EXTENDING THE LANGUAGE AND APPLICATIONS OF MAUDE-NPA THROUGH REWRITING SEMANTICS

BY

FAN YANG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

ABSTRACT

Formal methods have been used in analyzing cryptographic protocols since the 1980's. Formal analysis of cryptographic protocols involves properties that are generally undecidable; however it can often be automated. Maude-NPA is a special-purpose tool for verifying cryptographic protocols. Based on rewriting logic, Maude-NPA performs backward symbolic model checking on the unbounded session model, considering user defined signature and a wide range of equational theories. In this way, various properties, including secrecy, authentication and indistinguishability can be verified.

This thesis investigates and advances cryptographic protocol modeling and analysis, with a focus on extending the specification and analysis capabilities of the Maude-NPA tool. In particular, (i) it presents a hierarchy of FVP theories for approximating the algebraic property of homomorphic encryption over an Abelian group, which enables analysis of protocols having homomorphic encryption over abelian group in Maude-NPA; (ii) it extends the strand space model with support for choice, and develops a protocol process algebra with choice constructors; as a result, a new specification language is provided for Maude-NPA, and protocols with choices can be model and analyzed naturally in Maude-NPA; (iii) it develops a methodology for modular analysis of protocol composition for private channels: the security properties of the composed protocols are decomposed into corresponding properties of each component protocols. In each of these areas (i)-(iii), experiments are performed in Maude-NPA to illustrate and validate these approaches.

To my parents and husband, for their love and support.

ACKNOWLEDGMENTS

I would like to thank my advisor Jose Meseguer for his support and guidance. I am grateful for his encouragement on deep thinking, and all the invaluable discussions. His knowledge and passion for research has been inspiring me throughout this journey.

I am grateful for having had the pleasure to collaborate with Santiago Escobar and Catherine Meadows. I had many great discussion with them and learned so much from them. I am also grateful to have the opportunity to collaborate with Paliath Narendran. I also would like to thank Sonia Santiago. I had one of the best semesters working with her.

I am also very thankful to my committee members Gul Agha and Grigore Rosu, for all their kindness and help throughout this work.

I am grateful to the office mates, fellow students for their accompany, help, and great memories: Camilo Rocha, Ralf Sasse, Kyungmin Bae, Stephen Skeirik, Si Liu, Andrew Cholewa, Liyi li, Andrei Stefanescu, and Milos Gligoric.

Finally, I thank my parents and my husband, for their love, support and patience.

The work in this dissertation was partially supported by NSF grant CNS-131910, by the US Air Force Office of Scientific Research under award number FA9550-17-1-0286, and by NRL under contract number N00173-17-1-G002.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

There is a long history of analyzing cryptographic protocols using formal methods. Formal methods have been effective in revealing subtle flaws that would otherwise remain buried in protocols [1, 2, 3, 4, 5, 6]. One important advantage of verifying cryptographic protocols using formal methods is that some of the verification methods allow automatic verification. For example, by modeling the protocol using a process algebra model (i.e., CSP [7]) and using the general purpose model checker FDR, Lowe [1] found the man-in-the-middle attack for Needham-Shroeder public key protocol [8], a protocol that had been widely used before the attack was found. At around the same time, Mitchell et al. [9] also used the Murphi [10] model checker for analyzing variations on the TLS protocol, and Lawrence C. Paulson [11] used Isabelle/HOL theorem prover to verify security properties of TLS.

Being inspired by the benefits of formal analysis, a number of special-purpose tools for symbolic verification of cryptographic protocols have been developed [12, 13, 14, 15, 16, 17, 18, 19]. These tools assume a Dolev-Yao model [20], in which the communication is fully controlled by the intruder who can eavesdrop the communication, redirect messages, create and send new messages or change messages. Messages exchanged are represented by elements in a term algebra and the cryptographic primitives are represented by function symbols. That is, cryptographic functions are treated symbolically assuming perfect cryptography. Although there can be attacks that are possible in the computational model but are not captured in the symbolic model due to the abstract representations and assumptions like perfect cryptography, the symbolic proofs are relatively easier to be automated and can still capture many practically relevant attacks.

The state space of the protocols is generated by considering both the expected protocol behaviors and the actions of the intruder. The security properties are verified by exhaustively searching the state space of the protocols for security violations (model checking approach), or constructing proofs for theorems stating the security properties by induction on traces (the theorem proving approach). Some of the tools lean more towards the theorem proving approach. The Tamarin tool [13] specifies protocols using multiset rewriting rules. The properties are specified in a guarded fragment of first-order logic and checked against the traces of the transition system by constructing proofs automatically or interactively. The analyzer Proverif [12] represents the protocol by Horn clauses, and verifies the security properties through resolution theorem proving. Many other tools are based on the model checking approach, more specifically, symbolic model checking, in which messages and states are represented as symbolic patterns. The tool Scyther [21, 15] represents protocol behavior as events, sets of events as symbolic trace patterns, and performs backwards search to check whether the trace patterns can be realized. AVISPA [14] includes a set of model checkers, e.g., CL-Atse [17], SAT-MC [18] and OFMC [19], which share a common front end. In Maude-NPA [16], cryptographic protocols are represented as rewrite theories, and the security properties are specified as attack states and are verified by performing backwards (i.e., from an attack state to an initial state) symbolic reachability analysis based on backwards narrowing and equational unification. But Maude-NPA also has theorem proving aspects in terms of managing the state space, which we will mention later.

The state space generated can be infinite. Durgin et al. [22] have shown that the secrecy problem is undecidable for unbounded number of sessions and nonces. But it was later shown by Rusinowitch and Turuani [23, 24] that it is NP-complete if the number of sessions is bounded, assuming just encryption and decryption primitives. Inspired by this result, some tools choose to analyze protocol for bounded sessions. For example, the Scyther tool we mentioned before gives options for choosing between bounded session analysis and unbounded session analyses. The model checkers CL-Atse, SAT-MC and OFMC perform analysis assuming bounded sessions. Instead, Proverif, Tamarin and Maude-NPA perform analysis on an unbounded session model, and therefore they cannot guarantee termination. Proverif uses abstraction, while Maude-NPA has a built-in induction procedure that rules out certain non-terminating paths. Tamarin combines induction and heuristics. When the automatic proof construction cannot terminate, it also allows interactive proof by the user.

When it comes to modeling protocols, there have been two popular ways, used in both tools and hand written formal proofs. One is process algebra and another one is strand spaces. In the strand space model [25], the local behaviors of each role is denoted by a strand, which is a sequence of send and receive events. In the process algebra model (e.g., applied pi calculus) [26, 27], each role is represented by a process that sends and receives messages from the network (via channels). Protocol specification in Maude-NPA is based on the strand space model. Strands are used to represent both the actions of honest principals, with a strand specified for each protocol role, and the actions of an intruder. Each strand consists of a sequence of input/output message terms, denoting receiving and sending messages.

Much of the early research in symbolic verification of cryptographic protocols assumed a free algebra model, in which the crypto system is treated as a "black box", the intruder can learn the original plain text from an encrypted message only when it knows the exact decryption key. This is the approach taken by the previously mentioned papers [1] and [9]. It has been realized that such an assumption can be too strong. There can be attacks missed by the verification tools because of that [28, 29, 30]. Therefore, protocol verification considering the algebraic properties of the cryptographic primitives have become an important feature of cryptographic protocol analysis tools. We have seen continuing effort on extending the tools to handle additional theories [31, 16, 32, 33, 34, 35, 36]. These tools represent states as symbolic patterns and algebraic properties as equational theories. The algebraic properties are then considered in the analysis by exploring the states via unification modulo an equational theory describing the algebraic properties.

One desirable property of the unification algorithm used in cryptographic protocol analysis is that it can easily accommodate combinations of theories of interest and can be integrated with state space exploration techniques. Variant unification, first formalized as a general approach in [37], although used for specific theories much earlier than that (e.g. in [38, 31, 39]), satisfies all these properties. Thus, it is used by many cryptographic protocol analysis tools in one form or another, including ProVerif [38], OFMC [31], Maude-NPA [16] and Tamarin [35]. Thanks to its rewriting semantics, Maude-NPA has shown its great advantages in handling algebraic properties based on variant unification.

Although protocols may be designed alone and analyzed alone, they do not always work alone. It is also not uncommon that protocols are designed with assumptions that will be guaranteed by other protocols. The importance of understanding composition has been acknowledged [40, 41, 42, 43, 44, 45, 46]. Maude-NPA has extended its language and semantics to provides both a specification language and automatic verification methods that support protocol composition [47, 48], which also provides a base for further study on problems related to protocol composition in Maude-NPA as done in this thesis.

1.1 SUMMARY OF CONTRIBUTIONS

In this thesis, we present the techniques that we have developed to extend Maude-NPA's capabilities on modeling and analyzing cryptographic protocols. We present our research results in addressing the following challenges:

- How to analyze protocols in which the algebraic properties of the cryptographic primitives cannot be represented by equational theories having the finite variant property?
- How can we naturally model and analyze protocols with choice?
- How can we analyze protocols that are built by composing subprotocols while avoiding state explosion problems?

1.1.1 Theories of Homomorphic Encryption, Unification, and the FVP.

Variant unification requires, among other things, that the equational theory be decomposable into a set of axioms B and a set of convergent rewrite rules R such that R has the *finite variant property* with respect to B [37, 49, 50]. Meaning that, for each term t in the theory, there is a finite number of most general normal forms for the set of normal forms of all substitution instances of t. Although most theories that arise in cryptographic protocols have decompositions suitable for variant unification, there is one major exception: the theory that describes encryption that is homomorphic over an Abelian group. That is, a homomorphic property of the form $e(X * Y, K) = e(X, K) * e(Y, K)$ where $*$ obeys the Abelian group axioms. AGH (the case where $*$ is an abelian Group) is a property belonging to a number of different cryptographic algorithms, starting with RSA in the late 70's [51].

In Chapter 3 we address this problem by studying various approximations of homomorphic encryption over an Abelian group. We construct a hierarchy of theories approximating homomorphic encryption. The theories are verified to have the finite variant property. As a result, we have developed finitary unification algorithms for these theories, many for which are new unification results that had not previously been developed. We also construct a rough metric about the complexity of a theory with respect to variant unification, i.e., *variant complexity*. We specify different versions of protocols using the different theories, and analyze them in the Maude-NPA cryptographic protocol analysis tool to assess their behavior. This provides an experimental evaluation of the performance tradeoff between the faithfulness and variant complexity of the theories used in the analysis.

1.1.2 Protocol Process Algebra and Strands with Choice.

Roles in cryptographic protocols do not always have a linear execution: they may include choice points causing the protocol to continue along different paths. The strand space model [25] does not support choice in its original form; strands describe linear sequences of input and output messages, without any branching. One response to dealing with this limitation, and to formalizing strand spaces in general has been to embed the strand space model in some other formal system that supports choice [52, 53, 54].

To provide support for *choice*, a commonly used construct, Chapter 4 describes a choice model that we have developed for the Maude-NPA. We develop and give formal semantics to a process algebra for cryptographic protocols that supports a rich taxonomy of choice primitives for composing strand spaces, including deterministic and non-deterministic choices.

We have integrated the process algebra syntax in Maude-NPA. This allows one to write protocols using the process syntax, which is more convenient for expressing choice than the strand space syntax. Its expressive power and naturalness are illustrated with various examples. The correctness of our approach is supported by a soundness and completeness proof of an extended strand space choice semantics with respect to the process algebra semantics.

1.1.3 Modular Verification of Sequential Composition for Private Channels.

Security protocols often depend on other protocols to generate the keys and other values they use to communicate securely. This can be modeled in terms of protocol composition: the protocol receiving the keys runs as a subroutine of the protocol that generates the keys. But examining composed protocols can lead to state space explosion. Although some previous results on protocol composition and analyzing composed protocol in Maude-NPA had been developed [47, 48], those earlier approaches still lacked effective modular analysis techniques.

In Chapter 5, we present a *modular verification* methodology in which, given parametric specifications of a key establishment protocol P and a protocol Q providing private channel communication, verification of security properties of their sequential composition $P\ ;\ Q$ can be reduced to verification of corresponding properties for P, and of an abstract version Q^α of Q. Our results both support a large class of equational theories and security properties and provide effective tool support via Maude-NPA. The semantic basis of this methodology is provided by two simulation relations.

1.2 THESIS PLAN

The rest of this thesis is organized as follows. In Chapter 2 we recall some preliminaries needed for understanding this thesis. In Chapter 3 we provide a summary of Maude-NPA, the cryptographic protocol analyzer that the techniques of this thesis are based on. Chapter 4 introduces the FVP theories we developed for homomorphic encryption. We present in Chapter 4 the protocol process algebra with choice, and its soundness and completeness results. Chapter 5 presents a methodology for modular analysis of sequential protocol compositions for private channels in Maude-NPA. Chapter 6 concludes the thesis and discusses some possible future work.

CHAPTER 2: PRELIMINARIES

We follow the classical notation and terminology for term rewriting and for rewriting logic and order-sorted notions, see [55, 56]. We assume an order-sorted signature $\mathbf{\Sigma} = (\mathsf{S}, \leqslant, \Sigma)$ with poset of sorts $(\mathsf{S}, \leqslant)$. We also assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_{\mathsf{s}}\}_{\mathsf{s} \in \mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_{\mathsf{s}}$ countably infinite. $\mathcal{T}_{\Sigma}(\mathcal{X})_{\mathsf{s}}$ is the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort s. We write $\mathcal{T}_{\Sigma}(\mathcal{X})$ and $\mathcal{T}_{\Sigma}$ for the corresponding order-sorted term algebras. For a term t, $\mathit{Var}(t)$ denotes the set of variables in t.

A *substitution* $\sigma \in \mathcal{S}ubst(\Sigma, \mathcal{X})$ is a sorted mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_{\Sigma}(\mathcal{X})$. Substitutions are written as $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where the domain of σ is $\mathit{Dom}(\sigma) = \{X_1, \ldots, X_n\}$ and the set of variables introduced by terms $t_1, \ldots, t_n$ is written $\mathit{Ran}(\sigma)$. The identity substitution is denoted *id*. Substitutions are homomorphically extended to $\mathcal{T}_{\Sigma}(\mathcal{X})$. The application of a substitution σ to a term t is denoted by $t\sigma$. For simplicity, we assume that every substitution is idempotent, i.e., σ satisfies $\mathit{Dom}(\sigma) \cap \mathit{Ran}(\sigma) = \varnothing$. This ensures $t\sigma = (t\sigma)\sigma$. The restriction of σ to a set of variables V is written $\sigma|_V$. Composition of two substitutions σ and σ' is denoted by $\sigma\sigma'$. A substitution σ is a ground substitution if $\mathit{Ran}(\sigma) = \varnothing$.

Positions in a term are represented by sequences of natural numbers denoting an access path in the term when viewed as a tree. The top or root position is denoted by the empty sequence Λ. Given $U \subseteq \Sigma \cup \mathcal{X}$, $\mathit{Pos}_U(t)$ denotes the set of positions of a term t that are rooted by symbols or variables in U. The set of positions of a term t is written $\mathit{Pos}(t)$, and the set of non-variable positions $\mathit{Pos}_{\Sigma}(t)$. The subterm of t at position p is written $t|_p$ and $t[u]_p$ denotes the term t where $t|_p$ is replaced by u.

