rem_{At}(t)\theta.$$

Moreover, the successor state in the conclusion of the $RECV$ rule is

$$(rem_{At}(tr') \cdot (i, \mathsf{rcv}(rem_{At}(t))),\ rem_{At}(th')[i \mapsto (R, rem_{At}(tl))], \sigma),$$

which is identical to $(rem_{At}(tr), rem_{At}(th), \sigma)$. This concludes the proof of the theorem. □

We now show soundness for atom removal abstractions.

Theorem 10 (Soundness theorem for atom removal abstractions). *Let $At \subseteq atoms(\mathcal{M}_P)$ and $\phi \in \mathcal{L}_P$ be a formula such that*

(i) At is clear in $\mathcal{M}_P$,

(ii) $At^\sharp \cap fresh(IK_0) = \emptyset$,

(iii) $IK_0 \subseteq IK_0'$, and

(iv) ϕ is (P, rem_{At})-safe

Let (tr, th, σ) be a reachable state of P and $\rho \in irep(At, \mathcal{M}_P^\sharp)$ be such that

$$ran(\rho) \cap fresh(ran(\sigma|_{vars(tr)})) = \emptyset.$$

Let θ be a substitution such that $dom(\theta) = dom(\sigma)$ and for all $X \in vars(tr)$, we have $\theta(X) = \sigma(X)\rho$. Then the following holds.

1. *$(rem_{At}(tr), rem_{At}(th), \rho(\sigma))$ is reachable in $rem_{At}(P)$, and*
2. *$(tr, th, \theta) \not\models \phi$ implies $(rem_{At}(tr), rem_{At}(th), \rho(\sigma)) \not\models rem_{At}(\phi)$.*

Proof. Let $\theta = \rho(\sigma)$. By Theorem 9, we have that $(rem_{At}(tr), rem_{At}(th), \theta)$ is reachable in $rem_{At}(P)$ (and thus θ is well-typed). Hence, it remains to show that

$$\forall \vartheta.\ (tr, th, \sigma, \vartheta) \nvDash \phi \Rightarrow (rem_{At}(tr), rem_{At}(th), \theta) \nvDash rem_{At}(\phi).$$

We proceed by induction on the structure of ϕ and consider the following non-trivial cases.

- $\phi \equiv m = m'$ or $\phi \equiv \neg(m = m')$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \vDash m = m' & \\ \Leftrightarrow & m\sigma =_E m'\sigma & \\ \Leftrightarrow & (m\sigma)\rho =_E (m'\sigma)\rho & \text{since } ran(\rho) \cap fresh(ran(\sigma)) = \emptyset,\ \rho \text{ injective} \\ \Leftrightarrow & (m\rho)\theta =_E (m'\rho)\theta & \text{by Lemma 35} \\ \Leftrightarrow & rem_{At}(m)\theta =_E rem_{At}(m')\theta & \text{by Definition 21(i)} \\ \Leftrightarrow & (rem_{At}(tr), rem_{At}(th), \theta, \vartheta) \vDash rem_{At}(m) = rem_{At}(m') & \end{array}$$

- $\phi = secret(m)$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \nvDash secret(m) & \\ \Leftrightarrow & IK(tr)\sigma, IK_0 \vdash_E m\sigma & \\ \Rightarrow & rem_{At}(IK(tr))\theta, IK_0 \vdash_E rem_{At}(m)\theta & \text{by Lemma 36, Definition 21(i)} \\ \Rightarrow & IK(rem_{At}(tr))\theta, IK_0 \vdash_E rem_{At}(m)\theta & \\ \Leftrightarrow & (rem_{At}(tr), rem_{At}(th), \theta, \vartheta) \nvDash secret(rem_{At}(m)) & \end{array}$$

This completes the proof of the theorem. □

C.2. Soundness of Variable Removal Abstractions

In the following lemma, we abuse the notation and use $vars(tr)$ to denote the set of variables occurring in tr.

Lemma 37. *Suppose that P is well-formed and (tr, th, σ) is a reachable state of P. Let $V \subseteq \mathcal{V}$ be a set of variables such that V is clear in $\mathcal{M}_P$. Then we have that $rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E (IK(tr) \cup (vars(tr) \cap V^\flat))\sigma$.*

Proof. We proceed by induction on tr. For the base case, $tr = \epsilon$, the lemma holds trivially. For the inductive step, suppose (tr', th', σ) is reachable in P and there is a transition $(tr', th', \sigma) \to (tr, th, \sigma)$ such that $tr = tr' \cdot (i, ev(t))$ for some $i \in TID$ and some term t. By induction hypothesis, we have

$$rem_V(IK(tr'))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E (IK(tr') \cup (vars(tr') \cap V^\flat))\sigma.$$

and we have to show $rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E (IK(tr) \cup (vars(tr) \cap V^\flat))\sigma$. We reason by a case distinction on the rule r that has been applied in the last step. The case that $r = SIGNAL$ is trivial. We therefore consider two following cases:

- If $r = RECV$ then we have that $IK(tr') = IK(tr)$. Thus by induction hypothesis, we have $rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E IK(tr)\sigma$. Therefore, it remains to show that $rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E (vars(tr) \cap V^\flat)\sigma$.

  Note that $tr = tr' \cdot (i, \mathsf{rcv}(t))$. Let $X^i \in vars(t) \cap V^\flat$. We show that

  $$rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E X^i\sigma.$$

  By the premises of the $RECV$ rule, we know that $IK(tr')\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E t\sigma$. Since V is clear in t and $IK(tr) = IK(tr')$, we also have $IK(tr)\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E X^i\sigma$. Since $rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E IK(tr)\sigma$, we obtain $rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E X^i\sigma$.

- If $r = SEND$ then we have $tr = tr' \cdot (i, \mathsf{send}(t))$. Thus, we have that $IK(tr) = IK(tr') \cup \{t\}$. By the well-formedness of P, we have $vars(tr) = vars(tr')$. Hence, it follows from the induction hypothesis that

  $$rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E (vars(tr) \cap V^\flat)\sigma. \tag{C.4}$$

  We are left to show that $rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E IK(tr)\sigma$. Since V is clear in t, we obtain

  $$IK_0, ((vars(t) \cap V^\flat)\sigma \cup \{rem_V(t)\sigma\}) \setminus \{\mathsf{nil}\} \vdash_E t\sigma. \tag{C.5}$$

  Since $vars(t) \subseteq vars(tr)$. By (C.4), we have

  $$rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E (vars(t) \cap V^\flat)\sigma. \tag{C.6}$$

  Together with (C.6) and (C.5), we derive that

  $$IK_0, (rem_V(IK(tr))\sigma \cup \{rem_V(t)\sigma\}) \setminus \{\mathsf{nil}\} \vdash_E t\sigma.$$

  Since $t \in IK(tr)$, we have that $rem_V(t)\sigma \in rem_V(IK(tr))\sigma$. Hence, we obtain that

  $$rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E t\sigma.$$

  By induction hypothesis, we have $rem_V(IK(tr')\sigma) \setminus \{\mathsf{nil}\}, IK_0 \vdash_E IK(tr')\sigma$. Hence, we derive that $rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 \vdash_E IK(tr)\sigma$ as required.

This completes the proof of the lemma. $\square$

Proposition 19. *Suppose that P is well-formed, V is a set of variables V is clear in $\mathcal{M}_P$, and u is a term such that $\mathrm{rem}_V(u) \neq \mathsf{nil}$. Suppose (tr, th, σ) is a reachable state of P. Then $IK(tr)\sigma, IK_0 \vdash_E u\sigma$ implies $IK(rem_V(tr))\sigma, IK_0 \vdash_E rem_V(u)\sigma$.*

Proof. We derive

$$\begin{array}{rll} IK(rem_V(tr))\sigma, IK_0 & \vdash_E \; rem_V(IK(tr))\sigma \setminus \{\mathsf{nil}\}, IK_0 & \\ & \vdash_E \; IK(tr)\sigma, IK_0 & \text{by Lemma 37} \\ & \vdash_E \; u\sigma & \text{by assumption} \\ & \vdash_E \; rem_V(u)\sigma & \text{since } split(rem_V(u)) \subseteq split(u) \end{array}$$

Note that the first derivation follows $IK(rem_V(tr)) = rem_V(IK(tr)) \setminus \{\mathsf{nil}\}$. $\square$

The following theorem states the reachability preservation result for variable removal abstractions.