A Σ-*equation* is an unoriented pair $t = t'$, where $t, t' \in \mathcal{T}_{\Sigma}(\mathcal{X})_{\mathsf{s}}$ for some sort $\mathsf{s} \in \mathsf{S}$. Given Σ and a set E of Σ-equations, order-sorted equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}_{\Sigma}(\mathcal{X})$. The E-equivalence class of a term t is denoted by $[t]_E$ and $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ and $\mathcal{T}_{\Sigma/E}$ denote the corresponding order-sorted term algebras modulo E. An *equational theory* (Σ, E) is a pair with Σ an order-sorted signature and E a set of Σ-equations. The E-*subsumption* preorder $\sqsupseteq_E$ (or just $\sqsupseteq$ if E is understood) holds between $t, t' \in \mathcal{T}_{\Sigma}(\mathcal{X})$, denoted $t \sqsupseteq_E t'$ (meaning that t is *more general* than t' modulo E), if there is a substitution σ such that $t\sigma =_E t'$; such a substitution σ is said to be an E-*match* from t' to t. A set E of Σ-equations is *regular* if for each $t = t'$ in E, $\mathit{Var}(t) = \mathit{Var}(t')$. A set E of Σ-equations is *sort-preserving* if for each $t = t'$ in E and for each substitution σ, $t\sigma$ has sort s iff $t'\sigma$ has sort s. A set E of Σ-equations uses *top-sort variables* if for each $t = t'$ in E, the sort of each variable in $\mathit{Var}(t) \cup \mathit{Var}(t')$ is a top sort in the sort order $(S, \leqslant)$.

For a set E of Σ-equations, an E*-unifier* for a Σ-equation $t = t'$ is a substitution σ such that $t\sigma =_E t'\sigma$. For $Var(t) \cup Var(t') \subseteq W$, a set of substitutions $CSU_E^W(t = t')$ is said to be a *complete* set of unifiers for the equality $t = t'$ modulo E away from W iff: (i) each $\sigma \in CSU_E^W(t = t')$ is an E-unifier of $t = t'$; (ii) for any E-unifier ρ of $t = t'$ there is a $\sigma \in CSU_E^W(t = t')$ such that $\sigma|_W \sqsupseteq_E \rho|_W$; (iii) for all $\sigma \in CSU_E^W(t = t')$, $Dom(\sigma) \subseteq (Var(t) \cup Var(t'))$ and $Ran(\sigma) \cap W = \varnothing$. If the set of variables W is irrelevant or is understood from the context, we write $CSU_E(t = t')$ instead of $CSU_E^W(t = t')$. An E-unification algorithm is *complete* if for any equation $t = t'$ it generates a complete set of E-unifiers. A unification algorithm is said to be *finitary* and complete if it always terminates after generating a finite and complete set of solutions.

A *rewrite rule* is an oriented pair $l \to r$, where[1] $l \notin \mathcal{X}$ and $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s}}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *(unconditional) order-sorted rewrite theory* is a triple (Σ, E, R) with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules.

The rewriting relation on $\mathcal{T}_\Sigma(\mathcal{X})$, written $t \to_R t'$ or $t \to_{p,R} t'$ holds between t and t' iff there exist $p \in Pos_\Sigma(t)$, $l \to r \in R$ and a substitution σ, such that $t|_p = l\sigma$, and $t' = t[r\sigma]_p$. The subterm $t|_p$ is called a *redex*. The relation $\to_{R/E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $=_E; \to_R; =_E$, i.e., $t \to_{R/E} t'$ iff there exists u, u' s.t. $t =_E u \to_R u' =_E t'$. Note that $\to_{R/E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ induces a relation $\to_{R/E}$ on the free (Σ, E)-algebra $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ by $[t]_E \to_{R/E} [t']_E$ iff $t \to_{R/E} t'$. The transitive (resp. transitive and reflexive) closure of $\to_{R/E}$ is denoted $\to_{R/E}^+$ (resp. $\to_{R/E}^*$).

The $\to_{R/E}$ relation can be difficult to compute. However, under the appropriate conditions it is equivalent to the R,E relation in which it is enough to compute the relationship on any representatives of two E-equivalence classes. A relation $\to_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as: $t \to_{p,R,E} t'$ (or just $t \to_{R,E} t'$) iff there exist $p \in Pos_\Sigma(t)$, a rule $l \to r$ in R, and a substitution σ such that $t|_p =_E l\sigma$ and $t' = t[r\sigma]_p$. We denote by $t!_{R,E}$ the R, E-normal form of term t.

A *decomposition* (Σ, B, R) of an equational theory $(\Sigma, E \uplus B)$ is a rewrite theory that satisfies the following properties: (i) B is regular, sort-preserving and uses top-sort variables, (ii) B has a finitary unification algorithm, and (iii) the rules R are the left-to-right orientation as rules of the equations E and are *convergent* modulo B, i.e., sort-decreasing, confluent, terminating, and coherent modulo B. Given a decomposition $\mathcal{R} = (\Sigma, B, R)$, a variant of a term t is a pair (t', θ) such that t' is a $\to_{R,B}$-canonical form of the substitution instance $t\theta$, i.e., there is a term t'' such that $t\theta \to_{R,B}^* t''$, t'' is a $\to_{R,B}$-normal form, and $t' =_B t''$. A

[1]We do not impose the requirement $Var(r) \subseteq Var(l)$, since extra variables (e.g., choice variables) may be introduced in the righthand side of a rule. Rewriting with extra variables in righthand sides is handled by allowing the matching substitution to instantiate these extra variables in any possible way. Although this may produce an infinite number of one-step concrete rewrites from a term due to the infinite number of possible instantiations, the symbolic, narrowing-based analysis used by Maude-NPA and explained below can cover all those infinite possibilities in a finitary way.

decomposition (Σ, B, R) of an equational theory $(\Sigma, E \uplus B)$ has the *finite variant property* (FVP) if there is a complete and finite set of variants for each term (see [50, 57] for details). If a decomposition (Σ, B, R) of an equational theory $(\Sigma, E \uplus B)$ has the *finite variant property*, there is an algorithm to compute a finite complete set $CSU_{E \uplus B}(t = t')$ of $(E \uplus B)$-unifiers [57].

Let t be a term and W be a set of variables such that $Var(t) \subseteq W$, the R, E-*narrowing* relation on $\mathcal{T}_{\Sigma}(\mathcal{X})$ is defined as $t \rightsquigarrow_{p,\sigma,R,E} t'$ ($\rightsquigarrow_{\sigma,R,E}$ if p is understood, $\rightsquigarrow_{\sigma}$ if R, E are also understood, and $\rightsquigarrow$ if σ is also understood) if there is a non-variable position $p \in Pos_{\Sigma}(t)$, a rule $l \to r \in R$ properly renamed s.t. $(Var(l) \cup Var(r)) \cap W = \emptyset$, and a unifier $\sigma \in CSU_{E}^{W'}(t|_p = l)$ for $W' = W \cup Var(l)$, such that $t' = (t[r]_p)\sigma$. For convenience, in each narrowing step $t \rightsquigarrow_{\sigma} t'$ we only specify the part of σ that binds variables of t. The transitive (resp. transitive and reflexive) closure of $\rightsquigarrow$ is denoted by $\rightsquigarrow^{+}$ (resp. $\rightsquigarrow^{*}$). We may write $t \rightsquigarrow_{\sigma}^{k} t'$ if there are $u_1, \ldots, u_{k-1}$ and substitutions $\rho_1, \ldots, \rho_k$ such that $t \rightsquigarrow_{\rho_1} u_1 \cdots u_{k-1} \rightsquigarrow_{\rho_k} t'$, $k \geqslant 0$, and $\sigma = \rho_1 \cdots \rho_k$.

CHAPTER 3: OVERVIEW OF MAUDE-NPA

In this chapter, we give a summary of Maude-NPA, the cryptographic protocol analyzer that this thesis is based on. Further details can be found in [58, 59, 16, 60].

3.1 STATE, STRANDS, PROTOCOL SPECIFICATION

In Maude-NPA, a protocol is modeled as a set of rewrite rules. Specifically, as a rewrite theory $\mathcal{P}$ that describes the actions of honest principals communicating across a network controlled by an intruder. That is, $\mathcal{P}$ is a rewrite theory of the form $(\Sigma_{SS_{\mathcal{P}}}, E_{SS_{\mathcal{P}}}, R^{-1}_{B_{\mathcal{P}}})$, where (i) the signature $\Sigma_{SS_{\mathcal{P}}}$ includes predefined symbols and user-definable symbols denoting the cryptographic functions of the protocol and are based on a parametric sort Msg of messages, (ii) the algebraic properties $E_{SS_{\mathcal{P}}}$ include the algebraic properties of the cryptographic functions and the strand notation, and (iii) the transition rules $R^{-1}_{B_{\mathcal{P}}}$ are defined on states, i.e., terms of a predefined sort State. They are *reversed* for backwards symbolic execution. The states are modeled as elements of an initial algebra $\mathcal{T}_{\Sigma_{SS_{\mathcal{P}}}/E_{SS_{\mathcal{P}}}}$, i.e., an $E_{SS_{\mathcal{P}}}$-equivalence class $[t]_{E_{SS_{\mathcal{P}}}} \in \mathcal{T}_{\Sigma_{SS_{\mathcal{P}}}/E_{SS_{\mathcal{P}}}}$ with t a ground $\Sigma_{SS_{\mathcal{P}}}$-term.

A *state* in Maude-NPA has the form

$$\{S_1 \,\&\, \cdots \,\&\, S_n \,\&\, \{IK\}\}$$

where & is an infix associative-commutative union operator with identity symbol $\emptyset$. Each element in the set is either a *strand* S_i or the *intruder knowledge* $\{IK\}$ at that state.

The *intruder knowledge* $\{IK\}$ belongs to the state and is represented as a set of facts using comma as an infix associative-commutative union operator with identity element *empty*. There are two kinds of intruder facts: *positive* knowledge facts (the intruder knows m, i.e., $m{\in}\mathcal{I}$), and *negative* knowledge facts (the intruder *does not yet know* m but *will know it in a future state*, i.e., $m{\notin}\mathcal{I}$), where m is a message expression.

A *strand* [25] S_i specifies the sequence of messages sent and received by a principal executing the protocol and is represented as a sequence of messages

$$[msg_1^{\pm}, msg_2^{\pm}, msg_3^{\pm}, \ldots, msg_{k-1}^{\pm}, msg_k^{\pm}]$$

with $msg_i^{\pm}$ either msg_i^{-} (also written $-msg_i$) representing an input message, or msg_i^{+} (also written $+msg_i$) representing an output message. Note that each msg_i is a term of the special sort Msg. Variables of a special sort Fresh are used to represent pseudo-random values (nonces) and Maude-NPA ensures that two distinct fresh variables will never be merged. Strands are prefixed with all the fresh variables $r_1, \ldots, r_k$ created by that strand, i.e.,

$$:: r_1, \ldots, r_k :: [msg_1^{\pm}, msg_2^{\pm}, \ldots, msg_k^{\pm}]$$

.

Strands are used to represent both the actions of honest principals (with a strand specified for each protocol role) and also the actions of an intruder. In Maude-NPA strands evolve over time; the symbol | is used to divide past and future. That is, given a strand

$$[\ msg_1^{\pm},\ \ldots,\ msg_i^{\pm} \mid msg_{i+1}^{\pm},\ \ldots,\ msg_k^{\pm}\ ],$$

messages $msg_1^{\pm}, \ldots, msg_i^{\pm}$ are the *past messages*, and messages $msg_{i+1}^{\pm}, \ldots, msg_k^{\pm}$ are the *future messages* ($msg_{i+1}^{\pm}$ is the immediate future message). A strand $[msg_1^{\pm}, \ldots, msg_k^{\pm}]$ is shorthand for $[nil \mid msg_1^{\pm}, \ldots, msg_k^{\pm}, nil]$. An *initial state* is a state where the bar is at the beginning for all strands in the state, and the intruder knowledge has no fact of the form $m{\in}\mathcal{I}$. A *final state* is a state where the bar is at the end for all strands in the state and there is no intruder fact of the form $m{\notin}\mathcal{I}$.

Example 3.1. The Needham-Schroeder public key (NSPK) protocol [8] is as follows:

1. $A \rightarrow B : pk(B, A; N_A)$
2. $B \rightarrow A : pk(A, N_A; N_B)$
3. $A \rightarrow B : pk(B, N_B)$

where N_A and N_B are nonces generated by the respective principals.

To specify the honest protocol participants, we represent each role in the protocol as a strand. There are two roles in the NSPK protocol, we therefore define two strands:

```
 :: r ::
[nil | +(pk(B, A;n(A,r))), -(pk(A, n(A,r);NB)), +(pk(B, NB)), nil]
 :: r' ::
[nil | -(pk(B, A;NA)), +(pk(A, NA;n(B,r'))), -(pk(B, n(B,r'))), nil]
```

The intruder capabilities are specified in Maude-NPA by Dolev-Yao strands. A Dolev-Yao strand consists of a sequence of negative messages, followed by a single positive message.

Example 3.2. The Dolev-Yao strands for the NSPK protocol are as follows.

```
    :: nil :: [ nil | -(X), -(Y), +(X ; Y), nil ] &
    :: nil :: [ nil | -(X ; Y), +(X), nil ] &
    :: nil :: [ nil | -(X ; Y), +(Y), nil ] &
    :: nil :: [ nil | -(X), +(sk(i,X)), nil ] &
    :: nil :: [ nil | -(X), +(pk(A,X)), nil ]
```

The symbol & is the union operator for sets of strands. The operator ; denotes concatenation of messages. If the intruder knows messages X and Y, he can construct $X;Y$. If he knows $X;Y$, he can find X and Y. The operators *sk* and *pk* denote encryption with a private key and encryption with a public key respectively. The principal's name is used to stand for the key. The intruder can only apply the *sk* operator using his own identity. Note that the shared variables are automatically renamed when necessary.

Attack patterns define final states used for backwards search. The attack patterns are specified symbolically as terms (with variables) whose instances are the final attack states we are looking for. Since more than one attack patterns can be specified, we designate each attack pattern with a natural number.

Example 3.3. In the NSPK protocol, to check the secrecy attack in which an instance of Bob's role has finished its execution (by having the vertical bar at the end), and the intruder has learned the nonce, `n(b,r')`, generated by this Bob's instance, we specify the following attack pattern:

```
eq ATTACK-STATE(0) =
   :: r' ::
  [ nil,
   -(pk(b,a ; NA)),
   +(pk(a, NA ; n(b,r'))),
   -(pk(b,n(b,r'))) | nil ]
   || n(b,r) inI
   || nil || nil || nil
[nonexec] .
```

3.2 ALGEBRAIC PROPERTIES

Maude-NPA performs symbolic reachability analysis *modulo* the equational theory expressing the algebraic properties of the protocol cryptographic functions. This makes Maude-NPA verification much stronger than verification methods based on a purely syntactic view of the algebra of messages. It is well-know that protocols "proved" secure without taking into account the algebraic properties of their cryptographic functions can sometimes be broken [28, 29, 30] .

There are three types of algebraic properties that can be specified in Maude-NPA:

1. *axioms*, allowing any combination of associativity (A), commutativity (C), and identity (U) axioms,

2. *equations*, also called *variant equations*, and

3. *metadata equations*, the equations associated to dedicated unification algorithms.

Example 3.4. In NSPK, we consider the cancellation between encryption and decryption, which is defined by the variant equations as follows:

```
eq pk(A:Name,sk(A:Name,X:Msg)) = X:Msg [variant] .
eq sk(A:Name,pk(A:Name,X:Msg)) = X:Msg [variant] .
```

Maude-NPA uses *variant narrowing* [57] for unification of symbolic terms *modulo* the variant equations and the axioms specified for the algebraic properties of the protocol. In order for variant narrowing to provide a *finite* set of unifiers, an equational theory $T = (\Sigma, E \cup B)$ describing the algebraic property of cryptographic functions in a protocol is required to satisfy the following requirements:

1. The axioms B can be any combination of associativity (A), commutativity (C), and identity (U) axioms.

2. The equations E, when oriented as left-to-right rewrite rules, are sort-decreasing, confluent, terminating and coherent modulo B, and satisfy the *finite variant property.*

Confluence. The equations E are *confluent* modulo B if and only if for each term t in the theory, if $t \longrightarrow^{*}_{E/B} u$ and $t \longrightarrow^{*}_{E/B} v$, then there are terms w_1, w_2 such that: $u \longrightarrow^{*}_{E/B} w_1$, $v \longrightarrow^{*}_{E/B} w_2$, and $w_1 =_B w_2$.