Theorem 11. *Suppose that P is well-formed, $IK_0 \subseteq IK_0'$, and $V \subseteq \mathcal{V}$ is a set of variables such that V is clear in $\mathcal{M}_P$. Let (tr, th, σ) be reachable in P. Then we have that $(rem_V(tr), rem_V(th), \sigma)$ is reachable in $rem_V(P)$.*

Proof. We prove the reachability of $(rem_V(tr), rem_V(th), \sigma)$ by induction on the number n of transitions leading to a state (tr, th, σ). The theorem holds trivially for the empty trace ($n = 0$). For the inductive case ($n = k + 1$), assume that (tr', th', σ) is reachable in k steps and there is a transition $(tr', th', \sigma) \rightarrow (tr, th, \sigma)$. By induction hypothesis, $(rem_V(tr'), rem_V(th'), \sigma)$ is reachable in $rem_V(P)$.

We distinguish three cases according to the rule r used to justify the step $k+1$ in P. We first treat the case of the receive rule ($r = RECV$). The rule's premises require that there are $i \in TID$, $R \in dom(P)$, and a suffix tl of the role $P(R)^{\sharp i}$ such that

$$th'(i) = (R, \mathsf{rcv}(t) \cdot tl) \quad \text{and} \quad IK(tr')\sigma, IK_0 \vdash t\sigma. \tag{C.7}$$

The rule's conclusion implies that $tr = tr' \cdot (i, \mathsf{rcv}(t))$ and $th = th'[i \mapsto tl]$. We consider two cases.

- If $rem_V(t) = \mathsf{nil}$ then we have that

$$\begin{array}{rcl} rem_V(tr) & = & rem_V(tr'), \\ rem_V(th) & = & rem_V(th'). \end{array}$$

  Hence, we conclude that $(rem_V(tr), rem_V(th), \sigma)$ is reachable in $rem_V(P)$ by the induction hypothesis.

- If $rem_V(t) \neq \mathsf{nil}$ then we show that $rem_V(P)$ has a transition

$$(rem_V(tr'), rem_V(th'), \sigma) \rightarrow (rem_V(tr), rem_V(th), \sigma).$$

  In order to apply the $RECV$ rule in state $(rem_V(tr'), rem_V(th'), \sigma)$ the following two premises must be satisfied:

$$\begin{array}{l} rem_V(th')(i) = (R, \mathsf{rcv}(rem_V(t)) \cdot rem_V(tl)), \text{ and} \\ IK(rem_V(tr')))\sigma, IK_0' \vdash_E rem_V(t)\sigma. \end{array}$$

  The first premise holds by application of rem_V to th'. The second one follows from Proposition 19 and the assumption that $IK_0 \subseteq IK_0'$. The successor state in the conclusion of the $RECV$ rule is

$$(rem_V(tr') \cdot (i, \mathsf{rcv}(rem_V(t))),\ rem_V(th')[i \mapsto (R, rem_V(tl))], \sigma),$$

  which is identical to state $(rem_V(tr), rem_V(th), \sigma)$, whose reachability in $rem_V(P)$ we have hereby established.

The case of the send rule ($r = \mathit{SEND}$) is similar but simpler, since the deducibility condition falls away. The case that $r = \mathit{SIGNAL}$ is trivial. This completes the proof of the theorem. $\square$

We now show soundness for variable removal abstractions.

Theorem 12 (Soundness for variable removal abstractions)**.** *Suppose that P is well-formed. Let $V \subseteq \mathcal{V}$ be a set of variables that is clear in $\mathcal{M}_P$. Let $\phi \in \mathcal{L}_P$ be a safe formula for (P, rem_V) and (tr, th, σ) be a reachable state in P. Suppose that $IK_0 \subseteq IK'_0$. Then we have that*

1. *$(\mathit{rem}_V(tr), \mathit{rem}_V(th), \sigma)$ is a reachable state in $\mathit{rem}_V(P)$,*
2. *$(tr, th, \sigma) \not\models \phi$ implies $(\mathit{rem}_V(tr), \mathit{rem}_V(th), \sigma) \not\models \mathit{rem}_V(\phi)$.*

Proof. By Theorem 11, we have that $(\mathit{rem}_V(tr), \mathit{rem}_V(th), \sigma)$ is a reachable state in $\mathit{rem}_V(P)$. It remains to show that

$$\forall \vartheta.\ (tr, th, \sigma, \vartheta) \not\models \phi \Rightarrow (\mathit{rem}_V(tr), \mathit{rem}_V(th), \sigma) \not\models \mathit{rem}_V(\phi).$$

We proceed by induction on the structure of ϕ and consider the following non-trivial cases.

- $\phi \equiv m = m'$ or $\phi \equiv \neg(m = m')$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \models m = m' & \\ \Leftrightarrow & m\sigma =_E m'\sigma & \\ \Leftrightarrow & \mathit{rem}_V(m)\sigma =_E \mathit{rem}_V(m')\sigma & \text{since } \phi \text{ is } (P, \mathit{rem}_V)\text{-safe} \\ \Leftrightarrow & (\mathit{rem}_V(tr), \mathit{rem}_V(th), \sigma, \vartheta) \models \mathit{rem}_V(m) = \mathit{rem}_V(m') & \end{array}$$

- $\phi = \mathit{secret}(m)$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \not\models \mathit{secret}(m) & \\ \Leftrightarrow & IK(tr)\sigma, IK_0 \vdash_E m\sigma & \\ \Rightarrow & \mathit{rem}_V(IK(tr))\sigma, IK_0 \vdash_E m\sigma & \text{by Proposition 19} \\ \Leftrightarrow & \mathit{rem}_V(IK(tr))\sigma, IK_0 \vdash_E \mathit{rem}_V(m)\sigma & \text{since } \mathit{rem}_V(m) = m \\ \Rightarrow & IK(\mathit{rem}_V(tr))\sigma, IK'_0 \vdash_E \mathit{rem}_V(m)\sigma & \\ \Leftrightarrow & (\mathit{rem}_V(tr), \mathit{rem}_V(th), \sigma, \vartheta) \not\models \mathit{secret}(\mathit{rem}_V(m)) & \end{array}$$

This completes the proof of the theorem. $\square$

C.3. Soundness of Redundancy Removal Abstractions

We overload the notation and use $\mathit{term}(tr)$ to denote the set of terms occurring in trace tr. In the following theorem, we show reachability preservation for redundancy removal abstractions.

Lemma 38. *Let rd be a redundancy removal abstraction for P and $IK_0 \subseteq IK_0'$. Then, for all states (tr, th, σ) reachable in P, we have $IK(rd(tr))\sigma, IK_0' \vdash_E term(tr)\sigma$.*

Proof. We proceed by induction on the number n of transitions leading to a state (tr, th, σ). The theorem trivially holds for base case $(n = 0)$ where $tr = \epsilon$.

For the inductive case $(n = k + 1)$, we assume that (tr', th', σ) is reachable in k steps and there is a transition $(tr', th', \sigma) \to (tr, th, \sigma)$. If the last rule r that has been applied is *SIGNAL* then the conclusion immediately follows from the induction hypothesis. We therefore consider the case that $r = SEND$ or $r = RECV$. Suppose that this transition is performed by thread i. Then we know that $tr = tr' \cdot (i, ev(t))$ for some $ev \in \{\mathsf{send}, \mathsf{rcv}\}$. By the induction hypothesis, we have

$$IK(rd(tr'))\sigma, IK_0' \vdash_E term(tr')\sigma. \tag{C.8}$$

Moreover, we have $term(tr) = term(tr') \cup \{t\}$. Thus it is sufficient to show

$$IK(rd(tr))\sigma, IK_0' \vdash_E t\sigma. \tag{C.9}$$

We do this by case analysis on the rule that justifies transition $k + 1$.

- Rule *SEND*. We have $rd(tr) = rd(tr') \cdot (i, \mathsf{send}(rd(t)))$ and thus $IK(rd(tr)) = IK(rd(tr')) \cup \{rd(t)\}$ if $rd(t) \neq \mathsf{nil}$ and $rd(tr) = rd(tr')$ otherwise. Hence, we can derive

  $$\begin{array}{rlll} IK(rd(tr))\sigma, IK_0' & \vdash_E & IK(rd(tr'))\sigma \cup \{rd(t)\sigma\} \setminus \{\mathsf{nil}\}, IK_0' & \text{by above} \\ & \vdash_E & term(tr')\sigma \cup \{rd(t)\sigma\} \setminus \{\mathsf{nil}\}, IK_0' & \text{by (C.8)} \end{array}$$

  Next, since the terms of all events preceding $\mathsf{send}(t)$ on $P(R)$ are contained in $term(tr')$ and $rd \in RD_P$, we derive $IK_0, term(tr'), \mathcal{V}_\alpha \cup \{rd(t)\} \setminus \{\mathsf{nil}\} \vdash_E t$. Instantiating this with σ and observing that $\mathcal{V}_\alpha\sigma \subseteq \mathcal{A} \subseteq IK_0 \subseteq IK_0'$ yields

  $$term(tr')\sigma \cup \{rd(t)\sigma\} \setminus \{\mathsf{nil}\}, IK_0' \vdash_E t\sigma.$$

  Combining this with the derivation above yields the desired conclusion (C.9).

- Rule *RECV*. In this case we can reason as follows.

  $$\begin{array}{rlll} IK(rd(tr))\sigma, IK_0' & \vdash_E & term(tr')\sigma, IK_0 & \text{by (C.8), } IK(rd(tr')) \subseteq IK(rd(tr)) \\ & \vdash_E & IK(tr')\sigma, IK_0 & \text{since } IK(tr') \subseteq term(tr') \\ & \vdash_E & t\sigma & \text{by second premise of rule } RECV \end{array}$$

  This establishes (C.9) as required.

This concludes the proof of the lemma. □

Proposition 20. *Let rd be a redundancy removal abstraction for P. Suppose that $IK_0 \subseteq IK_0'$. Then, for all reachable states (tr, th, σ) in P, $(rd(tr), rd(th), \sigma)$ is a reachable state of $rd(P)$.*

Proof. We proceed by induction on the number n of transitions leading to a state (tr, th, σ). The theorem trivially holds for base case ($n = 0$) where tr is the empty trace. For the inductive case ($n = k+1$), we assume that (tr', th', σ) is reachable in k steps and there is a transition $(tr', th', \sigma) \to (tr, th, \sigma)$. Suppose that this transition is performed by thread i. From the transition rules, we know that we have $th'(i) = (R, ev(t) \cdot tl)$ where $R \in dom(P)$, $ev \in \{\mathsf{send}, \mathsf{rcv}\}$, and tl is a suffix of the role $P(R)^{\sharp i}$. We also have $tr = tr' \cdot (i, ev(t))$ and $th = th'[i \mapsto tl]$.

By the induction hypothesis, we have $(rd(tr'), rd(th'), \sigma)$ is a reachable state of $rd(P)$. If $rd(t) = \mathsf{nil}$ then we are done, since $rd(tr') = rd(tr)$ and $rd(th') = rd(th)$. Otherwise, we have $rd(t) \neq \mathsf{nil}$. In this case, it is sufficient to show that $rd(P)$ has a transition

$$(rd(tr'), rd(th'), \sigma) \to (rd(tr), rd(th), \sigma).$$

We proceed by case distinction on the rule applied to justify step $k+1$ of P and consider two non-trivial cases:

- *SEND* rule. The rule's premise requires $th'(i) = (R, \mathsf{send}(t) \cdot tl)$. Hence, by the definition of $rd(th')$, we have $rd(th')(i) = (R, \mathsf{send}(rd(t)) \cdot rd(tl))$. Moreover, we have $rd(tr) = rd(tr') \cdot (i, \mathsf{send}(rd(t)))$ and $rd(th) = rd(th')[i \mapsto rd(tl)]$, which by the *SEND* rule justifies the transition above.

- *RECV* rule. This rule's premises require that $th'(i) = (R, \mathsf{rcv}(t) \cdot tl)$ and

  $$IK(tr')\sigma, IK_0 \vdash_E t\sigma.$$

  The rule's conclusion implies that $tr = tr' \cdot (i, \mathsf{rcv}(t))$ and $th = th'[i \mapsto tl]$. In order to apply the *RECV* rule in state $(rd(tr'), rd(th'), \sigma)$ two premises must be satisfied: first, $rd(th')(i) = (R, \mathsf{rcv}(rd(e)) \cdot rd(tl))$, which holds by the definition of $rd(th')$, and, second,

  $$IK(rd(tr'))\sigma, IK_0' \vdash_E rd(t)\sigma, \tag{C.10}$$

  which we show now. Since $rd \in RD_P$ and $term(tr')$ contains the terms of all events preceding $\mathsf{rcv}(t)$ on $P(R)$, we have $IK_0, term(tr'), \mathcal{V}_\alpha, t \vdash_E rd(t)$. Noting that $term(tr) = term(tr') \cup \{t\}$ and $(\mathcal{V}_\alpha)\sigma \subseteq \mathcal{A} \subseteq IK_0$ we derive

  $$term(tr)\sigma, IK_0 \vdash_E rd(t)\sigma.$$

  Moreover, from Lemma 38, we have $IK(rd(tr))\sigma, IK_0 \vdash_E term(tr)\sigma$. Combining these facts with the observation that $IK(rd(tr)) = IK(rd(tr'))$ and the assumption that $IK_0 \subseteq IK_0'$, we obtain (C.10) as required.

This completes the proof of the proposition. □

We show the soundness result for redundancy removal abstractions in the following theorem.

Theorem (Soundness; Justification of Theorem 6)**.** *Let $\phi \in \mathcal{L}_P$ be a (P, rd)-safe formula and rd be a redundancy removal abstraction for P. Suppose that $IK_0 \subseteq IK_0'$. Then, for all reachable states (tr, th, σ) in P, we have*

1. $(rd(tr), rd(th), \sigma)$ *is reachable in* $rd(P)$ *and*

2. $(tr, th, \sigma) \nvDash \phi$ *implies* $(rd(tr), rd(th), \sigma) \nvDash rd(\phi)$.

Proof. By Proposition 20, we have that $(rd(tr), rd(th), \sigma)$ is reachable in $rd(P)$. It remains to show that

$$\forall\vartheta.\ (tr, th, \sigma, \vartheta) \nvDash \phi \Rightarrow (rd(tr), rd(th), \sigma) \nvDash rd(\phi).$$

We proceed by induction on the structure of ϕ and consider the following cases.

- $\phi \equiv m = m'$ or $\phi \equiv \neg(m = m')$.

$$\begin{array}{ll} & (tr, th, \sigma, \vartheta) \vDash m =_E m' \\ \Leftrightarrow & m\sigma =_E m'\sigma \\ \Leftrightarrow & (rd(tr), rd(th), \sigma, \vartheta) \vDash rd((m = m')). \end{array}$$

- $\phi = secret(m)$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \nvDash secret(m) & \\ \Leftrightarrow & IK(tr)\sigma, IK_0 \vdash_E m\sigma & \\ \Rightarrow & IK(rd(tr))\sigma, IK_0 \vdash_E m\sigma & \text{by Lemma 38} \\ \Leftrightarrow & (rd(tr), rd(th), \sigma, \vartheta) \nvDash rd(secret(m)). & \end{array}$$

The remaining cases are routine. This completes the proof of the theorem. $\square$

D. Proofs for Splitting Abstractions

D.1. Proof of Reachability Preservation

In this subsection, we show that splitting abstractions preserve reachability. We need the following auxiliary lemma.

Lemma 39. *For all terms $t \in \mathcal{T}$ and all substitutions θ, it holds that*

$$split(t\theta) = split(split(t)\theta).$$

Proof. We prove the lemma by induction on t.

- If t is not a pair, then $split(t) = \{t\}$. Hence $split(t\theta) = split(split(t)\theta)$.
- If $t = \langle t_1, t_2 \rangle$ then we have

$$\begin{aligned} split(t\theta) &= split(\langle t_1\theta, t_2\theta \rangle) \\ &= split(t_1\theta) \cup split(t_2\theta). \end{aligned} \tag{D.1}$$

  By induction hypothesis, we have

$$\begin{aligned} split(t_1\theta) = split(split(t_1)\theta), \\ split(t_2\theta) = split(split(t_2)\theta). \end{aligned} \tag{D.2}$$

  Moreover, we have

$$split(t) = split(t_1) \cup split(t_2). \tag{D.3}$$

  By (D.1), (D.2) and (D.3) we have

$$\begin{aligned} split(t\theta) &= split(split(t_1)\theta) \cup split(split(t_2)\theta) \\ &= split(split(t_1)\theta \cup split(t_2)\theta) \\ &= split((split(t_1) \cup split(t_2))\theta) \\ &= split(split(t)\theta). \end{aligned}$$

This completes the proof of the lemma. □

We now prove our reachability preservation result.