Termination. The equations E are *terminating* modulo B if and only if all rewrite sequences terminate; that is, there is no infinite sequence of rewrites:

$$t_0 \rightarrow_{E/B} t_1 \rightarrow_{E/B} t_2 \ldots t_n \rightarrow_{E/B} t_{n+1} \ldots$$

Confluence, termination and sort-decreasingness can be checked in the Maude Formal Environment (MFE) [61].

Coherence. For Maude-NPA, coherence is mainly an issue for A, AC and ACU symbols. Given a set E of oriented equations, the following method [58] can be applied to make E coherent modulo such axioms. For any symbol f which is AC, and for any equation of the form $f(u,v) = w$ in E, add the equation $f(f(u,v),x) = f(w,x)$, where x is a new variable. If f is ACU with identity symbol e, we replace the equation $f(u,v) = w$ by the extended equation $f(f(u,v),x) = f(w,x)$. If f is associative only, add the equations $f(x, f(u,v)) = f(x,w)$, $f(f(u,v),y) = f(w,y)$, and $f(x, f(f(u,v),y)) = f(x, f(w,y))$. Note that some equations may already be coherent modulo AC or ACU, so the extra equations are not needed.

The Finite Variant Property. The essential condition for Maude-NPA is the *finite variant property* [62]. This condition is satisfied by many useful cryptographic theories. We have defined this property in the Preliminaries.

Note that whether or not an equational theory has the finite variant property is undecidable [63]. However, there is a semi-decision procedure [64] that works well in practice. This check can be performed in Maude by using the `get variants` command: if for each operator $f : S_1\ S_2\ \cdots S_n \to S$ in the theory, the set of variants obtained by the `get variants` command for the term $f(X_1 : S_1, \ldots, X_n : S_n)$ is finite, then the theory has the finite variant property. If any such term $f(X_1 : S_1, \ldots, X_n : S_n)$ has an infinite set of variants (which in practice will be suggested by Maude not terminating its process of generating variants), then the theory does *not* have the finite variant property.

3.3 BACKWARDS AND FORWARDS REACHABILITY ANALYSIS

Maude-NPA analyzes cryptographic protocols by performing backwards search from a user-specified final state while considering the algebraic propertied of the cryptographic primitives that are specified in the form of an equational theory. Since the number of states $T_{\Sigma_{SS_\mathcal{P}}/E_{SS_\mathcal{P}}}$ is in general infinite, rather than exploring concrete protocol states $[t]_{E_{SS_\mathcal{P}}} \in T_{\Sigma_{SS_\mathcal{P}}/E_{SS_\mathcal{P}}}$ Maude-NPA explores *symbolic strand state patterns* $[t(x_1, \ldots, x_n)]_{E_{SS_\mathcal{P}}} \in T_{\Sigma_{SS_\mathcal{P}}/E_{SS_\mathcal{P}}}(\mathcal{X})$ on the free $(\Sigma_{SS_\mathcal{P}}, E_{SS_\mathcal{P}})$-algebra over a set of variables $\mathcal{X}$. In this way, a state pattern $[t(x_1, \ldots, x_n)]_{E_{SS_\mathcal{P}}}$ represents not a single concrete state but a possibly infinite set of such states, namely, all the *instances* of the pattern $[t(x_1, \ldots, x_n)]_{E_{SS_\mathcal{P}}}$ where the variables $x_1, \ldots, x_n$ have been instantiated by concrete ground terms.

The backwards semantics of Maude-NPA is expressed in terms of the following *forward rewrite rules* $R_{B_\mathcal{P}}$ that describe how a protocol moves from one state to another via the intruder's interaction with it.

$$\{SS \;\&\; [L \mid M^-, L'] \;\&\; \{M{\in}\mathcal{I}, IK\}\} \to \{SS \;\&\; [L, M^- \mid L'] \;\&\; \{M{\in}\mathcal{I}, IK\}\} \tag{-}$$

$$\{SS \;\&\; [L \mid M^+, L'] \;\&\; \{IK\}\} \to \{SS \;\&\; [L, M^+ \mid L'] \;\&\; \{IK\}\} \tag{+}$$

$$\{SS \;\&\; [L \mid M^+, L'] \;\&\; \{M{\notin}\mathcal{I}, IK\}\} \to \{SS \;\&\; [L, M^+ \mid L'] \;\&\; \{M{\in}\mathcal{I}, IK\}\} \tag{++}$$

$$\forall\ [l_1, u^+, l_2] \in \mathcal{P} : \{SS \,\&\, [\, l_1 | \, u^+, l_2 \,] \,\&\, \{u{\notin}\mathcal{I}, IK\}\} \to \{SS \,\&\, \{u{\in}\mathcal{I}, IK\}\} \tag{\&}$$

where L and L' are variables denoting a list of strand messages, IK is a variable for a set of intruder facts ($m{\in}\mathcal{I}$ or $m{\notin}\mathcal{I}$), SS is a variable denoting a set of strands, and l_1, l_2 denote a list of strand messages. The set $R^{-1}_{B_{\mathcal{P}}}$ of *backwards* state transition rules is defined by reversing the direction of the above set of rules $\{(\text{-}), (+), (++)\} \cup (\&)$. The intruder knowledge acts as the only communication channel. The Rule (-) denotes receiving a message. Rules (+) and (++) describe sending a message. Extra strands necessary to reach an initial state are dynamically added by (&), a set of protocol-specific rewrite rules. In the backwards executions of $(\&), (++)$, $u{\notin}\mathcal{I}$ marks when the intruder learnt u.

In the backwards semantics, Maude-NPA attempts to find a path from an initial state to the attack pattern via backwards narrowing (narrowing using the rewrite rules with the orientation reversed). That is, a narrowing sequence from an initial state to an attack state is searched *in reverse* as a *backwards path* from the attack state to an initial state. Maude-NPA attempts to find paths until it can no longer form any backwards narrowing steps, at which point it terminates. If at that point it has not found an initial state, the attack pattern is shown to be *unreachable* modulo $E_{SS_{\mathcal{P}}}$. Note that Maude-NPA places no bound on the number of sessions, so reachability is undecidable in general. However, the tool makes use of a number of sound and complete state space reduction techniques that help to identify unreachable and redundant states, and thus make termination more likely.

The *forwards semantics* of Maude-NPA [60] is expressed in terms of *rewrite rules* $R_{F\mathcal{P}}$ that describe how protocol transitions from one state to another via the intruder's interaction with it. The rewrite rules $R_{F\mathcal{P}}$ are extracted from the strand specification $SSpec_{\mathcal{P}}$ of a protocol. A state consists of a multiset of partially executed strands and a set of terms in the intruders knowledge. Unlike the backwards semantics, only the part of the strand that has already executed is present in the state, and each such partial strand instantiates a prefix of a strand in $\mathcal{P}$. One progresses by either: (i) adding a positive message to an existing strand, denoting sending out a message, (ii) adding a negative message to an existing strand only if it is already present in the intruders knowledge, which denotes receiving a message or (iii) starting a new strand. For example, the intruder encryption capability denoted by the strand $[-(K), -(M), +(e(K, M)]$ has the following associated rewrite rules:

$$
\begin{aligned}
\{SS\,\&\,\{K{\in}\mathcal{I}, IK\}\,\} &\to \{SS\,\&\,[-(K)]\,\&\,\{K{\in}\mathcal{I}, IK\}\,\} \\
\{SS\,\&\,[-(K)]\,\&\,\{M{\in}\mathcal{I}, IK\}\,\} &\to \{SS\,\&\,[-(K), -(M)]\,\&\,\{M{\in}\mathcal{I}, IK\}\,\} \\
\{SS\,\&\,[-(K), -(M)]\,\&\,\{IK\}\,\} &\to \{SS\,\&\,[-(K), -(M), +(e(K, M)] \\
&\qquad \{e(K, M){\in}\mathcal{I}, IK\}\,\}
\end{aligned}
$$

where SS denotes a set of strands and IK a set of intruder knowledge facts. In this way, a protocol $\mathcal{P}$ defined in Maude-NPA by an equational theory $(\Sigma_{SS_{\mathcal{P}}}, E_{SS_{\mathcal{P}}})$ and strand specification $SSpec_{\mathcal{P}}$ also defines a corresponding rewrite theory $(\Sigma_{SS_{\mathcal{P}}}, E_{SS_{\mathcal{P}}}, R_{F\mathcal{P}})$ defining its rewriting logic forward semantics.

The *forwards execution* of a protocol begins with an "empty" initial state, that is, a state with the empty set of strands and intruder knowledge. The protocol is executed to determine whether or not an *attack state* can be reached, where an attack state is a ground instance of a state pattern specified by the user. One progresses by applying *rewrite rules* to states.

3.4 SEQUENTIAL PROTOCOL COMPOSITION

To support *sequential protocol composition*, strands can be extended with *synchronization messages* [48] of the form

$$\{Role_1 \to Role_2 \;;;\; mode \;;;\; w\}$$

where $Role_1, Role_2$ are constants of sort Role provided by the user, *mode* can be either `1-1` or `1-*` representing a one-to-one or one-to-many synchronization (whether an output message can synchronize with one or many input messages), and w is a term representing the information passed along in the synchronization messages. These synchronization messages are intended for reasoning about child protocols that use information received from the parent protocols, e.g., a session key establishment protocol that uses a master key received from a master key establishment protocol.

Intuitively, the sequential composition of two strands describes a situation in which one strand (the *child*), can only execute after another strand (*the parent*) has completed its execution. Each composition of two strands is obtained by *matching the output parameters of the parent strand with the input parameters of the child strand* in a user-specified way. Note that it may be possible for a single parent strand to have more than one child strand.

Definition 3.1 (Sequential Strand Composition[48])**.** Given two strands $(a) :: \overrightarrow{r_a} :: [\{\overrightarrow{I_a}\}, \overrightarrow{M_a}, \{\overrightarrow{O_a}\}]$ and $(b) :: \overrightarrow{r_b} :: [\{\overrightarrow{I_b}\}, \overrightarrow{M_b}, \{\overrightarrow{O_b}\}]$ that are properly renamed to avoid variable sharing, a sequential strand composition is a triple of the form $(a, b, MODE)$, where a and b denote the parent and child roles, respectively, and $MODE$ is either 1-1 or 1-*, indicating a one-to-one or one-to-many composition. This triple satisfies the following conditions for consistency:

1. both $\overrightarrow{O_a}$ and $\overrightarrow{I_b}$ have the same length, i.e. $\overrightarrow{O_a} = m_1, \ldots, m_n$ and $\overrightarrow{I_b} = m'_1, \ldots, m'_n$, and
2. there exists at least one substitution σ such that $\overrightarrow{O_a} =_{E_\mathcal{P}} \overrightarrow{I_b}\sigma$.

CHAPTER 4: THEORIES OF HOMOMORPHIC ENCRYPTION, UNIFICATION, AND THE FINITE VARIANT PROPERTY

4.1 INTRODUCTION

Recent advances in the automated analysis of cryptographic protocols [31, 16, 32, 66, 34, 35] have demonstrated that state exploration via unification modulo theories is often manageable and can make a substantial difference to the expressiveness of the system model and the accuracy and effectiveness of the formal analysis. In systems of this type, protocol execution paths are computed by unifying messages received with messages sent. Since equational properties are usually involved, the unification must be *modulo* the equational theory describing those properties. The technique of *variant unification*, first formalized as a general approach in [37], although used for specific theories much earlier than that (e.g. in [38, 31, 39]), is used by many cryptographic protocol analysis tools in one form or another, including ProVerif [38], OFMC [31], Maude-NPA [16] and Tamarin [35].

In order for variant unification to work, the equational theory $(\Sigma, E \uplus B)$ of interest must be decomposable into (Σ, B, R), where: (1) B is regular and has a finitary unification algorithm, (2) R are the rules obtained by orienting the equations E from left to right; they are confluent, terminating, and coherent modulo B (for simplicity, we will simply say from now on that the rules R are *convergent* modulo B) and (3) R has the *finite variant property* (FVP) [50] with respect to B; i.e., for each term t there is a finite set of most general pairs $\{(\sigma_1, t_1), \ldots, (\sigma_n, t_n)\}$ (called *variants*) of $\rightarrow_{R,B}$-normalized substitutions and terms such that $t\sigma_i$ normalizes to t_i modulo B. Fortunately, many cryptographic theories of interest can be decomposed in this way, with B being any combination of A, C, and U axioms.

However, there is one important family of theories that fails to have a decomposition that satisfies our requirements: H for a homomorphic property of the form $e(X*Y, K) = e(X, K)* e(Y, K)$ where $*$ may or may not have other properties. AGH (the case where $*$ is an Abelian group) is a property belonging to a number of different cryptographic algorithms, starting with RSA in the late 70's [51], and from early on it was realized to have a number of potential applications, including anonymous payment systems [67], computation on encrypted data [68], and voting [69].

The fact that the theory AGH by itself does not satisfy our needs does not mean that all is lost, however. But it does mean that we must investigate approximations that give us the

[1]This chapter is based on the paper [65], joint work with Santiago Escobar, Catherine A. Meadows, José Meseguer and Paliath Narendran.

ability to model a broad class of protocols while still satisfying the conditions necessary for variant based unification. First, we could use a dedicated unification algorithm for AGH, as we did in [70] for H; however, this is not satisfactory, see Section 4.2.2 for the reasons. Second, we can use an under-approximation of AGH, as we did in [70] with only one homomorphic encryption operator. Third, we can embed AGH or a subtheory in a richer theory; this may introduce false paths when verifying a protocol, but these can be discarded upon inspection. Fourth, we can use a combination of under and over-approximations in the same theory.

Such a strategy of course requires a fair amount of experimentation, both in checking for FVP and in using the theories in protocol analysis. Although necessary and sufficient conditions for FVP have been known for some time [50, 57], checks for these conditions are not straightforward to implement. Indeed, this turns out to be an undecidable problem [63]. However, a new semi-decision procedure for FVP has been developed which works well in practice as already explained in Section 3.2. The other properties of a decomposition (Σ, B, R) can be checked via the use of the Maude Formal Environment [61]. Thus it has become straightforward for us to verify that an equational theory has the properties we need.

The ability to generate all the variants of each term of the form $f(X_1, \ldots, X_n)$ for each $f \in \Sigma$ also gives us a measure of the overhead introduced by using this theory in a variant-based cryptographic protocol analysis tool: the larger the number of variants produced, the larger the number of variants of a state generated by a unification-based protocol analysis tool, and thus the larger the number of states generated during a search, and the longer the time required to generate them. We refer to this, admittedly quite rough, measure as the *variant complexity* of the equational theory.

All these capabilities taken together greatly facilitate the definition and verification of theories with the finite variant property, and allow us to refine our strategy for generating theories with FVP decompositions. We start with a non-FVP theory such as homomorphic encryption, or a theory with an unacceptably large variant complexity. First, we add equations that we conjecture achieve a finite number of variants. Then we check for convergence modulo B using Maude tools. If convergence is not satisfied we add additional equations (possibly suggested by Maude Church-Rosser checker's output) and try again. Once we are satisfied that the theory is convergent modulo B, we then use Maude to check for FVP and compute its variant complexity. We can then test the theory using Maude-NPA. If the performance is unacceptable we go through the process again, introducing under or over approximations to achieve a lower variant complexity. If the performance is acceptable we may again try to tweak the theory, but in this case to increase its faithfulness to the theory of interest rather than reducing variant complexity. We note that use of under approximation means that Maude-NPA may miss attacks (that is, it affects completeness), while over approximation means that Maude-NPA may find spurious attacks (that is, it affects soundness). Thus, ultimately we would expect to use several different theories, some under and some over approximations to approximate a theory of interest.