Proposition (Reachability preservation; Justification of Proposition 10)**.** *Let P be a protocol and f_{sp}: $Evt(\mathcal{M}) \to Evt(\mathcal{M})^+$ be a splitting abstraction. Assume that $IK_0 \subseteq IK_0'$. Then for all states (tr, th, σ) reachable in P, we have $(f_{sp}(tr), f_{sp}(th), \sigma)$ is reachable in $f_{sp}(P)$.*

Proof. We prove the proposition by induction on the number n of transitions leading to a state (tr, th, σ).

- Base case ($n = 0$): We have $tr = \epsilon$. Moreover, for all $i \in TID$, there exists $R \in dom(()P)$ such that

$$\begin{array}{rcl} th(i) & = & (R, P(R)^{\sharp i}), \\ f_{sp}(th)(i) & = & (R, f_{sp}(P(R))^{\sharp i}). \end{array} \quad \text{(D.4)}$$

  Since $f_{sp}(\epsilon) = \epsilon$, (D.4) implies that $(f_{sp}(tr), f_{sp}(th), \sigma) \in Init(P)$ and hence we have that $(f_{sp}(tr), f_{sp}(th), \sigma)$ is reachable in $f_{sp}(P)$.

- Inductive case ($n = k + 1$): Suppose (tr', th', σ) is reachable in k steps and there is a transition $(tr', th', \sigma) \to (tr, th, \sigma)$. By induction hypothesis we have that

$$(f_{sp}(tr'), f_{sp}(th'), \sigma) \text{ is reachable in } f_{sp}(P). \quad \text{(D.5)}$$

  We consider two cases according to the rule r that has been applied in step $k + 1$.

  - If $r = SEND$ then there exists $i \in TID$ and $R \in dom(P)$ such that

$$\begin{array}{rcl} th'(i) & = & (R, \mathsf{send}(pt) \cdot tl), \\ tr & = & tr' \cdot (i, \mathsf{send}(pt)), \\ th & = & th'[i \mapsto (R, tl)]. \end{array} \quad \text{(D.6)}$$

    Moreover, by (D.6) we have

$$\begin{array}{rcl} f_{sp}(th) & = & f_{sp}(th'[i \mapsto (R, tl)]) \\ & = & f_{sp}(th')[i \mapsto (R, f_{sp}(tl))], \\ f_{sp}(tr) & = & f_{sp}(tr' \cdot (i, \mathsf{send}(pt))) \\ & = & f_{sp}(tr') \cdot (i, \mathsf{send}(t_1)) \cdots (i, \mathsf{send}(t_n)), \end{array} \quad \text{(D.7)}$$

    where $n \geq 1$.

    For all $j \in \{1, \ldots, n\}$, we use the following abbreviations:

$$\begin{array}{rcl} tr_j & = & f_{sp}(tr') \cdot (i, \mathsf{send}(t_1)) \cdots (i, \mathsf{send}(t_j)), \\ th_j & = & f_{sp}(th')[i \mapsto (R, \mathsf{send}(t_j) \cdots \mathsf{send}(t_n) \cdot f_{sp}(tl))]. \end{array} \quad \text{(D.8)}$$

    By (D.7), for all $j \in \{1, \ldots, n\}$ we have

$$th_j(i) = (R, \mathsf{send}(t_j) \cdots \mathsf{send}(t_n) \cdot f_{sp}(tl)). \quad \text{(D.9)}$$

    By (D.7), (D.8), (D.9) and rule *SEND*, there is a sequence of transitions

    $(f_{sp}(tr'), f_{sp}(th'), \sigma) \to (tr_1, th_1, \sigma) \to (tr_2, th_2, \sigma) \to \cdots \to (tr_n, th_n, \sigma)$
    where $(tr_n, th_n, \sigma) = (f_{sp}(tr), f_{sp}(th), \sigma)$.

    This together with (D.5) implies that $(f_{sp}(tr), f_{sp}(th), \sigma)$ is reachable in P.

– If $r = RECV$ then there exists $i \in TID$ and $R \in dom(()P)$ such that

$$\begin{array}{lcl} th'(i) & = & (R, \mathsf{rcv}(pt) \cdot tl), \\ tr & = & tr' \cdot (i, \mathsf{rcv}(pt)), \\ th & = & th'[i \mapsto (R, tl)], \\ IK(tr')\sigma \cup IK_0 & \vdash_E & pt\sigma. \end{array} \tag{D.10}$$

By (D.10) we have

$$f_{sp}(th')(i) = f_{sp}(th'(i)) = (R, \mathsf{rcv}(t_1) \cdots \mathsf{rcv}(t_n) \cdot f_{sp}(tl)), \text{ where } n \geq 1. \tag{D.11}$$

Moreover, it follows from Definition 26(ii) that

$$split(IK(tr')) = split(IK(f_{sp}(tr'))).$$

This by Lemma 39 yields

$$split(IK(tr')\sigma) = split(IK(f_{sp}(tr'))\sigma).$$

Hence since $IK_0 \subseteq IK_0'$, by (D.10) we have $IK(f_{sp}(tr'))\sigma \cup IK_0' \vdash_E pt\sigma$. By Definition 26 we have

$$split(\langle t_1, \ldots, t_n \rangle) = split(pt).$$

Therefore, for all $i \in \{1, \ldots, n\}$ we have

$$IK(f_{sp}(tr'))\sigma \cup IK_0' \vdash_E t_i\sigma. \tag{D.12}$$

For all $j \in \{1, \ldots, n\}$, let

$$\begin{array}{ll} tr_j & = f_{sp}(tr') \cdot (i, \mathsf{rcv}(t_1)) \cdots (i, \mathsf{rcv}(t_j)), \\ th_j & = f_{sp}(th')[i \mapsto (R, \mathsf{rcv}(t_j) \cdots \mathsf{rcv}(t_n) \cdot f_{sp}(tl))]. \end{array} \tag{D.13}$$

Then by (D.13), for all $j \in \{1, \ldots, n\}$ we have

$$th_j(i) = \mathsf{rcv}(t_j) \cdots \mathsf{rcv}(t_n) \cdot f_{sp}(tl) \tag{D.14}$$

By (D.12), (D.13), (D.14), (D.11) and rule $RECV$, there is a sequence of transitions

$$(f_{sp}(tr'), f_{sp}(th'), \sigma) \to (tr_1, th_1, \sigma) \to (tr_2, th_2, \sigma) \to \cdots \to (tr_n, th_n, \sigma)$$
where $(tr_n, th_n, \sigma) = (f_{sp}(tr), f_{sp}(th), \sigma)$.

Hence by (D.5) we have $(f_{sp}(tr), f_{sp}(th), \sigma)$ is reachable in P.

This completes the proof of the proposition. □

Next, we show that splitting abstractions preserve attacks.