4.1.1 The contributions

The contributions of this chapter are thus fourfold. First, we develop a *general strategy* for *approximating* theories without FVP or with high variant complexity by using a combination of under-approximation (eliminating equations from a theory or substituting them by weaker ones) and over-approximation (adding additional function symbols and equations), and then computing the variant complexity. The use of Maude to verify convergence modulo B and to compute the variant complexity greatly facilitates the experimentation needed to carry this strategy out.

Secondly, we apply the strategy to develop a *hierarchy* of theories approximating homomorphic encryption that are verified to have the finite variant property.

Thirdly, as a result of proving the finite variant property for the theories in the hierarchy, we automatically obtain unification algorithms for these theories, which to the best of our knowledge are new unification results except for the already known H and AGH cases.

Fourthly, we have performed a careful experimental evaluation of the performance tradeoff between the faithfulness with which the theory models cryptographic operations and the number of variants, giving us a better understanding of how variant complexity affects performance of automated protocol analysis tools. We used the most promising theories from the point of view of variant complexity and reasonable expressiveness to specify and analyze different versions of two protocols using the unification-based cryptographic protocol analysis tool Maude-NPA [16]. In one of the protocols, two principals A and B want to learn the result of performing the operation $*$ on their respective secret information D_A and D_B without revealing the information to each other, or $D_A * D_B$ to anyone else. Each one encrypts its information using a homomorphic encryption algorithm and sends it to a server, who performs the operation $*$ on the encrypted data and sends it to A and B who decrypt the result to obtain $D_A * D_B$. In [70] it was shown that the protocol was subject to an authentication attack if a principal was unable to tell whether it had received valid data or nonsense. In that paper * was a free operator, so the theory E had a trivial decomposition, in which $R = \varnothing$ and $B = E$. We show in the present work that in the case where $*$ is a group operator it is also subject to a secrecy attack: A can simply compute $D_A^{-1} * (D_A * D_B)$ to obtain D_B. In the other protocol, two principals A and B intend to agree on a secret while using a hash to protect the integrity of the messages. But the hash function is flawed in the

$$\begin{tikzcd}
XORH^{\infty} & AGH^{\infty} \arrow[l, two heads] & & \\
2XORH^{48} \arrow[u, two heads] \arrow[rrr, bend right] & 2AGH^{2276} \arrow[l, two heads] \arrow[u, two heads] \arrow[r] & 2AGHD^{2279} \arrow[r, two heads] & 2XORHD^{51} \\
PXorAAH^{22}_{\&} \arrow[u, dotted] \arrow[rrr, bend right] & & & PXorAAHD^{28}_{\&} \arrow[u, dotted] \\
PXorH^{12}_{\&} \arrow[u] \arrow[rrr, bend right] & & & PXorHD^{18}_{\&} \arrow[u] \\
& APGAAH^{32}_{\&} \arrow[uul] \arrow[uuu, dotted] \arrow[r] & APGAAHD^{38}_{\&} \arrow[uur] \arrow[uuu, dotted] & \\
APGH^{20}_{\&} \arrow[uu, two heads] \arrow[ur] \arrow[rrr, bend right] & & & APGHD^{26}_{\&} \arrow[uu, two heads] \arrow[ul] \\
& PGAAH^{22}_{\&} \arrow[uu] \arrow[r] & PGAAHD^{38}_{\&} \arrow[uu] & \\
PGH^{20}_{\&} \arrow[uu] \arrow[ur] \arrow[rrr] & & & PGHD^{26}_{\&} \arrow[uu] \arrow[ul] \\
H^{8}_{\&} \arrow[u] \arrow[rrr] & & & HD^{13}_{\&} \arrow[u] \\
H^{\infty} \arrow[u] \arrow[rrr] & & & HD^{\infty} \arrow[u] \\
3H^{4} \arrow[u] \arrow[rrr] & & & 3HD^{29} \arrow[u]
\end{tikzcd}$$

Figure 4.1: Relations between the theories discussed in this work

sense that it is homomorphic. As a result, there is an authentication attack in which the responder B may think that he has shared a secret with A when A actually didn't participant in the protocol. This attack was also found in [70]. We were able to found this attack using theories presented in this work. We also present a fix that can prevent the attack when $*$ is a free operator, which is the case that is shown in [70], but cannot prevent the attack when $*$ has a richer axioms. To our knowledge, this is the first successful use of a cryptographic protocol analysis tool to analyze homomophic encryption over theories obeying nontrivial equational theories.

The theories for which we have found variant-based unification algorithms using our strategy are summarized in Figure 4.1. We explain the notation in Figure 4.1 as follows. A full arrow denotes theory inclusion, a full arrow with two heads denotes a theory quotient, and a dotted arrow denotes a generalization, which is not a theory inclusion, but where more equations and/or axioms have been added. All theories are described in Section 4.3 and involve the function $e(X, K)$ where e is an encryption operator, X is a term of sort Message, and K is a term of sort Key or Keys depending on the theory, where such sorts are always subsorts of Message. H denotes the homomorphic equation $e(X * Y, K) = e(X, K) * e(Y, K)$. kH denotes the *bounded homomorphism theory*, in which sorts are used to restrict the num-

Theories	**Over-Approximation**
kH	-
$H_{\&}$	Over approximation of H via use of multiset of keys.
$PGH_{\&}, PGAAH_{\&}$, $APGH_{\&}, APGAAH_{\&}$	Over approximation of H via use of multiset of keys.
$PXorH_{\&}$, $PXorAAH_{\&}$	Over approximation of H via use of multiset of keys. Over approximation of Group Axioms.*
$2AGH$	-
$2XORH$	Over approximation of Group Axioms.*

* Note that this is not an over-approximation when the encryption function being modeled is indeed a homomorphism over Xor.

Table 4.1: Summary of Over approximations w.r.t. AGH

Theories	**Under-Approximation**
kH	Under approximation of H via bound on message length. No group axioms.
$H_{\&}$	No group axioms.
$PGH_{\&}$ $PGAAH_{\&}$ $APGH_{\&}$ $APGAAH_{\&}$ $PXorH_{\&}$ $PXorAAH_{\&}$	Under approximation of associativity axiom.
$2AGH$	Under approximation of H via use of two groups instead of one.* Under approximation of encryption via bound on number of keys.
$2XORH$	Under approximation of H via use of two groups instead of one.* Under approximation of encryption via bound on number of keys.

* Note that this is not an under-approximation when the encryption function being modeled is indeed a homomorphism from one group to another.

Table 4.2: Summary of Under approximations w.r.t. AGH

ber of variants of $h(x)$ that can be computed. The symbol & represents the addition of an AC binary function symbol & on terms of a new sort Keys; it can be thought of as a multiset union operator. This introduces an over approximation if the order of applications of encryption to a message matters. PG adds an inverse $(\_)^{-1}$, a constant 1 and equations making $*$ a *pre-group operator* (unit, inverses, but no associativity, also known as a *loop*, see [71]), while APG makes $*$ an *Abelian pre-group operator* (commutativity, unit, inverses, but no associativity). PXor makes $*$ a *pre-Exclusive-or operator* (Exclusive-or without associativity). AA denotes an under approximation of associativity. D denotes the decryption equation $d(e(X, K), K) = X$; this says that the result of decrypting an encrypted message is the original message. 2AGH denotes homomorphic encryption that maps one Abelian group to another: $e(X *_a Y) = e(X) *_b e(Y)$. The operators $*_a$ and $*_b$ are *Abelian group operators* (commutativity, unit, inverses, associativity). Notice that, there is no key explicitly defined in 2AGH, since encryption with a specific key is implicitly captured by the definition of the encryption operator e. 2XORH denotes homomorphic encryption over two Xor operators, which is an over approximation of 2AGH. In all cases the axioms B are either $B = \varnothing$ or the union of all the equations defining C and AC properties. We also note that in many cases we completed the theory to ensure convergence; these are described in detail in Section 4.3. The superscript number of each theory denotes the "*variant complexity*" and denotes the sum of the number of variants obtained for each function symbol in the theory (excluding constants). If the superscript is ∞, this means that the theory doesn't have the FVP. The details of over approximation and under approximation w.r.t. AGH of all theories are given in Tables 4.1 and 4.2 respectively. The theories with a decryption operator are omitted from Tables 4.1 and 4.2 .

The rest of this chapter is organized as follows. In Section 4.2 we give the motivation of FVP in terms of cryptographic protocol analysis. In addition we describe related work in unification and apply it to show that none of the possible decompositions of AGH satisfy the necessary conditions for variant unification. In Section 4.3 we present the various homomorphic theories we investigated and their properties. In Section 4.4 we present the results of performing experiments on several representative theories, using Maude-NPA to analyze protocols specified using these theories. We conclude in Section 4.5.

4.2 MOTIVATION AND RELATED WORK

In this section we discuss the related work that precedes and motivates the work presented in this chapter. This is divided into two parts. The first motivates our interest in FVP in terms of its application to cryptographic protocol analysis. The second gives a brief history of work on unification modulo one-sided distributivity that applies to homomorphic encryption and uses these results to show that no decomposition of AGH satisfies all the conditions necessary for the finite variant property, and thus demonstrates the need for other solutions such as theory approximations.

4.2.1 Motivation

Unification-based cryptographic protocol analysis tools are used to analyze cryptographic protocols in which an attacker interacting with the protocol may cause security properties to be violated. Actions of principals are modeled symbolically using logical variables, and paths through protocols are computed by unifying messages expected by a principal with messages sent by a principal, often modulo some equational theory that describes the properties of the crypto algorithms used.

Any unification technique used in cryptographic protocol analysis must satisfy two properties. First of all, it must behave well with respect to composition, especially of disjoint theories, since cryptographic protocols often combine different algorithms described by different theories. Although methods for combining unification algorithms of disjoint theories are well-known [72, 73], the solution in the general case can be highly nondeterministic and inefficient, so more efficient special algorithms are desirable.

The second property that must be satisfied is a little more subtle, and has to do with the fact that many of the state space reduction techniques used require that terms in the state be in some kind of normal form with respect to the theory $E \uplus B$ used. Generally this is expressed by writing $E \uplus B$ as a decomposition (R, B) where B is regular and has a finitary unification algorithm, and R are the rules obtained by orienting E, which are convergent modulo B. This ensures enough stability in normal form representations of terms so that syntactic state space reduction techniques can be applied.

Both the first and second desiderata of unification-based cryptographic can be achieved, if the decomposition (R, B) has the finite variant property, via variant unification as described in Chapter 2. Since the first step of variant unification requires the computation of all the irreducible variants of each side of the unification problem, and a solution is discarded if a solution makes either side reducible, variant unification guarantees the irreducibility constraint required by state space reduction techniques. Moreover, variant unification behaves well under composition, at least in the area of cryptographic protocol analysis. First of all, the axioms B are relatively few and well-understood. Moreover, if the combination of the two theories also has a finite variant decomposition, then the same finite variant algorithm can be applied as well.

Not surprisingly many tools have followed approaches similar, if not identical, to variant unification. Both Maude-NPA [16] and Tamarin [35] use variant-based unification explicitly. Indeed support for variant-based unification is built into Maude 2.7. Moreover, other tools have used approaches that have many features in common with variant-based unification. For example, ProVerif [38, 74] and OFMC [31, 75] both compute the variants of protocol rules, modulo the free theory for ProVerif, and modulo the free theory or AC for OFMC. This has the effect of computing the variants of both sides of the unification problem. More recently, variants have been applied to expanding the capacity of ProVerif to deal with AC theories. Thus, in [66] Küsters and Truderung implement a special case of the exclusive-or theory in the ProVerif tool by expressing it as a rewrite theory with the finite variant property with respect to the free theory and computing variants that are unified syntactically. This requires some restrictions on the syntax of the protocol, however. Similar approaches have been applied by Küsters and Truderung for modular exponentiation [32], and Arapinis et al. [34] for commuting encryption and AC theories.

Variant-based unification does have some drawbacks, however. First of all, it can be inefficient for theories of high variant complexity. This can be mitigated by the use of *asymmetric unification* [76], in which only the variants of the right-hand side of a unification problems are computed, and irreducibility constraints are also enforced only on the right-hand sides. This requires the use of specialized asymmetric unification algorithms, so combination is no longer as straightforward, but it is still possible to apply the state space reduction techniques, and efficiency gains, as shown in [76], can be dramatic.

A more serious problem arises when a theory of interest fails to have an FVP decomposition at all. Fortunately, most theories of interest to cryptographic protocol analysis are FVP with respect to a decomposition in which B is either the empty theory or any combination of A, C, and U axioms. However, there is one notable exception, the theory of encryption homomorphic over another operator, that is $e(X * Y, K) = e(X, K) * e(Y, K)$ where $*$ is an operator that may have some other equational properties, shown not to satisfy FVP when the homomorphic equation is in R by Comon and Delaune in [50]. Comon and Delaune consider the case in which e has only one argument and $*$ is exclusive-or, but their case can be considered as corresponding to a degenerate case of e with two arguments, in which only one key is used. Moreover, their counterexamples apply to any sub theory of the theory they use for $*$, including the Abelian group theory, AC, and the free theory.

Comon and Delaune prove their result by producing counterexamples to a property that they show to be equivalent to FVP. However, it is also easy to produce direct counter examples. Assuming that the distributive equation is oriented as the rewrite rule $e(X * Y) \to e(X) * e(Y)$, then the irreducible variants of $e(Z)$ are $[(e(Z), \iota), (e(Z_1) * e(Z_2), \{Z \mapsto Z_1 * Z_2\}), \ldots]$. If the equation is oriented as $e(X) * e(Y) \to e(X * Y)$, then the variants of $e(Z * W)$ are $[(e(Z * W), \iota), (e(e(Z_1 * W_1)), \{Z \mapsto e(Z_1), W \mapsto e(W_1)\}), \ldots]$.

4.2.2 Related Work in Equational Unification and its Application to Decompositions of AGH

Equational theories that include a homomorphic property appear in many applications, and thus there is a long history of research on unification in this area. As in Comon and Delaune, the homomorphic operator under consideration generally has only one argument. However, as shown before, this theory can be considered as applying to a degenerate case of homomorphic encryption in which only one key is used and so is still relevant.

The earliest work on homomorphic theories is by Tiden and Arnborg [77] who gave a unification algorithm modulo the theory of one-sided distributivity:[2] $x * (y + z) = (x * y) + (x * z)$. The complexity of their algorithm is exponential. A polynomial-time algorithm for unifiability (existence of a unifier) modulo this theory was developed by Marshall [78, 79]. An alternative unification algorithm using a very different approach was developed by Hai Lin in his dissertation [80, 81]. This algorithm for the equational property $e(X * Y, K) = e(X, K) * e(Y, K)$ in which $*$ is a free operator was implemented in Maude-NPA and several protocols were tested [70].

Theories of homomorphisms of the form $e(x * y) = e(x) * e(y)$, where the $*$ operator has additional properties, were first considered by Nutt [82] and also by Baader [83, 84]. The unification problem is decidable when $*$ is an Abelian group [84] and undecidable when $*$ only has the associative and commutative properties [85]. The decidability results were extended to one-sided distributivity in [86, 87]. Liu in [88] gives a dedicated algorithm for the case in which $*$ is exclusive-or.