D.2. Proof of Attack Preservation

Theorem (Attack preservation;Justification of Theorem 7)**.** *Let P be a non-overlapping protocol and $f\colon \mathit{Event}_P \to \mathit{Evt}^+$ be a splitting abstraction, and $\phi \in \mathcal{L}_P$ be closed. Then we have, for all states (tr, th, σ) reachable in P,*

- *$(f_{sp}(tr), f_{sp}(th), \sigma)$ is reachable in $f_{sp}(P)$ and*
- *$(tr, th, \sigma) \nvDash_E \phi$ implies $(f_{sp}(tr), f_{sp}(th), \sigma) \nvDash_E f_{sp}(\phi)$.*

Proof. Let (tr, th, σ) be a reachable state of P. The first conjunct, namely that the state $(f_{sp}(tr), f_{sp}(th), \sigma)$ is reachable in $f_{sp}(P)$ follows from Proposition 10. To show the second conjunct, we prove the following statement by induction on the structure of ϕ:

$$\forall \vartheta.\ (tr, th, \sigma, \vartheta) \nvDash_E \phi \Rightarrow (f_{sp}(tr), f_{sp}(th), \sigma, \vartheta) \nvDash_E f_{sp}(\phi).$$

We consider the following cases:

- $\phi \equiv i = j$ or $\phi \equiv \neg(i = j)$.

$$\begin{array}{ll} & (tr, th, \sigma, \vartheta) \vDash_E i = j \\ \Leftrightarrow & \vartheta(i) = \vartheta(j) \\ \Leftrightarrow & (f_{sp}(tr), f_{sp}(th), \sigma, \vartheta) \vDash_E f_{sp}(i = j) \end{array}$$

- $\phi \equiv m = m'$ or $\phi \equiv \neg(m = m')$.

$$\begin{array}{ll} & (tr, th, \sigma, \vartheta) \vDash_E m =_E m' \\ \Leftrightarrow & m\sigma =_E m'\sigma \\ \Leftrightarrow & (f_{sp}(tr), f_{sp}(th), \sigma, \vartheta) \vDash_E m =_E m' \end{array}$$

- $\phi \equiv \mathit{role}(i, R)$ or $\phi \equiv \neg\mathit{role}(i, R)$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \vDash_E \mathit{role}(i, R) & \\ \Leftrightarrow & \exists \mathit{seq} \in \mathit{Evt}^*.\ th(\vartheta(i)) = (R, \mathit{seq}) & \\ \Leftrightarrow & \exists \mathit{seq} \in \mathit{Evt}^*.\ f_{sp}(th)(\vartheta(i)) = (R, f_{sp}(\mathit{seq})) & \text{by Definition 27} \\ \Leftrightarrow & (f_{sp}(tr), f_{sp}(th), \sigma, \vartheta) \vDash_E \mathit{role}(i, R) & \end{array}$$

- $\phi \equiv \mathit{honest}(i, R)$ or $\phi \equiv \neg\mathit{honest}(i, R)$.

$$\begin{array}{ll} & (tr, th, \sigma, \vartheta) \vDash_E \mathit{honest}(i, R) \\ \Leftrightarrow & R_{\vartheta(i)}\sigma \in \mathcal{A}_H \\ \Leftrightarrow & (f_{sp}(tr), f_{sp}(th), \sigma, \vartheta) \vDash_E \mathit{honest}(i, R) \end{array}$$

- $\phi \equiv \mathit{steps}(i, s(m))$ or $\phi \equiv \neg\mathit{steps}(i, s(m))$, where $s \in \{\mathsf{send}, \mathsf{rcv}\}$. We have

$$(tr, th, \sigma, \vartheta) \vDash_E \mathit{steps}(i, s(m)) \Leftrightarrow \quad (\vartheta(i), s(m)) \in tr.$$

We now show that $(\vartheta(i), s(m)) \in tr \Leftrightarrow (\vartheta(i), f_{sp}(s(m))) \in f_{sp}(tr)$. The direction $(\vartheta(i), s(m)) \in tr \Rightarrow (\vartheta(i), f_{sp}(s(m))) \in f_{sp}(tr)$ is obvious by Definition 27. It remains to show that $(\vartheta(i), f_{sp}(s(m))) \in f_{sp}(tr) \Rightarrow (\vartheta(i), s(m)) \in tr$. By Definition 26, we derive that

$$\begin{array}{l} f_{sp}(s(m)) = s(m_1) \cdots s(m_n), \text{ where } n \geq 1, \\ split(m) = split(\langle m_1, \ldots, m_n \rangle). \end{array} \tag{D.15}$$

Then by Definition 27, $(\vartheta(i), f_{sp}(s(m))) \in f_{sp}(tr)$ implies that there exists $s(m') \in Evt(\mathcal{M}_P^\sharp)$ such that the following holds:

$$\begin{array}{l} (\vartheta(i), s(m')) \in tr \\ f_{sp}(s(m')) = s(m'_1) \cdots s(m'_k), \text{ where } k \geq 1, \\ split(m') = split(\langle m'_1, \ldots, m'_k \rangle), \\ m_1 \in \{m'_1, \ldots, m'_k\}. \end{array} \tag{D.16}$$

By (D.15) and (D.16) we have

$$split(m_1) \subseteq split(m) \cap split(m'). \tag{D.17}$$

If $m \neq m'$ then by (D.17) and Definition 28, we conclude that P is overlapping which is a contradiction. Hence $m' = m$ which by (D.16) implies $(\vartheta(i), s(m)) \in tr$. Therefore, we have

$$(\vartheta(i), s(m)) \Leftrightarrow (\vartheta(i), f_{sp}(s(m))) \in f_{sp}(tr).$$

Moreover, we also have

$$(\vartheta(i), f_{sp}(s((m)))) \in f_{sp}(tr) \Leftrightarrow (f_{sp}(tr), f_{sp}(th), \sigma, \vartheta) \vDash_E f_{sp}(steps(i, s(m))).$$

Hence it holds that

$$(tr, th, \sigma, \vartheta) \vDash_E steps(i, s(m)) \Leftrightarrow (f_{sp}(tr), f_{sp}(th), \sigma, \vartheta) \vDash_E f_{sp}(steps(i, s(m)))$$

- $\phi \equiv (i, s(m)) \prec (j, s'(m'))$ or $\phi \equiv \neg((i, s(m)) \prec (j, s'(m')))$, where $s, s' \in \{\mathsf{send}, \mathsf{rcv}\}$.

$$(tr, th, \sigma, \vartheta) \vDash_E (i, s(m)) \prec (j, s'(m') \Leftrightarrow \quad (\vartheta(i), s(m)) \prec_{tr} (\vartheta(j), s'(m')).$$

We now show that

$$(\vartheta(i), s(m)) \prec_{tr} (\vartheta(j), s'(m')) \Leftrightarrow (\vartheta(i), f_{sp}(s(m))) \prec_{f_{sp}(tr)} (\vartheta(j), f_{sp}(s'(m'))).$$

By Definition 26 we have

$$\begin{array}{rcl} f_{sp}(s(m)) & = & s(m_1) \cdots s(m_n), \\ f_{sp}(s'(m')) & = & s'(m'_1) \cdots s'(m'_k), \text{ where } n, m \geq 1. \end{array}$$

By Definition 27, it is obvious that

$$(\vartheta(i), s(m)) \prec_{tr} (\vartheta(j), s'(m')) \Rightarrow (\vartheta(i), s(m_n)) \prec_{f_{sp}(tr)} (\vartheta(j), s'(m'_1)).$$

We need to show that

$$(\vartheta(i), s(m_n)) \prec_{f_{sp}(tr)} (\vartheta(j), s'(m'_1)) \Rightarrow (\vartheta(i), s(m)) \prec_{tr} (\vartheta(j), s'(m')). \tag{D.18}$$

Suppose that $(\vartheta(i), s(m_n)) \prec_{f_{sp}(tr)} (\vartheta(j), s'(m'_1))$. Then there exists two events $s(t), s'(t') \in tr$ such that

$$\begin{aligned} &(\vartheta(i), s(t)) \prec_{tr} (\vartheta(j), s'(t')), \\ &f_{sp}(s(t)) = s(t_1) \cdots s(t_l), \text{ where } l \geq 1 \text{ and } m_n \in \{t_1, \ldots, t_l\}, \\ &f_{sp}(s'(t')) = s'(t'_1) \cdots s'(t'_v), \text{ where } v \geq 1 \text{ and } m'_1 \in \{t'_1, \ldots, t'_v\}. \end{aligned} \tag{D.19}$$

Similar as above, since P is non-overlapping we must have $t = m$ and $t' = m'$. This by (D.19) implies $(\vartheta(i), s(m)) \prec_{tr} (\vartheta(j), s'(m'))$. Hence (D.18) holds. Moreover, we have

$$\begin{aligned} &(f_{sp}(tr), f_{sp}(th), \sigma, \vartheta) \vDash_E f_{sp}((i, s(m)) \prec (j, s'(m')) \\ &\Leftrightarrow (\vartheta(i), s(m_n)) \prec_{f_{sp}(tr)} (\vartheta(j), s'(m'_1)). \end{aligned}$$

Therefore, we have

$$\begin{aligned} &(tr, th, \sigma, \vartheta) \vDash_E (i, s(m)) \prec (j, s'(m')) \\ &\Leftrightarrow (f_{sp}(tr), f_{sp}(th), \sigma, \vartheta) \vDash_E f_{sp}((i, s(m)) \prec (j, s'(m'))). \end{aligned}$$

- $\phi \equiv \mathit{secret}(m)$.

$$\begin{array}{lll} & (tr, th, \sigma, \vartheta) \nvDash_E \mathit{secret}(m) & \\ \Leftrightarrow & IK(tr)\sigma \cup IK_0 \vdash_E m\sigma & \\ \Rightarrow & f_{sp}(IK(tr))\sigma \cup IK'_0 \vdash_E m\sigma & \text{since } IK_0 \subseteq IK'_0, \text{ and} \\ & & \mathit{split}(f_{sp}(IK(tr))) = \mathit{split}(IK(tr)) \\ \Leftrightarrow & (f_{sp}(tr), f_{sp}(th), \sigma, \vartheta) \nvDash_E \mathit{secret}(m). & \end{array}$$

The inductive cases are routines.