We make use of all these previous results to study the decompositions of AGH and show that none of them meet our needs. We give the equations for AGH below (ignoring equations needed to complete the theory for coherence):

$$
\begin{aligned}
(x * y) * z &= x * (y * z) && (4.1)\\
x * y &= y * x && (4.2)\\
x * 1 &= x && (4.3)\\
x * (x)^{-1} &= 1 && (4.4)\\
e(1) &= 1 && (4.5)\\
e(x * y) &= e(x) * e(y) && (4.6)
\end{aligned}
$$

We note that although unification modulo AGH itself is finitary, this does not help us for variant-based unification, since AGH is not regular. Indeed, because of the need for regularity of B (without it a convergent decomposition becomes practically impossible), Eq. 4.4 must be in R for any decomposition (Σ, B, R). Moreover, because commutativity can not be written as a rewrite rule, and because unification modulo associativity without commutativity is not finitary, Eqs. 4.1 and 4.2 must be in B.

We also know that if Eq. 4.6 is in R, then the decomposition will fail to have the finite variant property, as we have noted earlier. Thus the only choices left for B are $B_1 =$

[2]Note: in this theory the operation $+$ corresponds to our use of $*$ and $*$ corresponds to our use of e.

$\{4.1, 4.2, 4.3, 4.5, 4.6\}$, $B_2 = \{4.1, 4.2, 4.6\}$, $B_3 = \{4.1, 4.2, 4.5, 4.6\}$, and $B_4 = \{4.1, 4.2, 4.3, 4.6\}$. Unification for B_1 and B_2 is known to be undecidable [85]. The theories B_3 and B_4 have not been as well studied, but we note that the problems $e(x) =^? x$ and $x{*}x =^? x$ have nonfinitary solutions for B_3 and B_4 respectively. For the former, the set of mgu's is $\{x \to 1, x \to 1{*}1, \ldots\}$; for the latter it is $\{x \to e(1), x \to e(e(1)), \ldots\}$.

4.3 FVP THEORIES OF HOMOMORPHIC ENCRYPTION

In this section we present all these theories mentioned in Figure 4.1. All the theories are based on the following homomorphic theory.

Definition 4.1. The homomorphic theory is defined as $T_H = (\Sigma_H, \varnothing, R_H)$. The signature Σ_H is defined by sorts $\{\mathsf{Key}, \mathsf{Msg}\}$ with the subsort relation $\mathsf{Key} < \mathsf{Msg}$, the binary operator $\_ * \_ : \mathsf{Msg} \times \mathsf{Msg} \to \mathsf{Msg}$, the encryption operator $e : \mathsf{Msg} \times \mathsf{Key} \to \mathsf{Msg}$. and possible symbols for constructing keys. The set R_H of rules contains only the following rule, which is a variation of Equation (4.6), where X, Y are of sort Msg and K is of sort Key:

$$e(X * Y, K) \;\; \to \;\; e(X, K) * e(Y, K) \tag{4.7}$$

The bounded homomorphism theories kH and kHD are presented in Section 4.3.1, in which the number of message concatenations by $*$ are bounded taking advantage of sorts. The theories $\mathrm{H}_{\&}$ and $\mathrm{HD}_{\&}$ for homomorphic encryption with a multiset of keys are presented in Section 4.3.2.1, in which an AC binary function symbol & on keys is introduced, and the homomorphic encryption thus uses a multiset of keys instead of a key. The theories $\mathrm{PGH}_{\&}$, $\mathrm{PGHD}_{\&}$ for homomorphic encryption over a Pre-Group are presented in Section 4.3.2.2. Besides the AC binary function symbol & on keys, these theories also feature an inverse operation $(\_)^{-1}$, a unit constant 1 and equations making the binary operator $*$ a *pre-group operator* (unit, inverses, but no associativity). The theories are extended with associativity approximation in $\mathrm{PGAAH}_{\&}$ and $\mathrm{PGAAHD}_{\&}$. Theories $\mathrm{APGH}_{\&}$, $\mathrm{APGHD}_{\&}$ denoting homomorphic encryption over an Abelian Pre-Group are presented in Section 4.3.2.3. Based on homomorphic encryption over a Pre-Group, the operator $*$ is further interpreted as an *Abelian pre-group operator* (commutativity, unit, inverses, but no associativity). The theories $\mathrm{APGH}_{\&}$, $\mathrm{APGHD}_{\&}$ are extended with associativity approximation in $\mathrm{APGAAH}_{\&}$ and $\mathrm{APGAAHD}_{\&}$ respectively. Theories $\mathrm{PXorH}_{\&}$, $\mathrm{PXorHD}_{\&}$ denoting homomorphic encryption over a Pre-Xor operator are presented in Section 4.3.2.4. Based on homomorphic encryption over an Abelian Pre-Group, the operator $*$ is now interpreted as a *pre-Xor operator* (exclusive-or without associativity). The theories $\mathrm{PXorAAH}_{\&}$ and $\mathrm{PXorAAHD}_{\&}$ extend

PXorH$_{\&}$ and PXorHD$_{\&}$, respectively with associativity approximations. Theories 2AGH and 2AGHD are presented in Section 4.3.3.1, and 2XORH in Section 4.3.3.2. 2AGH denotes homomorphic encryption that maps one Abelian group to another. 2XORH denotes homomorphic encryption over two Xor operators, which is an over approximation of 2AGH. We have proved that these theories all have the FVP. Therefore, they can be often combined with other theories with disjoint symbols, which is another great benefit of our approach. As an example, we presented in each Section the theories that can be combined with T_{PK}, a theory denoting another encryption which is not homomorphic. The combined theories kHPK, kHDPK, H$_{\&}$PK, HD$_{\&}$PK, ... all have the FVP.

We also note that, any FVP decomposition (Σ, B, R) has a fixed minimal set of variants determined by the symbols in Σ. We use this to introduce the notion of *variant complexity*, which can be used as a rough metric of the complexity inherent in using a particular theory.

Definition 4.2. For $T = (\Sigma, B, R)$, the decomposition of an FVP theory, the *variant complexity* is defined as the sum $vc(T) = \sum_{f \in \Sigma} v(f)$, where $v(f)$ is the cardinality of the complete and finite set of variants of $f(v_1, \ldots, v_n)$ where $v_1, \ldots, v_n$ are distinct variables of the biggest possible sorts making $f(v_1, \ldots, v_n)$ well typed.

Example 4.1. Consider the theory with sorts $\{Msg, Key\}$, operators: $e : \mathsf{Msg} \times \mathsf{Key} \rightarrow \mathsf{Msg}, d : \mathsf{Msg} \times \mathsf{Key} \rightarrow \mathsf{Msg}$ and rule $d(e(X, K), K) = X$, where X is of sort Msg, and K is of sort Key. The variant complexity of this theory is 3, since the term $e(X, K)$ has 1 variant: $(e(X, K), \iota)$ and the term $d(X, K)$ has 2 variants: $(d(X, K), \iota)$ and $(X_1, \{X \rightarrow e(X_1, K_1), K \rightarrow K_1\})$, with X, X_1 variables of sort Msg and K, K_1 variables of sort Key.

4.3.1 Theory of Bounded Homomorphism

In this section, we first introduce equational theories for bounded homomorphism. The length of possible message concatenations is bounded by introducing a new sort $\mathsf{SingleMsg}$, which is a subsort of Msg. The theories T_{kH} and T_{kHD} thus obtained are convergent and FVP. We also illustrate in this section that T_{kH} and T_{kHD} can be combined with some other commonly used theories while remaining FVP.

Definition 4.3 (T_{kH})**.** The bounded homomorphic theory is defined as $T_{kH} = (\Sigma_{kH}, \varnothing, \widehat{R_H}^{k})$ for k the bound. The signature Σ_{kH} is defined by adding sort $\mathsf{SingleMsg}$ to the previous sorts Msg and Key with the subsort relation $\mathsf{Key} < \mathsf{SingleMsg} < \mathsf{Msg}$. The signature Σ_{kH} contains an overloaded definition of the encryption operator, i.e.,

$$e : \mathsf{SingleMsg} \times \mathsf{Key} \rightarrow \mathsf{SingleMsg} \quad \textit{and} \quad e : \mathsf{Msg} \times \mathsf{Key} \rightarrow \mathsf{Msg}.$$

The equation (4.7) of R_H is replaced in $\widehat{R_H}^k$ by the following rules: for all $1 < j \leqslant k$, add $e(S_1 * \cdots * S_j, K) \to e(S_1, K) * \cdots * e(S_j, K)$ to $\widehat{R_H}^k$, where the operator $\_*\_$ denotes message concatenation, K is a variable of sort Key, and $S_1, \ldots, S_k$ are variables of sort $\mathsf{SingleMsg}$. For $k = 3$, the set of rules in $\widehat{R_H}^3$ is:

$$e(S_1 * S_2, K) \to e(S_1, K) * e(S_2, K)$$
$$e(S_1 * S_2 * S_3, K) \to e(S_1, K) * e(S_2, K) * e(S_3, K)$$

The theory T_{3H} is convergent and have the FVP. Indeed, $vc(T_{3H}) = 4$, since $v(*) = 1, v(e) = 3$. To show that the theory T_{3H} has FVP, we verified the confluence of T_{3H} using Maude CRC tool, terminating using Maude MTT tool and the variant complexity using Maude by variant generation.

Note that in order to derive a bounded homomorphic theory we need to take advantage of the order-sorted technique by declaring a subsort $\mathsf{SingleMsg}$, and the homomorphic encryption operator e is overloaded. Also note that the bound k is usually chosen depending on different applications. Although the bounded homomorphic theory is an underapproximation of the theory T_H, in concrete applications, with a properly chosen bound k, manual reasoning based on the results we get from using the bounded theory T_{kH} could be used to overcome the restriction of underapproximation.

Example 4.2. Consider the following unification problem:

$$X{:}\mathsf{Msg} * Y{:}\mathsf{Msg} =^?$$
$$e(data(A{:}\mathsf{Name}, r{:}\mathsf{Fresh}) * data(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh}), K{:}\mathsf{Key})$$

in the theory T_{3H}, where $data : \mathsf{Name} \times \mathsf{Fresh} \to \mathsf{Data}$ is an additional operator that generates principal's secrets, and we assume that a secret is a single message, i.e., the subsort relation $\mathsf{Data} < \mathsf{SingleMsg}$. We get the following unifier by variant based unification:

$$\begin{array}{ll}
\{\ X{:}\mathsf{Msg} \mapsto e(data(A1{:}\mathsf{Name}, r1{:}\mathsf{Fresh}), K1{:}\mathsf{Key}), & \\
\quad Y{:}\mathsf{Msg} \mapsto e(data(B1{:}\mathsf{Name}, r2{:}\mathsf{Fresh}), K1{:}\mathsf{Key}), & \\
\quad A{:}\mathsf{Name} \mapsto A1{:}\mathsf{Name}, & r{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh}, \\
\quad B{:}\mathsf{Name} \mapsto B1{:}\mathsf{Name}, & r'{:}\mathsf{Fresh} \mapsto r2{:}\mathsf{Fresh}, \\
\quad K : \mathsf{Key} \mapsto K1{:}\mathsf{Key}\ \ \} &
\end{array}$$

Decryption operators are usually used in protocols to denote principals' ability to extract information out of certain encrypted messages. We can extend the theory T_{kH} by adding a decryption operator d and the equations capturing the encryption/decryption cancellation relation between d and the homomorphic encryption operator e. We refer to the resulted theory as T_{kHD}.

Definition 4.4 (T_{kHD})**.** The bounded homomorphic theory with decryption is defined as $T_{kHD} = (\Sigma_H \cup \Sigma_{kH} \cup \Sigma_{Dec}, \varnothing, \widehat{R_H}^{k} \cup R_{Dec} \cup R_{kH\text{-}Dec})$. The signature Σ_{Dec} contains the overloaded decryption operator d with typings:

$$d : \mathsf{Msg} \times \mathsf{Key} \rightarrow \mathsf{Msg}, \qquad d : \mathsf{SingleMsg} \times \mathsf{Key} \rightarrow \mathsf{SingleMsg}.$$

The encryption/decryption cancellation properties are captured by the rules R_{Dec}, where X is of sort Msg and K is of sort Key:

$$e(d(X,K),K) \rightarrow X \qquad\qquad d(e(X,K),K) \rightarrow X$$

The rules $R_{kH\text{-}Dec}$ capture the homomorphic property of the decryption operation, and are needed for the theory to be convergent. For all $1 < j \leqslant k$, we add $d(S_1 * \cdots * S_j, K) \rightarrow d(S_1,K) * \cdots * d(S_j,K)$ to $R_{kH\text{-}Dec}$, where the operator $\_ * \_$ denotes message concatenation, K is a variable of sort Key, and $S_1, \ldots, S_k$ are variables of sort SingleMsg. The rules for bound $k = 3$ are as follows:

$$d(S_1 * S_2, K) \rightarrow d(S_1,K) * d(S_2,K)$$
$$d(S_1 * S_2 * S_3, K) \rightarrow d(S_1,K) * d(S_2,K) * d(S_3,K)$$

We can follow the same approach as in theory T_{kH} to compute the variant complexity and verify that the theory T_{kHD} has the finite variant property. The variant complexity of the theory T_{3HD} is 29.

Example 4.3. To illustrate the decryption operation, we consider the following unification problem:

$$X{:}\mathsf{Msg} * Y{:}\mathsf{Msg} =^{?}$$
$$d(e(data(A{:}\mathsf{Name}, r{:}\mathsf{Fresh}) * data(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh}), K{:}\mathsf{Key}), K{:}\mathsf{Key})$$

in the theory T_{3HD}, where $data : \mathsf{Name} \times \mathsf{Fresh} \rightarrow \mathsf{Data}$ is an additional operator that generates principal's secrets, and we assume the subsort relation $\mathsf{Data} < \mathsf{SingleMsg}$. We get the following unifier by variant based unification:

$$\begin{array}{ll} \{\ X{:}\mathsf{Msg} \mapsto data(A1{:}\mathsf{Name}, r1{:}\mathsf{Fresh}), & Y{:}\mathsf{Msg} \mapsto data(B1{:}\mathsf{Name}, r2{:}\mathsf{Fresh}), \\ A{:}\mathsf{Name} \mapsto A1{:}\mathsf{Name}, & r{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh}, \\ B{:}\mathsf{Name} \mapsto B1{:}\mathsf{Name}, & r'{:}\mathsf{Fresh} \mapsto r2{:}\mathsf{Fresh}, \\ K : \mathsf{Key} \mapsto K1{:}\mathsf{Key}\ \ \} & \end{array}$$

The theory T_{kH} and T_{kHD} can be extended by adding axioms to the free operator $*$, while remaining FVP.

Definition 4.5 ($T_{kH_{AC}}$)**.** The theory $T_{kH_{AC}}$ extends T_{kH} by adding associativity and commutativity axioms to $*$, i.e., $T_{kH_{AC}} = (\Sigma_{kH}, B_{AC}, \widehat{R_H}^{k})$ for k the bound, where the set of axioms B_{AC} includes the associativity and commutativity (AC) axioms of the binary operator $*$.

The theory $T_{kHD_{AC}}$ extends the theory T_{kHD} in the same way.

4.3.1.1 Combining T_{kH} with theories having disjoint symbols.

Cryptographic protocols usually use more than one kind of encryption. Since the theory T_{kH} is FVP, the general variant-based unification algorithm can be applied, which can easily be combined with other theories with disjoint symbols. Consider the theory T_{PK} defined below, which has the FVP.