This completes the proof of the theorem. □

E. Justification of Soundness Conditions for Protocol Abstractions

E.1. Justification of $\mathsf{IKE_m}$-to-$\mathsf{IKE_m^1}$ Abstraction

Here, we establish the soundness conditions for the abstraction $F_1 = (f_1, E_{f_1})$ in Example 17 with respect to the properties ϕ_s and ϕ_a expressed in Example 8. For simplicity, let ϕ represent both ϕ_s and ϕ_a. We assume that $IK_0 = IK'_0 = \mathcal{A} \cup \mathcal{C} \cup \mathcal{F}^\bullet \cup \bigcup_{a \in \mathcal{A}, b \in \mathcal{A}_C} \mathsf{sh}(a, b)$. We need to show that the conditions required for Theorem 4 hold. These conditions are:

(i) $f_1(IK_0) \subseteq IK'_0$,

(ii) F_{f_1} is Ax-closed and pattern-disjoint,

(iii) $\mathcal{M}_{\mathsf{IKE_m}} \subseteq udom(F_{f_1}) \cap rdom(F_{f_1})$,

(iv) $Sec_\phi \cup EqTerm_\phi \subseteq udom(F_{f_1}) \cap rdom(F_{f_1})$,

(v) $f_1(t\sigma) =_E f_1(u\sigma)$ implies $t\sigma =_E u\sigma$ for all $(t, u) \in Eq^+_\phi$ and for all well-typed and R,Ax-normal ground substitutions σ, and

(vi) $f_1(t) = f_1(u)$ implies $t = u$, for all $e(t) \in Evt^+_\phi$ and $e(u) \in Evt(\mathcal{M}_{\mathsf{IKE_m}})$.

Note that the last three conditions are an unfolding of condition (iv) in Theorem 4. To justify conditions (i)-(vi), we rely on the following observations.

(O1) $f_1(IK_0) = IK_0 = IK'_0$.

(O2) All terms in $subs(\mathcal{M}_{\mathsf{IKE_m}} \cup Sec_\phi \cup EqTerm_\phi)$ are abstracted using only clauses in E_{f_1}.

(O3) No term in $subs(\mathcal{M}_{\mathsf{IKE_m}} \cup Sec_\phi \cup EqTerm_\phi)$ contains reducible function symbols.

(O4) for all terms $t, u \in \mathcal{M}_{\mathsf{IKE_m}}$ such that $t \neq u$, we have $f_1(t) \neq f_1(u)$.

Condition (i) follows from (O1). To see that F_{f_1} is pattern-disjoint, note that except mac, each function symbol is handled by at most one clause in E_{f_1}. Moreover, the types of the mac patterns in the second and the third clauses, i.e.,

$$\mathsf{mac}(\mathsf{msg}, \mathsf{msg}, \gamma_o, \mathsf{msg}, \mathsf{msg}, \mathsf{msg}, \mathsf{msg}, \mathsf{msg}) \text{ and}$$
$$\mathsf{mac}(\mathsf{msg}, \mathsf{msg}, \mathsf{Nonce}, \mathsf{msg}, \mathsf{msg}, \mathsf{msg}, \mathsf{msg}, \mathsf{msg}),$$

are disjoint. This shows that F_{f_1} is pattern-disjoint. In Example 23, we use a syntactic criterion to show that F_{f_1} is Ax-closed. Hence condition (ii) holds. Condition (iii) holds by (O2) and (O3). Condition (iv) follows from (O2) and (O3). Condition (vi) holds by (O4). To justify condition (ii), note that we can rewrite the equality on the tuples in ϕ_a as a conjunction of equalities on the tuples' components. Since f_1 is the identity on atoms and variables, it suffices to establish condition (v) for the two equalities of the form $X = g(a)$ with X is of type msg and a is an atom. We have formally justified these cases in Example 25.

E.2. Justification of Abstractions in Section 4.5

Note that the properties ϕ_s^A and ϕ_s^B contain no send/receive events and no equalities. Hence for each typed abstraction f_i with $i \in \widetilde{2}$ and the corresponding original protocol P, we only need to check the following conditions:

(i) F_{f_i} is Ax-closed and pattern-disjoint,

(ii) $Sec_\phi \subseteq udom(F_{f_i}) \cap rdom(F_{f_i})$ for $\phi \in \{\phi_s^A, \phi_s^B\}$, and

(iii) $\mathcal{M}_P \subseteq udom(F_{f_i}) \cap rdom(F_{f_i})$.

We see that f_i is pattern-disjoint. Moreover, our set of axioms Ax contains exactly one equation, namely $\mathsf{sh}(X, Y) = \mathsf{sh}(Y, X)$ where $X, Y : \mathsf{msg}$. Hence, by Proposition 4 we derive that f_i is Ax-closed. Therefore, condition (i) holds. Since Sec_ϕ for ϕ contains only non-composed terms, condition (ii) also holds for f_i. To show (iii), note that all protocol subterms $subs(\mathcal{M}_P)$ are transformed by clauses in E_{f_i}. Therefore, we have $\mathcal{M}_P \subseteq udom(F_{f_i})$. Moreover, no term in $subs(\mathcal{M}_P)$ contains reducible function symbols. Hence, we have $\mathcal{M}_P \subseteq rdom(F_{f_i})$. Thus condition (iii) immediately follows.

Since $\mathsf{K3}'$ is non-verlapping, we know that f_{sp} is sound. As no event is involved in the properties, the soundness of rem_{T_1} and rem_{T_2} is trivial. Thus, we conclude that all abstractions are sound.

Bibliography

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL*, pages 104–115, 2001.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.

[3] M. Abadi and L. Lamport. The existence of refinement mappings. *Theor. Comput. Sci.*, 82(2):253–284, May 1991.

[4] J. Abrial, M. Lee, D. Neilson, P. Scharbach, and I. Sørensen. The b-method. In S. Prehn and H. Toetenel, editors, *VDM '91 Formal Software Development Methods*, volume 552 of *Lecture Notes in Computer Science*, pages 398–405. Springer Berlin / Heidelberg, 1991. 10.1007/BFb0020001.

[5] J.-R. Abrial. *The B-book: Assigning Programs to Meanings.* Cambridge University Press, New York, NY, USA, 1996.

[6] M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In V. Arvind and S. Prasad, editors, *FSTTCS*, volume 4855 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2007.

[7] M. Arapinis, E. Ritter, and M. D. Ryan. Statverif: Verification of stateful processes. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*, pages 33–47. IEEE Computer Society, 2011.

[8] J. Arkko and H. Haverinen. RFC 4187: Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA), 2006. `http://www.ietf.org/rfc/rfc4187`.

[9] A. Armando and L. Compagna. SAT-based model-checking for security protocols analysis. *International Journal of Information Security*, 7(1):3–32, 2008.

[10] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[11] M. Backes, A. Cortesi, R. Focardi, and M. Maffei. A calculus of challenges and responses. In *Proceedings of the 2007 ACM Workshop on Formal Methods in Security Engineering*, FMSE '07, pages 51–60, New York, NY, USA, 2007. ACM.

[12] M. Backes and D. Unruh. Computational soundness of symbolic zero-knowledge proofs against active attackers. In *CSF*, pages 255–269. IEEE Computer Society, 2008.

[13] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin. Computer-aided security proofs for the working cryptographer. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011.

[14] D. A. Basin and C. Cremers. Know your enemy: Compromising adversaries in protocol analysis. *ACM Trans. Inf. Syst. Secur.*, 17(2):7, 2014.

[15] D. A. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *Int. J. Inf. Sec.*, 4(3):181–208, 2005.

[16] G. Bella and E. Riccobene. Formal analysis of the Kerberos authentication system. *Journal of Universal Computer Science*, 3(12):1337–1381, 1997.

[17] P. Bieber and N. Boulahia-Cuppens. Formal development of authentication protocols. In *Sixth BCS-FACS Refinement Workshop*, 1994.

[18] P. Bieber, N. Boulahia-Cuppens, T. Lehmann, and E. van Wickeren. Abstract machines for communication security. In *Proc. 6th IEEE Computer Security Foundations Workshop (CSFW)*, pages 137–146, 1993.

[19] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 82–96. IEEE Computer Society, 2001.

[20] B. Blanchet. Security protocols: from linear to classical logic by abstract interpretation. *Inf. Process. Lett.*, 95(5):473–479, 2005.

[21] B. Blanchet. Security protocols: From linear to classical logic by abstract interpretation. *Information Processing Letters*, 95(5):473–479, Sept. 2005.

[22] B. Blanchet. A Computationally Sound Mechanized Prover for Security Protocols. *IEEE Trans. Dependable Sec. Comput.*, 5(4):193–207, 2008.

[23] N. Brownlee. Formal systems (europe) ltd. failures-divergence refinement. fdr2 user manual. available at http://www.formal.demon.co.uk/fdr2manual/index.html. In *Blount MetraTech Corp. Accounting Attributes and Record Formats http://www.ietf.org/rfc/rfc2924.txt*, 2000.

[24] A. D. Brucker and S. Mödersheim. Integrating automated and interactive protocol verification. In P. Degano and J. D. Guttman, editors, *Formal Aspects in Security and Trust, 6th International Workshop, FAST 2009, Eindhoven, The Netherlands,*

November 5-6, 2009, Revised Selected Papers, volume 5983 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2009.

[25] M. Bugliesi, R. Focardi, and M. Maffei. Compositional analysis of authentication protocols. In *In Proceedings of European Symposium on Programming (ESOP 2004*, pages 140–154. Springer-Verlag, 2004.

[26] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, Feb. 1990.

[27] M. Butler and M. Butler. Using refinement to analyse the safety of an authentication protocol, 1998.

[28] M. J. Butler. On the use of data refinement in the development of secure communications systems. *Formal Aspects of Computing*, 14(1):2–34, 2002.

[29] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. pages 55–69. IEEE Computer Society Press, 1999.

[30] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of the 12th IEEE Workshop on Computer Security Foundations*, CSFW '99, pages 55–, Washington, DC, USA, 1999. IEEE Computer Society.

[31] I. Cervesato, C. Meadows, and D. Pavlovic. An encapsulated authentication logic for reasoning about key distribution protocols. In *CSFW '05: Proceedings of the 18th IEEE workshop on Computer Security Foundations*, pages 48–61, Washington, DC, USA, 2005.

[32] C.Kaufman, P. Hoffman, Y. Nir, and P. Eronen. RFC 5996: Internet Key Exchange Protocol Version 2 (IKEv2), September 2010. `http://www.rfc-editor.org/info/rfc5996`.

[33] H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In J. Giesl, editor, *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2005.

[34] H. Comon-Lundh and C. Véronique. Computational soundness of observational equivalence. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 109–118, New York, NY, USA, 2008. ACM.

[35] V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In S. Arun-Kumar and N. Garg, editors, *FSTTCS*, volume 4337 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2006.

[36] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proceedings of the 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 157–171, Edinburgh, U.K, April 2005. Springer.

[37] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In R. M. Graham, M. A. Harrison, and R. Sethi, editors, *POPL*, pages 238–252. ACM, 1977.

[38] C. Cremers. IKEv1 and IKEv2 protocol suites, 2011. `https://github.com/cascremers/scyther/tree/master/gui/Protocols/IKE`.

[39] C. Cremers. ISO/IEC 9798 authentication protocols, 2012. `https://github.com/cascremers/scyther/tree/master/gui/Protocols/ISO-9798`.

[40] C. Cremers and S. Mauw. *Operational Semantics and Verification of Security Protocols*. Information Security and Cryptography. Springer, 2012.

[41] C. J. F. Cremers. The Scyther tool: Verification, falsification, and analysis of security protocols. In A. Gupta and S. Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.

[42] C. J. F. Cremers. Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2. In V. Atluri and C. Díaz, editors, *ESORICS*, volume 6879 of *Lecture Notes in Computer Science*, pages 315–334. Springer, 2011.

[43] C. J. F. Cremers, S. Mauw, and E. P. de Vink. Injective synchronisation: An extension of the authentication hierarchy. *Theor. Comput. Sci.*, 367(1-2):139–161, 2006.

[44] M. Daniele and W. Bogdan. Completeness theorems for the abadi-rogaway language of encrypted expressions. *J. Comput. Secur.*, 12(1):99–129, Jan. 2004.

[45] M. Daniele and W. Bogdan. Soundness of formal encryption in the presence of active adversaries. In M. Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.

[46] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In D. Volpano, editor, *Proceedings of CSFW 2003*, pages 109–125. IEEE, 2003.

[47] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition (extended abstract). In M. Backes, D. Basin, and M. Waidner, editors, *Proceedings of FMCS 2003*, pages 11–23. ACM, 2003.

[48] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Abstraction and refinement in protocol derivation. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, 2004.

[49] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositionl logic for security protocols. *Journal of Computer Security*, 13:423–482, 2005.

[50] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in maude. In *Proc. of Workshop on Formal Methods and Security Protocols*, 1998.

[51] N. Dershowitz and J. Jouannaud. *Rewrite Systems.* 1990.

[52] T. Dierks and E. Rescorla. RFC 4346: The transport layer security (TLS) protocol Version 1.1, April 2006. `http://www.rfc-editor.org/info/rfc4346`.

[53] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, Sept. 2006.

[54] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[55] F. Durán and J. Meseguer. A church-rosser checker tool for conditional order-sorted equational maude specifications. In P. C. Ölveczky, editor, *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Held as a Satellite Event of ETAPS 2010, Paphos, Cyprus, March 20-21, 2010, Revised Selected Papers*, volume 6381 of *Lecture Notes in Computer Science*, pages 69–85. Springer, 2010.

[56] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In N. Heintze and E. Clarke, editors, *Proceedings of the Workshop on Formal Methods and Security Protocols - FMSP, Trento, Italy*, 1999.

[57] N. Durgin, J. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11(4):677–721, 2003.

[58] N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. In S. Schneider, editor, *Proceedings of CSFW 2001*, pages 241–255. IEEE, 2001.

[59] S. Escobar, R. Sasse, and J. Meseguer. Folding variant narrowing and optimal variant termination. *Journal of Logic and Algebraic Programming*, (0):–, 2012.

[60] D. Galindo, F. D. Garcia, and P. van Rossum. Computational soundness of non-malleable commitments. In L. Chen, Y. Mu, and W. Susilo, editors, *Information Security Practice and Experience, 4th International Conference, ISPEC 2008, Sydney, Australia, April 21-23, 2008, Proceedings*, volume 4991 of *Lecture Notes in Computer Science*, pages 361–376. Springer, 2008.

[61] J. Giesl, P. Schneider-Kamp, and R. Thiemann. Automatic termination proofs in the dependency pair framework. In U. Furbach and N. Shankar, editors, *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, pages 281–286. Springer, 2006.

[62] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, pages 270–299, 1984.

[63] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, Apr. 1988.

[64] Y. Gurevich. Specification and validation methods. chapter Evolving Algebras 1993: Lipari Guide, pages 9–36. Oxford University Press, Inc., New York, NY, USA, 1995.

[65] J. D. Guttman. Transformations between cryptographic protocols. In P. Degano and L. Viganò, editors, *ARSPA-WITS*, volume 5511 of *LNCS*, pages 107–123. Springer, 2009.

[66] J. D. Guttman. Security goals and protocol transformations. In *Theory of Security and Applications (TOSCA), an ETAPS associated event*, volume 6993 of *LNCS*. Springer, 2011.

[67] J. D. Guttman. Shapes: Surveying crypto protocol runs, 2011.