Definition 4.6 (T_{PK})**.** The theory T_{PK} is defined as: $T_{PK} = (\Sigma_{PK}, \varnothing, R_{PK})$, where Σ_{PK} is defined by two sorts $\{\mathsf{Name}, \mathsf{Msg}\}$ with the subsort relation $\mathsf{Name} < \mathsf{Msg}$, together with operators $pk : \mathsf{Msg} \times \mathsf{Name} \to \mathsf{Msg}$ and $sk : \mathsf{Msg} \times \mathsf{Name} \to \mathsf{Msg}$, where pk and sk represent encrypt a message with public key and private key respectively. The encryption/decryption cancellation property is described in R_{PK} as follows:

$$sk(pk(X, A), A) = X \qquad pk(sk(X, A), A) = X$$

where X is a variable of sort Msg and A is a variable of sort Name.

Combining the theories T_{kH} and T_{PK}, we obtain the theory $T_{kHPK} = T_{kH} \cup T_{PK}$, which has the FVP. The same proof approach as in theory T_{kH} can be applied. Also, by combing the theories T_{kHD} and T_{PK}, we get the theory $T_{kHDPK} = T_{PK} \cup T_{kHD}$, which also has the FVP.

Example 4.4. Consider the same unification problem as in Example 4.2

$$\begin{array}{l} X{:}\mathsf{Msg} * Y{:}\mathsf{Msg} =^{?} \\ e(data(A{:}\mathsf{Name}, r'{:}\mathsf{Fresh}) * data(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh}), K{:}\mathsf{Key}) \end{array}$$

in the theory T_{3HPK} we get the same unifier as in Example 4.2 using variant based unification.

As another example, consider the theory $T_{Xor} = (\Sigma_{Xor}, B_{Xor}, R_{Xor})$, where Σ_{Xor} is defined by sorts $\{\mathsf{Nonce}, \mathsf{NSet}\}$ with the subsort relation $\mathsf{Nonce} < \mathsf{NSet}$, together with the exclusive-or operator $\oplus : \mathsf{NSet} \times \mathsf{NSet} \to \mathsf{NSet}$ and the identity $0 :\to \mathsf{NSet}$. The axioms B_{Xor} are associativity and commutativity axioms for $\oplus$. The exclusive-or rules R_{Xor} are as follows:

$$N \oplus N = 0 \qquad N \oplus 0 = N \qquad N \oplus N \oplus M = M$$

where N and M are variables of sort NSet. The theory T_{Xor} has the finite variant property. Therefore, by combining the theory T_{kH} with T_{Xor}, we obtain the theory $T_{kH\text{-}Xor} = T_{kH} \cup T_{Xor}$, which is also FVP.

4.3.2 Homomorphic Theories: Theory of Homomorphic Encryption with a Multiset of Keys

Since neither AGH nor H have the FVP, we have extended H with a new presentation of $e(M,\ K)$ built on top of a new symbol $\_\&\_$, which is associative and commutative (AC), and keeps all the keys used for homomorphic encryption in a multiset. Since nested encryptions with the same subset of keys can be flattened using AC, together with orienting the equation 4.7 as $e(X, K) * e(Y, K) \to e(X * Y, K)$, this theory has the FVP. But one side effect is that the order of applications of homomorphic encryption to a message becomes immaterial. This is not a standard property of encryption, homomorphic or not, so in most cases, the multiset AC axioms are over approximations that are used to get the finite variant property. Soundness (i.e., any attacks found are real attacks) is lost since the over approximation may introduce spurious attacks, but they can be discarded upon inspection. Also, we note that there are a few cases, such as Distributed ElGamal [89], in which encryption does satisfy this multiset condition, so this may be useful for reasoning about an additional class of crypto-algorithms as well.

In this section, we first introduce the theory $T_{H_\&}$, which consists of homomorphic encryption over a free operator and uses a multiset of keys. This theory is convergent and has the FVP. Indeed, $vc(T_{H_\&}) = 8$.

The theory $T_{H_\&}$ is then enriched with group operators and axioms. Since the FVP is lost again when adding the full group axioms due to the fact that confluence requires adding an infinite number of extra rules, associativity is approximated by a sub-theory. We call a group without associativity a *pre-group*. The theory $T_{PGH_\&}$, which is obtained by extending the $T_{H_\&}$ to homomorphic encryption over a pre-group operator is convergent and has the

FVP. Indeed, $vc(T_{PGH_{\&}}) = 20$.

The theory $T_{H_{\&}}$ is then further enriched with Abelian group operators and axioms. Again, associativity is approximated. The theory $T_{APGH_{\&}}$, which is obtained by extending the $T_{H_{\&}}$ to homomorphic encryption over an Abelian pre-group operator is convergent and has the FVP. Indeed, its variant complexity is $vc(T_{APGH_{\&}}) = 20$.

The theory $T_{H_{\&}}$ can also be enriched with an Xor operator and axioms. Associativity of the Xor operator is also approximated. The theory $T_{PXorH_{\&}}$, which is obtained by extending the $T_{H_{\&}}$ to homomorphic encryption over a pre-Xor operator is convergent and has the FVP. Indeed, its variant complexity is $vc(T_{PXorH_{\&}}) = 12$.

4.3.2.1 Theory of Homomorphic Encryption over a Free Operator.

In this section we introduce the theories of homomorphic encryption over a free operator with a multiset of keys (with/without the identity). We started with the theory $T_{H_{\&}}$, which is then extended with a decryption operation in $T_{HD_{\&}}$, a projection operation over the free operator in $T_{H^{\triangleright}_{\&}}$. The identity of the multiset of keys is included in the theory $T_{H^{\circ}_{\&}}$

Definition 4.7 ($T_{H_{\&}}$)**.** The theory for homomorphic encryption with a multiset of keys is defined as $T_{H_{\&}} = (\Sigma_{H_{\&}}, B_{H_{\&}}, R_{H_{\&}})$. The signature $\Sigma_{H_{\&}}$ is defined by sorts $\{\mathsf{Key},\ \mathsf{Keys},\ \mathsf{Msg}\}$ with the subsort relation $\mathsf{Key} < \mathsf{Keys}$, $\mathsf{Keys} < \mathsf{Msg}$, and operators

$$\_\&\_ : \mathsf{Keys} \times \mathsf{Keys} \to \mathsf{Keys}, \qquad \_ * \_ : \mathsf{Msg} \times \mathsf{Msg} \to \mathsf{Msg}, \qquad e : \mathsf{Msg} \times \mathsf{Keys} \to \mathsf{Msg}$$

where $\_ * \_$ is a free operator, which can be understood as list concatenation operator, $\_\&\_$ denotes a multiset union operator that is associative and commutative, and e denotes message encryption. The axioms $B_{H_{\&}}$ are AC for $\_\&\_$. There are five rules defined in $R_{H_{\&}}$.

$$e(X,U) * e(Y,U) \to e(X * Y, U) \tag{4.8}$$
$$e(e(X,V),U) \to e(X, U\&V) \tag{4.9}$$
$$e(X,U\&V) * e(Y,U) \to e(e(X,V) * Y, U) \tag{4.10}$$
$$e(X,U) * e(Y,U\&V) \to e(X * e(Y,V), U) \tag{4.11}$$
$$e(X,U\&V) * e(Y,U\&W) \to e(e(X,V) * e(Y,W), U) \tag{4.12}$$

where X, Y are variables of sort Msg, and U, V, W are variables of sort Keys. The equation (4.8) is the reversed version of (4.7). The equation (4.9) captures the property that a nested encryption is simplified into an encryption with a multiset of keys, which is the key point for achieving a theory having the FVP. The remaining rules describe the homomorphic property of an encryption with respect to a multiset of keys, and are added for confluence of the theory.

Note that the order of key application is immaterial in this theory. When the order of the key application matters, the set of actual attack states will be a subset of the attack states with this theory.

Example 4.5. The unification problem of Example 4.2 returns the same unifiers with this new theory $T_{H_\&}$ as with the previous theory T_{kH}.

The extension of $T_{H_\&}$ with a decryption operator is denoted $T_{HD_\&}$. The theory obtained is convergent and has the FVP ($vc(T_{HD_\&}) = 13$).

Definition 4.8 ($T_{HD_\&}$)**.** The theory $T_{HD_\&}$ is defined by extending $T_{H_\&}$ with a decryption operator d : $\mathsf{Msg} \times \mathsf{Keys} \to \mathsf{Msg}$ together with additional rules $R_{H_\&\text{-}Dec}$ which describe encryption/decryption cancellation properties with respect to a multiset of keys:

$$\begin{aligned} d(e(X,U),U) &\to X \\ d(e(X,U\&V),U) &\to e(X,V) \\ d(e(X,U),U\&W) &\to d(X,W) \\ d(e(X,U\&V),U\&W) &\to d(e(X,V),W) \end{aligned}$$

with X of sort Msg, and U, V, W of sort Keys.

Example 4.6. The unification problem of Example 4.3 returns the same unifiers with this new theory $T_{HD_\&}$ as with the previous theory T_{3HD}.

We can also add projection operators to the theory $T_{H_\&}$ (resp. $T_{HD_\&}$) to extract messages from concatenated messages. The theory $T_{H^{\triangleright}_\&}$ (resp. $T_{HD^{\triangleright}_\&}$) is obtained by adding the projection operator p_1 and p_2, which are defined as: $p_1 : \mathsf{Msg} \to \mathsf{Msg}$, $p_2 : \mathsf{Msg} \to \mathsf{Msg}$ to the signature $\Sigma_{H_\&}$ (resp. $\Sigma_{HD_\&}$), together with the following rules to $R_{H_\&}$ (resp. $R_{HD_\&}$):

$$\begin{aligned} p_1(X * Y) &\to X & \qquad p_1(e(X,U)) &\to e(p_1(X),U) \\ p_2(X * Y) &\to Y & \qquad p_2(e(X,U)) &\to e(p_2(X),U) \end{aligned}$$

where X, Y, Z are variables of sort Msg, and U is a variable of sort Keys.

The theories $T_{H^{\triangleright}_\&}$ and $T_{HD^{\triangleright}_\&}$ have the FVP.

Example 4.7. Consider the following unification problem:

$$e(X{:}\mathsf{Msg}, K{:}\mathsf{Key}) =^?$$
$$p_1(e(data(A{:}\mathsf{Name}, r{:}\mathsf{Fresh}) * data(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh}), K'{:}\mathsf{Key})$$

in the theory $T_{HD^{\triangleright}_{\&}}$, where $data$: $\mathsf{Name} \times \mathsf{Fresh} \to \mathsf{Data}$ is an additional operator denoting a principal's secrets, and we assume the subsort relation $\mathsf{Data} < \mathsf{Msg}$. We get the following unifier by variant based unification:

$$\begin{array}{ll} \{\ X{:}\mathsf{Msg} \mapsto data(A1{:}\mathsf{Name}, r1{:}\mathsf{Fresh}), & K{:}\mathsf{Key} \mapsto K1{:}\mathsf{Key}, \\ A{:}\mathsf{Name} \mapsto A1{:}\mathsf{Name}, & r{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh}, \\ B{:}\mathsf{Name} \mapsto B1{:}\mathsf{Name}, & r'{:}\mathsf{Fresh} \mapsto r2{:}\mathsf{Fresh}, \\ K' : \mathsf{Key} \mapsto K1{:}\mathsf{Key}\ \ \} & \end{array}$$

An identity of the multiset of keys can be added to the theory $T_{H_{\&}}$, which we call the theory $T_{H^{\circ}_{\&}}$. The signature $\Sigma_{H_{\&}}$ is extended with a new sort NeKeys denoting a non-empty set of keys, with the subsort relation $\mathsf{Key} < \mathsf{NeKeys}$ and $\mathsf{NeKeys} < \mathsf{Keys}$. The operators & is overloaded by : $\_\&\_$: $\mathsf{NeKeys} \times \mathsf{NeKeys} \to \mathsf{NeKeys}$. The axioms $B_{H^{\circ}_{\&}}$ extends $B_{H_{\&}}$ by including ACU for $\_\&\_$. A new rule $e(X, null) \to X$ is added to $R_{H_{\&}}$, and rules 4.10 and 4.11 in $R_{H_{\&}}$ thus become redundant, i.e., the set of rules in the theory $R_{H^{\circ}_{\&}}$ is:

$$e(X, null) \to X$$
$$e(e(X, V), U) \to e(X, V\&U)$$
$$e(X, K\&V) * e(Y, K\&U) \to e(e(X, V) * e(Y, U), K)$$

where U and V are variables of sort Keys, and K is a variable of sort NeKey.

Example 4.8. Consider a similar unification problem as in Example 4.2

$$X{:}\mathsf{Msg} * Y{:}\mathsf{Msg} =^?$$
$$e(data(A{:}\mathsf{Name}, r{:}\mathsf{Fresh}) * data(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh}), K{:}\mathsf{Keys})$$

in the theory $T_{H^{\circ}_{\&}}$, we get the same unifier as in Example 4.2 together with the following extra unifier by variant based unification.

$$\begin{aligned}
\{\quad & X{:}\mathsf{Msg} \mapsto data(A1{:}\mathsf{Name}, r1{:}\mathsf{Fresh}), && Y{:}\mathsf{Msg} \mapsto data(B1{:}\mathsf{Name}, r2{:}\mathsf{Fresh}),\\
& A{:}\mathsf{Name} \mapsto A1{:}\mathsf{Name}, && r{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh},\\
& B{:}\mathsf{Name} \mapsto B1{:}\mathsf{Name}, && r'{:}\mathsf{Fresh} \mapsto r2{:}\mathsf{Fresh},\\
& K : \mathsf{Key} \mapsto null \quad \}
\end{aligned}$$

This extra unifier can be excluded by changing the variable K in the unification problem to be of sort NeKeys instead of Keys.

The theory $T_{H_\&}$ has the FVP, therefore the general variant-based unification algorithm can be applied. Thus, $T_{H_\&}$ can be combined with other theories with disjoint symbols. For example, consider the theory T_{PK} that we introduced in Section 4.3.1. By combining the theory $T_{H_\&}$ with T_{PK}, we obtain the FVP theort $T_{H_\& PK} = T_{H_\&} \cup T_{PK}$. Similarly, combining the theory $T_{HD_\&}$ with T_{PK}, we get the FVP theory $T_{HD_\& PK} = T_{PK} \cup T_{HD_\&}$.

4.3.2.2 Theory of Homomorphic Encryption over a Pre-Group.

In this section we extend $T_{H_\&}$ to be a homomorphic encryption over a pre-group. For the reasons that we mentioned before, associativity is under approximated by a sub-associativity theory.

Definition 4.9 (T_{PG})**.** The pre-group theory is defined as $T_{PG} = (\Sigma_{PG}, \varnothing, R_{PG})$. There is only one sort Msg, and three group operators $\{\_ * \_, \_^{-1}, 1\}$, where $\_ * \_ : \mathsf{Msg} \times \mathsf{Msg} \to \mathsf{Msg}$ is the group operator, $\_^{-1} : \mathsf{Msg} \to \mathsf{Msg}$ generates the inverse of an element and the constant 1 is the identity. The set of rules R_{PG} contains all group axioms except associativity, where X is a variable of sort Msg:

$$\begin{aligned}
X * 1 &\to X & 1 * X &\to X\\
X * X^{-1} &\to 1 & X^{-1} * X &\to 1\\
(X^{-1})^{-1} &\to X & 1^{-1} &\to 1
\end{aligned}$$

Definition 4.10 ($T_{PGH_\&}$)**.** The theory for homomorphic encryption with a multiset of keys over pre-group is defined as $T_{PGH_\&} = (\Sigma_{H_\&} \cup \Sigma_{PG}, B_{H_\&}, R_{PGH_\&})$, where $R_{PGH_\&} = R_{PG} \cup R_{H_\&} \cup R_{PGH_\&\text{-}Aux}$. The set of rules $R_{H_\&}$ is as defined in the theory $T_{H_\&}$ in Definition 4.7 and R_{PG} is as defined in the theory T_{PG} in Definition 4.9. The following set of rules $R_{PGH_\&\text{-}Aux}$ is added to complete the theory:

$$e(1, U) \to 1 \qquad\qquad (e(X, U))^{-1} \to e(X^{-1}, U)$$

with X of sort Msg and U of sort Keys.