[68] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). IETF RFC 2409 (Proposed Standard), November 1998. Obsoleted by RFC 4306, updated by RFC 4109.

[69] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[70] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[71] O. Horvitz and V. D. Gligor. Weak key authenticity and the computational completeness of formal encryption. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 530–547. Springer, 2003.

[72] M. L. Hui and G. Lowe. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Security*, 9(1/2):3–46, 2001.

[73] International Organization for Standardization, Genéve, Switzerland. ISO/IEC 9798-3, Information technology - Security techniques - Entity authentication - Part 1: General, 2010.

[74] International Organization for Standardization, Genéve, Switzerland. ISO/IEC 9798-3, Information technology - Security techniques - Entity authentication - Part 2: Mechanisms using symmetric encipherment algorithms, 2010.

[75] International Organization for Standardization, Genéve, Switzerland. ISO/IEC 9798-3, Information technology - Security techniques - Entity authentication - Part 3: Mechanisms using digital signature techniques, 2010.

[76] R. Janvier, Y. Lakhnech, and L. Mazaré. Computational soundness of symbolic analysis for protocols using hash functions. *Electr. Notes Theor. Comput. Sci.*, 186:121–139, 2007.

[77] J.-P. Jouannaud, C. Kirchner, and H. Kirchner. Incremental construction of unification algorithms in equational theories. In J. Díaz, editor, *ICALP*, volume 154 of *Lecture Notes in Computer Science*, pages 361–373. Springer, 1983.

[78] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM J. Comput.*, 15(4):1155–1194, 1986.

[79] B. S. Kaliski, Jr. An Unknown Key-share Attack on the MQV Key Agreement Protocol. *ACM Trans. Inf. Syst. Secur.*, 4(3):275–288, Aug. 2001.

[80] C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). IETF RFC 5996, September 2010.

[81] S. Kent and K. Seo. Security architecture for the internet protocol. IETF RFC 4301, December 2005. Updated by RFC 6040.

[82] J. Kohl and C. Neuman. RFC 1510: The Kerberos Network Authentication Service (V5), 1993. `http://www.ietf.org/rfc/rfc4187`.

[83] H. Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, CRYPTO'05, pages 546–566, Berlin, Heidelberg, 2005. Springer-Verlag.

[84] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Des. Codes Cryptography*, 28(2):119–134, Mar. 2003.

[85] Y. Li, W. Yang, and C. Huang. On preventing type flaw attacks on security protocols with a simplified tagging scheme. *J. Inf. Sci. Eng.*, 21(1):59–84, 2005.

[86] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '96, pages 147–166, London, UK, UK, 1996. Springer-Verlag.

[87] G. Lowe. A hierarchy of authentication specifications. In *IEEE Computer Security Foundations Workshop*, pages 31–43, Los Alamitos, CA, USA, 1997. IEEE Computer Society.

[88] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(1):89–146, 1999.

[89] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[90] N. A. Lynch. I/O automaton models and proofs for shared-key communication systems. In *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW)*, pages 14–29, 1999.

[91] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. Technical Report MIT/LCS/TM-373, Laboratory for Computer Science, Massachusetts Institute of Technology, 1988.

[92] A. Martín and R. Phillip. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics*, TCS '00, pages 3–22, London, UK, UK, 2000. Springer-Verlag.

[93] S. Meier, C. J. F. Cremers, and D. A. Basin. Strong invariants for the efficient construction of machine-checked protocol security proofs. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 231–245. IEEE Computer Society, 2010.

[94] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[95] A. Menezes. Another look at HMQV. *IACR Cryptology ePrint Archive*, 2005:205, 2005.

[96] C. Meyer and J. Schwenk. Lessons Learned From Previous SSL/TLS Attacks - A Brief Chronology Of Attacks And Weaknesses. *IACR Cryptology ePrint Archive*, 2013:49, 2013.

[97] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. In *IN PROJECT ATHENA TECHNICAL PLAN*, 1987.

[98] S. Mödersheim. Deciding security for a fragment of aslan. In S. Foresti, M. Yung, and F. Martinelli, editors, *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, volume 7459 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 2012.

[99] S. Mödersheim and L. Viganò. The open-source fixed-point model checker for symbolic analysis of security protocols. In A. Aldini, G. Barthe, and R. Gorrieri, editors, *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, volume 5705 of *Lecture Notes in Computer Science*, pages 166–194. Springer, 2009.

[100] S. Mödersheim. On the relationships between models in protocol verification. *Inf. Comput.*, 206(2-4):291–311, 2008.

[101] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[102] D. M. Nessett. A Critique of the Burrows, Abadi and Needham Logic. *SIGOPS Oper. Syst. Rev.*, 24(2):35–38, Apr. 1990.

[103] B. T. Nguyen and C. Sprenger. Sound security protocol transformations. In D. A. Basin and J. C. Mitchell, editors, *POST*, volume 7796 of *Lecture Notes in Computer Science*, pages 83–104. Springer, 2013.

[104] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[105] L. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.

[106] D. Pavlovic. Categorical logic of names and abstraction in action calculus. *Math. Structures in Comp. Sci.*, 7:619–637, 1997.

[107] D. Pavlovic and C. Meadows. Deriving secrecy in key establishment protocols. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS)*, pages 384–403, 2006.

[108] A. Perrig, J. D. Tygar, D. Song, and R. Canetti. Efficient authentication and signing of multicast streams over lossy channels. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 56–, Washington, DC, USA, 2000. IEEE Computer Society.

[109] G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *J. ACM*, 28(2):233–264, 1981.

[110] N. Polikarpova and M. Moskal. Verifying implementations of security protocols by refinement. In R. Joshi, P. Müller, and A. Podelski, editors, *VSTTE*, volume 7152 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2012.

[111] R. Ramanujam and S. P. Suresh. A decidable subclass of unbounded security protocols, 2003.

[112] J. D. Ramsdell and J. D. Guttman. CPSA: A cryptographic protocol shapes analyzer. In *Hackage.The MITRE Corporation*, 2009. `https://hackage.haskell.org/package/cpsa`.

[113] J. Romain, L. Yassine, and M. Laurent. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In S. Sagiv, editor, *ESOP*, volume 3444 of *Lecture Notes in Computer Science*, pages 172–185. Springer, 2005.

[114] A. W. Roscoe. A classical mind. chapter Model-checking CSP, pages 353–378. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1994.

[115] B. Schmidt, S. Meier, C. J. F. Cremers, and D. A. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In S. Chong, editor, *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 78–94. IEEE, 2012.

[116] S. Schneider. Verifying authentication protocols with CSP. In *10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA*, pages 3–17. IEEE Computer Society, 1997.

[117] S. A. Shaikh, V. J. Bush, and S. A. Schneider. Specifying authentication using signal events in csp. *Computers & Security*, 28(5):310–324, 2009.

[118] C. Sprenger and D. Basin. Developing security protocols by refinement. In *Proc. 17th ACM Conference on Computer and Communications Security (CCS)*, pages 361–374, October 4-8, Chicago, IL, USA, 2010.

[119] C. Sprenger and D. Basin. Refining key establishment. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF)*, pages 230–246, 2012.

[120] P. Syverson. The use of logic in the analysis of cryptographic protocols. In *IEEE Symposium on Security and Privacy*, pages 156–170, 1991.

[121] F. L. Tiplea, C. Enea, and C. V. Birjoveanu. Decidability and complexity results for security protocols. In E. M. Clarke, M. Minea, and F. L. Tiplea, editors, *Verification of Infinite-State Systems with Applications to Security, Proceedings of the NATO Advanced Research Workshop "Verification of Infinite State Systems with Applications to Security VISSAS 2005", Timisoara, Romania, March 17-22, 2005*, volume 1 of *NATO Security through Science Series D: Information and Communication Security*, pages 185–211. IOS Press, 2005.

[122] M. Turuani. The CL-Atse protocol analyser. In F. Pfenning, editor, *RTA*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2006.

[123] C. Véronique, K. Steve, and W. Bogdan. A survey of symbolic methods in computational analysis of cryptographic systems. *Journal of Automated Reasoning*, 46:225–259, Apr. 2010.

[124] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. pages 314–328. Springer-Verlag, 1999.

[125] A. C. Yao. Theory and application of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 80–91, Washington, DC, USA, 1982. IEEE Computer Society.