Example 4.9. Recall the unification problem in Example 4.2, with an instance of theory $T_{PGH_{\&}}$, we get all the unifiers in Example 4.2, together with the following unifiers:

$$\begin{aligned}
\{&X{:}\mathsf{Msg} \mapsto e(data(A1{:}\mathsf{Name}, r1{:}\mathsf{Fresh}) * data(B1{:}\mathsf{Name}, r2{:}\mathsf{Fresh}), K1{:}\mathsf{Key}),\\
&Y{:}\mathsf{Msg} \mapsto 1, \qquad\qquad\qquad K{:}\mathsf{Key} \mapsto K1{:}\mathsf{Key},\\
&A{:}\mathsf{Name} \mapsto A1{:}\mathsf{Name}, \qquad r{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh},\\
&B{:}\mathsf{Name} \mapsto B1{:}\mathsf{Name}, \qquad r'{:}\mathsf{Fresh} \mapsto r2{:}\mathsf{Fresh}, \quad \}
\end{aligned}$$

$$\begin{aligned}
\{&Y{:}\mathsf{Msg} \mapsto e(data(A1{:}\mathsf{Name}, r1{:}\mathsf{Fresh}) * data(B1{:}\mathsf{Name}, r1{:}\mathsf{Fresh}), K1{:}\mathsf{Key}),\\
&X{:}\mathsf{Msg} \mapsto 1, \qquad\qquad\qquad K{:}\mathsf{Key} \mapsto K1{:}\mathsf{Key},\\
&A{:}\mathsf{Name} \mapsto A1{:}\mathsf{Name}, \qquad r{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh},\\
&B{:}\mathsf{Name} \mapsto B1{:}\mathsf{Name}, \qquad r'{:}\mathsf{Fresh} \mapsto r2{:}\mathsf{Fresh} \quad \}
\end{aligned}$$

Adding associativity approximation to T_{PG}. Although the theory T_{PG} defines a group without associativity, we can provide several sound approximations of associativity by adding a subtheory of the full associativity theory. This is an under approximation and the completeness (i.e., if there is an attack, an attack will be found) is lost since there may be attacks that can show up with full associativity theory but cannot be found with a sub-associativity theory. Here we introduce some of the possible sub-associativity theories as examples to illustrate this approach. The first associativity approximation below captures the property that a term of sort Nonce can be canceled by its inverse when they are in the two separate ends of a sequence of terms of sort Msg.

Definition 4.11 (T_{PGAA})**.** The theory for a pre-group with associativity approximation T_{PGAA} is defined as $T_{PGAA} = (\Sigma_{PG} \cup \{\mathsf{Nonce}\}, \varnothing, R_{PG} \cup R_{PGAA})$. We assume the subsort relation $\mathsf{Nonce} < \mathsf{Msg}$. The sub-associativity axioms R_{PGAA} are specified as follows, where X is a variable of sort Msg and N is a variable of sort Nonce:

$$\begin{aligned}
(N^{-1} * X) * N &\to X & \qquad N * (X * N^{-1}) &\to X\\
(N * X) * N^{-1} &\to X & \qquad N^{-1} * (X * N) &\to X
\end{aligned}$$

For different applications, different approximation can be chosen. We list below some alternative sub-associativity axioms of R_{PGAA}, which are combined with the pre-group theory T_{PG} in the same way as in T_{PGAA}.

1. Alternative sub-associativity axioms R_{PGAA_1}. This captures the property that a term of sort Nonce can be canceled by its inverse when they are next to each other but not associated together.

$$
\begin{aligned}
(X * N^{-1}) * N &\to X & \qquad N * (N^{-1} * X) &\to X \\
(X * N) * N^{-1} &\to X & \qquad N^{-1} * (N * X) &\to X
\end{aligned}
$$

2. Alternative sub-associativity axioms R_{PGAA_2}. This captures the property that a term of sort Nonce can be canceled by its inverse when they are in the two separate ends of a sequence of terms of sort Msg.

$$
\begin{aligned}
(N * X) * N^{-1} &\to X & \qquad (X * N) * N^{-1} &\to X \\
N^{-1} * (X * N) &\to X & \qquad N^{-1} * (N * X) &\to X
\end{aligned}
$$

3. Alternative sub-associativity axioms R_{PGAA_3}. This captures the property that a term of sort Msg can be canceled by its inverse when they are separated by a nonce.

$$
\begin{aligned}
(X^{-1} * N) * X &\to N & \qquad X * (X^{-1} * N) &\to N \\
(N * X^{-1}) * X &\to N & \qquad X * (N * X^{-1}) &\to N \\
(X * N) * X^{-1} &\to N & \qquad X^{-1} * (X * N) &\to N \\
(N * X) * X^{-1} &\to N & \qquad X^{-1} * (N * X) &\to N \\
X * (X * N^{-1})^{-1} &\to N & \qquad X * (N^{-1} * X)^{-1} &\to N \\
(X * N^{-1})^{-1} * X &\to N & \qquad (N^{-1} * X)^{-1} * X &\to N
\end{aligned}
$$

where X is a variable of sort Msg and N is a variable of sort Nonce.

Remark 4.1. Notice that, in order to get a sound sub-associativity approximation for this theory, it is crucial to take advantage of the order-sorted type structure.

We can also add associativity approximation to the theory of homomorphic encryption over a pre-group by combining $T_{PGH_\&}$ with T_{PGAA}.

Definition 4.12 ($T_{PGAAH_\&}$)**.** Combining the theories $T_{PGH_\&}$ and T_{PGAA}, we obtain the theory $T_{PGAAH_\&} = T_{PGH_\&} \cup T_{PGAA}$.

With different associativity approximations, we similarly obtain the theories $T_{PGAA1H_\&} = T_{PGH_\&} \cup T_{PGAA_1}$ and $T_{PGAA2H_\&} = T_{PGH_\&} \cup T_{PGAA_2}$. For the alternative associativity approximation T_{PGAA_3}, the following auxiliary rules $R_{AA_{3_{Aux}}}$ are needed, i.e., $T_{PGAA3H_\&} =$

$T_{PGH_{\&}} \cup T_{PGAA_3} \cup R_{AA3_{Aux}}$.

$$\begin{array}{ll}
e(X,U) * (N * e(X^{-1},U)) \to N & e(X,U) * (e(X^{-1},U) * N) \to N \\
(N * e(X^{-1},U)) * e(X,U) \to N & (e(X^{-1},U) * N) * e(X,U) \to N \\
(e(X,U) * N) * e(X^{-1},U) \to N & (N * e(X,U)) * e(X^{-1},U) \to N \\
e(X^{-1},U) * (N * e(X,U)) \to N & e(X^{-1},U) * (e(X,U) * N) \to N
\end{array}$$

The theories $T_{PGH_{\&}}$ and $T_{PGAAH_{\&}}$ can be extended by adding a decryption operator, and/or associativity approximations, while remaining FVP.

Definition 4.13 ($T_{PGHD_{\&}}$). The theory $T_{PGHD_{\&}}$ is defined by extending the theory $T_{PGH_{\&}}$ with the decryption operator d : $\mathsf{Msg} \times \mathsf{Keys} \to \mathsf{Msg}$, together with the equation $R_{H_{\&}\text{-}Dec}$ introduced in Definition 4.8, and the auxiliary rule

$$d(1,U) \to 1$$

Definition 4.14 ($T_{PGAAHD_{\&}}$). Adding the same associativity approximation to $T_{PGHD_{\&}}$, we obtain the theory $T_{PGAAHD_{\&}} = T_{PGHD_{\&}} \cup T_{PGAAH_{\&}}$.

Similarly, adding different alternative associativity approximations to $T_{PGHD_{\&}}$, we achieve the FVP theories $T_{PGAA1HD_{\&}} = T_{PGHD_{\&}} \cup T_{PGAA1}$, $T_{PGAA2HD_{\&}} = T_{PGHD_{\&}} \cup T_{PGAA2H_{\&}}$, and $T_{PGAA3HD_{\&}} = T_{PGHD_{\&}} \cup T_{PGAA3H_{\&}}$.

Example 4.10. Consider the following unification problem:

$$\begin{aligned}
&X{:}\mathsf{Nonce} =? \\
&\quad d(e(n(A{:}\mathsf{Name}, r{:}\mathsf{Fresh}) * n(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh}), K{:}\mathsf{Key}), K{:}\mathsf{Key}) \\
&\quad * \, inv(n(A{:}\mathsf{Name}, r{:}\mathsf{Fresh})).
\end{aligned}$$

in the theory $T_{PGAAHD_{\&}}$, where n : $\mathsf{Name} \times \mathsf{Fresh} \to \mathsf{Nonce}$ is an additional operator that generates nonces. We get the following unifier by variant based unification:

$$\begin{array}{ll}
\{X{:}\mathsf{Nonce} \mapsto n(B1{:}\mathsf{Name}, r2{:}\mathsf{Fresh}), & K : \mathsf{Key} \mapsto K1{:}\mathsf{Key} \\
\ A{:}\mathsf{Name} \mapsto A1{:}\mathsf{Name}, & r{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh}, \\
\ B{:}\mathsf{Name} \mapsto B1{:}\mathsf{Name}, & r'{:}\mathsf{Fresh} \mapsto r2{:}\mathsf{Fresh}\}
\end{array}$$

The theory $T_{PGH_{\&}}$ can also be combined with the theory T_{PK} defined in 4.3.1. The resulting theory $T_{PGH_{\&}PK} = T_{PGH_{\&}} \cup T_{PK}$ also has the FVP. Similarly, we can also combine the theory

$T_{PGHD_{\&}}$ or $T_{PGAAH_{\&}}$ with T_{PK} and derive more theories enjoying the finite variant property.

4.3.2.3 Theory of Homomorphic Encryption over an Abelian Pre-Group.

We further approximate the theory AGH by adding a commutative axiom to the binary group operator in the pre-group. This provides a theory for homomorphic encryption over an Abelian pre-group.

Definition 4.15 (T_{APG})**.** The theory of Abelian pre-groups $T_{APG} = (\Sigma_{APG}, B_{APG}, R_{APG})$ is obtained from the theory of pre-group T_{PG} by adding as axioms B_{APG} the commutativity equation $X * Y = Y * X$. Because of commutativity, the rules $1 * X \to X$ and $X * X^{-1} \to 1$ in T_{PG} become redundant.

We can similarly provide a sound approximation of associativity by adding a subtheory of the full associativity theory. Here again, we introduce some possible sub-associativity theories for Abelian pre-group as examples to illustrate this approach. The first approximation captures the property that a term of sort Nonce can be canceled by its inverse when they are in the two separate ends of a sequence of terms of sort Msg. Since $*$ is commutative in Abelian pre-groups, less rules are needed comparing to the ones for pre-groups (e.g., T_{PGAA}).

Definition 4.16 (T_{APGAA})**.** The theory for an Abelian pre-group with associativity approximation T_{APGAA} is defined as $T_{APGAA} = (\Sigma_{APG} \cup \{\mathsf{Nonce}\}, B_{APG}, R_{APG} \cup R_{APGAA})$. We assume the subsort relation $\mathsf{Nonce} < \mathsf{Msg}$. The sub-associativity rules R_{APGAA} are specified as follows:

$$(N^{-1} * X) * N \to X \qquad\qquad (N * X) * N^{-1} \to X$$

with X of sort Msg and N of sort Nonce.

We also present a theory for an Abelian pre-group with an alternative associative approximation T_{APGAA1}. The theory T_{APGAA1} combines a sub-associativity theory with the Abelian pre-group theory T_{APG} in the same way as in T_{APGAA}, except that it uses an alternative set of sub-associativity rules R_{APGAA1}. This captures the property that a term of sort Msg can be canceled by its inverse when they are separated by a nonce. The sub-associativity rules are as follows:

$$(X^{-1} * N) * X \to N \qquad\qquad (X * N) * X^{-1} \to N$$
$$X * (X * N^{-1})^{-1} \to N$$

The homomorphic encryption with a multiset of keys over an Abelian pre-group is thus defined by combining the theories $T_{H_\&}$ and T_{APG} as follows:

Definition 4.17 ($T_{APGH_\&}$)**.** The theory of homomorphic encryption with a multiset of keys over an Abelian pre-group is defined as $T_{APGH_\&} = (\Sigma_{H_\&} \cup \Sigma_{APG}, B_{APGH_\&}, R_{PGH_\&} \cup R_{PGH_\&\text{-}Aux})$. The axioms $B_{APGH_\&}$ define the commutativity property of $\_ * \_$ and AC of $\_\&\_$.

Example 4.11. For the unification problem in Example 4.2, with an instance of the theory $T_{APGH_\&}$, we found the unifiers described in Example 2, together with the following unifier:

$$\begin{aligned}
&\{X \mapsto e(data(B1{:}\mathsf{Name}, r1{:}\mathsf{Fresh}), K1{:}\mathsf{Key}),\\
&\ Y \mapsto e(data(A1{:}\mathsf{Name}, r1{:}\mathsf{Fresh}), K1{:}\mathsf{Key}),\\
&A{:}\mathsf{Name} \mapsto A1{:}\mathsf{Name}, \qquad r'{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh},\\
&B{:}\mathsf{Name} \mapsto B1{:}\mathsf{Name}, \qquad K : \mathsf{Key} \mapsto K1{:}\mathsf{Key}\}
\end{aligned}$$

The theory $T_{APGH_\&}$ can be extended by adding a decryption operator, and/or associativity approximations. Both extensions have the FVP.

Definition 4.18 ($T_{APGHD_\&}$)**.** $T_{APGHD_\&}$ is obtained by adding decryption operator $d : \mathsf{Msg} \times \mathsf{Keys} \to \mathsf{Msg}$ to the signature of $T_{APGH_\&}$, together with the set of encryption/decryption cancellation equations $R_{H_\&\text{-}Dec}$ in Definition 5, and together with the auxiliary rule $d(1, U) \to 1$.

By Combining the theory $T_{APGH_\&}$ with T_{APGAA}, we obtain the theory of homomorphic encryption with a multiset of keys over Abelian pre-group with associativity approximation, which is defined as: $T_{APGAAH_\&} = T_{APGH_\&} \cup T_{APGAA}$.

To combine the theory $T_{APGH_\&}$ with the alternative associativity approximation T_{APGAA1}, the following auxiliary rules are added to achieve the theory $T_{APGAA1H_\&}$:

$$(N * e(X^{-1}, U)) * e(X, U) \to N \qquad\qquad (N * e(X, U)) * e(X^{-1}, U) \to N$$

Adding the associativity approximation to $T_{APGHD_\&}$, we obtain the theories $T_{APGAAHD_\&} = T_{APGHD_\&} \cup T_{APGAAH_\&}$, and $T_{APGAA1HD_\&} = T_{APGHD_\&} \cup T_{APGAA1H_\&}$.

We can combine the theory $T_{APGH_\&}$ with T_{PK} to get the theory $T_{APGH_\& PK} = T_{APGH_\&} \cup T_{PK}$, which also has the finite variant property. Similarly, we can define the theory $T_{APGHD_\& PK} = T_{APGHD_\&} \cup T_{PK}$. A similar combination approach can be applied to $T_{APGAAH_\&}$ and $T_{APGAA1H_\&}$ to obtain new theories with the FVP.

4.3.2.4 Theory of Homomorphic Encryption over Pre-Xor.

Based on the theory of homomorphic encryption over an Abelian pre-group, in this section we approximate the theory AGH by adding pre-Xor axioms instead of Abelian pre-group axioms to the binary operator $*$. Again, associativity of $*$ is under approximated by a sub-associativity theory. This provides a theory for homomorphic encryption over pre-Xor.

Definition 4.19 (T_{PXor})**.** The pre-Xor theory is defined as $T_{PXor} = (\Sigma_{PXor}, B_{PXor}, R_{PXor})$. There is only one sort Msg, and two operators $\{\_ * \_, 1\}$, where $\_ * \_ \; : \; \mathsf{Msg} \times \mathsf{Msg} \to \mathsf{Msg}$ is the binary Xor operator and the constant 1 is the identity. The axioms B_{PXor} contain the commutativity equation $X * Y = Y * X$. The rules R_{PXor} are specified as follows, where X is a variable of sort Msg:

$$X * X \to 1 \qquad\qquad X * 1 \to X$$

The associativity of the Xor operator $*$ can be approximated as follows:

Definition 4.20 (T_{PXorAA})**.** The theory for a pre-Xor with associativity approximation T_{PXorAA} is defined as $T_{PXorAA} = (\Sigma_{PXor}, B_{PXor}, R_{PXor} \cup R_{XorAA})$. The set of sub-associativity axioms R_{XorAA} contains the equation

$$X * (X * Y) = Y$$

where X and Y are variables of sort Msg.

Note that, although this associativity approximation contains only one equation, it still covers a fair amount of different cases, since both X and Y have the top sort Msg.

Definition 4.21 ($T_{PXorH_\&}$)**.** The theory for homomorphic encryption with a multiset of keys over pre-Xor is defined by combining the theories $T_{H_\&}$ and T_{PXor} as follows: $T_{PXorH_\&} = (\Sigma_{H_\&} \cup \Sigma_{PXor}, B_{H_\&} \cup B_{PXor}, R_{PXorH_\&})$, where $R_{PXorH_\&} = R_{PXor} \cup R_{H_\&} \cup R_{PXorH_\&\text{-}Aux}$. The set $R_{PXorH_\&\text{-}Aux}$ contains only one rule $e(1, U) \to 1$, where U a variable of sort Keys.

Example 4.12. For the unification problem in Example 4.2, with an instance of the theory $T_{PXorH_\&}$, we find all the unifiers described in Example 4.11, together with the following unifier:

$$
\begin{array}{ll}
\{X{:}\mathsf{Msg} \mapsto Z{:}\mathsf{Msg}, & Y{:}\mathsf{Msg} \mapsto Z{:}\mathsf{Msg} \\
A{:}\mathsf{Name} \mapsto C{:}\mathsf{Name}, & r{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh}, \\
B{:}\mathsf{Name} \mapsto C{:}\mathsf{Name}, & r'{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh}, \\
K : \mathsf{Key} \mapsto K1{:}\mathsf{Key}\} &
\end{array}
$$

We can also add a decryption operator, and/or associativity approximations to the theory of homomorphic encryption over Xor, while remaining FVP.

Definition 4.22 ($T_{PXorAAH_{\&}}$). Combining the theories $T_{PXorH_{\&}}$ and T_{PXorAA}, we obtain the theory $T_{PXorAAH_{\&}} = T_{PXorH_{\&}} \cup T_{PXorAA}$.

Definition 4.23 ($T_{PXorHD_{\&}}$). The theory $T_{PXorHD_{\&}}$ is defined by extending the theory $T_{PXorH_{\&}}$ with the decryption operator d : $\mathsf{Msg} \times \mathsf{Keys} \to \mathsf{Msg}$, together with the equation $R_{H_{\&}\text{-}Dec}$ introduced in Definition 4.8, and the auxiliary rule $d(1, U) \to 1$.

The theory that includes both the decrypiton operator and the associativity approximation is: $T_{PXorAAHD_{\&}} = T_{PXorHD_{\&}} \cup T_{PXorAAH_{\&}}$.

Example 4.13. Consider the following unification problem:

$$
\begin{array}{l}
X{:}\mathsf{Nonce} =? \\
\quad d(e(e(n(A{:}\mathsf{Name}, r{:}\mathsf{Fresh}), K'{:}\mathsf{Key}) * n(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh}), K{:}\mathsf{Key}), K{:}\mathsf{Key}) \\
\quad * \, e(n(A{:}\mathsf{Name}, r{:}\mathsf{Fresh}), K'{:}\mathsf{Key}).
\end{array}
$$

in the theory $T_{PXorAAHD_{\&}}$, where $data$: $\mathsf{Name} \times \mathsf{Fresh} \to \mathsf{Data}$ is an additional operator that generates principal's data, and we assume the subsort relation $\mathsf{Data} < \mathsf{Msg}$. We get the following unifier by variant based unification:

$$
\begin{array}{ll}
\{X{:}\mathsf{Nonce} \mapsto n(B1{:}\mathsf{Name}, r2{:}\mathsf{Fresh}), & K : \mathsf{Key} \mapsto K1{:}\mathsf{Key} \\
A{:}\mathsf{Name} \mapsto A1{:}\mathsf{Name}, & r{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh}, \\
B{:}\mathsf{Name} \mapsto B1{:}\mathsf{Name}, & r'{:}\mathsf{Fresh} \mapsto r2{:}\mathsf{Fresh}, \\
K'{:}\mathsf{Key} \mapsto K2{:}\mathsf{Key}\} &
\end{array}
$$

4.3.3 Homomorphic Theories: Theory of Homomorphic Encryption over Two Groups

In this section we introduce theories of homomorphic encryption over two different groups. Since this is essentially a function mapping from one group to another, it does not have recursive calls, allowing the theories to have the FVP.

We first introduce the theory T_{2AGH}, which defines homomorphic encryption over two Abelian groups. This theory is defined based on the decomposition of an Abelian group by Lankford in [90], which was proved to have the FVP by [50, 57]. The theory T_{AG} has the FVP. Indeed, $vc(T_{2AGH}) = 2276$.

Notice that the variant complexity of the theory T_{2AGH} is really high. To achieve a lower variant complexity, we introduce the theory T_{2XORH}, which defines a homomorphic encryption over two Exclusive-or(Xor) theories. This is an over approximation of homomorphic encryption over two Abelian groups. The variant complexity of this theory is much lower, indeed, $vc(T_{2XORH}) = 48$.

4.3.3.1 Theory of Homomorphic Encryption over Two Abelian Groups

Definition 4.24 (T_{AG})**.** The theory of Abelian groups is defined as $T_{AG} = (\Sigma_{AG}, B_{AG}, R_{AG})$. The signature Σ_{AG} is defined by sort AG and the set of Abelian-group operators

$$\_ * \_ : AG \times AG \to AG$$
$$\_^{-1} : AG \to AG \qquad\qquad 1 :\to AG$$

The axioms B_{AG} are associativity and commutativity axioms for $\_ * \_$ .The set of rules R_{AG} define all other Abelian group axioms:

$$\begin{array}{ll}
X * 1 \to X & X^{-1} * Y^{-1} \to (X * Y)^{-1} \\
X * (X^{-1}) \to 1 & (X * Y)^{-1} * Y \to (X)^{-1} \\
(X^{-1})^{-1} \to X & (X^{-1} * Y)^{-1} \to X * (Y^{-1}) \\
1^{-1} \to 1 & X^{-1} * (Y^{-1} * Z) \to (X * Y)^{-1} * Z \\
X * (X^{-1} * Y) \to Y & (X * Y)^{-1} * (Y * Z) \to X^{-1} * Z
\end{array}$$

with X, Y, Z of sort AG.

Definition 4.25 (T_{2AGH})**.** The theory of homomorphic encryption over two Abelian groups is defined as $T_{2AGH} = (\Sigma_{2AGH}, B_{2AGH}, R_{2AGH})$. The signature Σ_{2AGH} is defined by sorts $\{\mathsf{AG_a},\ \mathsf{AG_b}\}$, and the homomorphic encryption operator $e : \mathsf{AG_a} \to \mathsf{AG_b}$, together with the set of group operators for the domain Abelian group $\mathsf{AG_a}$, which are $\{*_a, 1_a, \_^{-1_a}\}$, and the set of group operators for codomain Abelian group $\mathsf{AG_b}$, which are $\{*_b, 1_b, \_^{-1_b}\}$. Notice that there is no key explicitly defined here, since encryption with a specific key is implicitly captured by the definition of the encryption operator e. The axioms B_{2AGH} are AC axioms for the binary Abelian group operators $\_ *_a \_$, $\_ *_b \_$. The set of rules R_{2AGH} defines the remaining Abelian group axioms of AG_a and AG_b, which are instances of R_{AG} in T_{AG} together with the homomorphic property of e:

$$e(X) *_b e(Y) \to e(X *_a Y)$$
$$e(X) *_b e(Y) *_b Z \to e(X *_a Y) *_b Z$$

The following rules are added to complete the theory.

$$e(1_a) \to 1_b$$
$$e(X)^{-1_a} \to e((X)^{-1_b})$$
$$(e(X) *_b Z)^{-1_b} \to e(X^{-1_a}) *_b (Z^{-1_b})$$

with X, Y of sort $\mathsf{AG_a}$, and Z of sort $\mathsf{AG_b}$

Example 4.14. Consider the following unification problem:

$$X{:}\mathsf{AG_b} =^? e(data(A{:}\mathsf{Name}, r{:}\mathsf{Fresh}) *_a data(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh})) \\ *_b e(data(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh})^{-1_a})$$

in the theory T_{2AGH}, where $data : \mathsf{Name} \times \mathsf{Fresh} \to \mathsf{Data}$ is an additional operator that generates principal's secrets, and we assume the subsort relation $\mathsf{Data} < \mathsf{AG_a}$. We get the following unifier by variant based unification:

$$\begin{aligned}\{&X{:}\mathsf{AG_b} \mapsto e(data(A1{:}\mathsf{Name}, r1{:}\mathsf{Fresh})),\\ &A{:}\mathsf{Name} \mapsto A1{:}\mathsf{Name}, \quad r{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh},\\ &B{:}\mathsf{Name} \mapsto B1{:}\mathsf{Name}, \quad r'{:}\mathsf{Fresh} \mapsto r2{:}\mathsf{Fresh}\}\end{aligned}$$

The unification problem:

$$X{:}\mathsf{AG_b} *_b Y{:}\mathsf{AG_b} =^? e(data(A{:}\mathsf{Name}, r{:}\mathsf{Fresh}) *_a data(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh})$$

in the theory T_{2AGH} get 64 unifiers by variant based unification, which we omit here.

We can also extend T_{2AGH} by adding decryption operator in the following way:

Definition 4.26 (T_{2AGHD}). T_{2AGHD} is obtained by adding a decryption operator $d : \mathsf{AG_b} \to$

$\mathsf{AG_a}$ to the signature of T_{2AGH}, together with the rules $d(1_b) \to 1_a$ and $d(e(X)) \to X$ with X of sort $\mathsf{AG_a}$.

4.3.3.2 Theory of Homomorphic Encryption over Two Xor Operators

Definition 4.27 (T_{2XORH})**.** The theory for homomorphic encryption over two Xor operators is defined as $T_{2XORH} = (\Sigma_{2XORH}, B_{2XORH}, R_{2XORH})$. The signature Σ_{2XORH} is defined by sorts $\{\mathsf{Xor_a},\ \mathsf{Xor_b}\}$, and the homomorphic encryption operator $e : \mathsf{Xor_a} \to \mathsf{Xor_b}$, together with the Xor operator and the corresponding identity of $\mathsf{Xor_a}$, which are $\{*_a, 1_a\}$, and together with the Xor operator and the corresponding identity of $\mathsf{Xor_b}$, which are $\{*_b, 1_b\}$. Notice again that there is no key explicitly defined here. The axioms B_{2XORH} are associativity and commutativity axioms for $\_ *_a \_, \_ *_b \_$. The set of rules R_{2XORH} are defined as following:

$$\begin{aligned} X *_a X &\to 1_a & \qquad P *_b P &\to 1_b \\ X *_a 1_a &\to X & P *_b 1_b &\to P \\ X *_a X *_a Y &\to Y & P *_b P *_b Q &\to Q \end{aligned}$$

together with the homomorphic property of e:

$$e(1_a) \to 1_b$$
$$e(X) *_b e(Y) \to e(X *_a Y)$$
$$e(X) *_b e(Y) *_b P \to e(X *_a Y) *_b P$$

with X, Y of sort $\mathsf{Xor_a}$, and P, Q of sort $\mathsf{Xor_b}$

We can also extend T_{2XORH} by adding a decryption operator in the similar way as in T_{2AGHD}.

Definition 4.28 (T_{2XORHD})**.** The theory T_{2XORHD} is obtained by adding a decryption operator $d : \mathsf{Xor_b} \to \mathsf{Xor_a}$ to the signature of T_{2XORH}, together with rules $d(1_b) \to 1_a$ and $d(e(X)) \to X$ with X of sort $\mathsf{Xor_a}$.

Example 4.15. Consider the following unification problem:

$$X{:}\mathsf{Xor_b} =^?$$
$$e(data(A{:}\mathsf{Name}, r'{:}\mathsf{Fresh}) *_a data(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh})) *_b e(data(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh}))$$

in the theory T_{2XORH}, where $data$: $\mathsf{Name} \times \mathsf{Fresh} \to \mathsf{Data}$ is an additional operator that generates principal's secrets, and we assume the subsort relation $\mathsf{Data} < \mathsf{Xor_a}$. We get the following unifier by variant based unification:

$$\begin{aligned}
&\{X{:}\mathsf{Xor_b} \mapsto e(data(A1{:}\mathsf{Name}, r1{:}\mathsf{Fresh})),\\
&\ A{:}\mathsf{Name} \mapsto A1{:}\mathsf{Name}, \qquad r{:}\mathsf{Fresh} \mapsto r1{:}\mathsf{Fresh},\\
&\ B{:}\mathsf{Name} \mapsto B1{:}\mathsf{Name}, \qquad r'{:}\mathsf{Fresh} \mapsto r2{:}\mathsf{Fresh}\}
\end{aligned}$$

The unification problem:

$$X{:}\mathsf{Xor_b} *_b Y{:}\mathsf{Xor_b} =^? e(data(A{:}\mathsf{Name}, r{:}\mathsf{Fresh}) *_a data(B{:}\mathsf{Name}, r'{:}\mathsf{Fresh})$$

in the theory T_{2XorH} has 17 unifiers by variant based unification, which we omit here.

4.4 EXPERIMENTS

In this section we describe the experiments we have performed on various of the Multiparty Computation Protocol and Homomorphic Hash Protocol that were previously specified and analyzed in [70]. In that paper a number of protocols were analyzed in Maude-NPA using a dedicated unification algorithm for equational theory H. That is, Equation (4.7) was used and nothing else. This was necessary because the special-purpose unification algorithm was not easily combinable with other theories. However, since in the present work we are using variant unification, we have much more freedom with respect to the equations we can include, as long as the theories satisfy FVP. Extra security properties are therefore analyzed in this work thanks to the richer equational theories.

The experiments in this section serve several purposes. The first is to determine which of the theories we have generated are suitable for cryptographic protocol analysis. The second is to evaluate the variant complexity metric defined in this work. How well does lower variant complexity correlate with performance, and if it does, at what point does higher variant complexity begin to make analysis impossible? The third is to use the experimental results to gain insights into how performance can be improved.

The protocols have been specified and analyzed using the Maude-NPA cryptographic protocol analysis tool. One uses Maude-NPA by specifying an insecure state, called an *attack state*, from which Maude-NPA searches backwards. If it finds a path to an initial state then it has found an attack on the protocol. If it terminates without reaching an initial state then the attack state has been proven unreachable modulo the axioms of the given equational theory.[3]

4.4.1 Multiparty Computation Protocol.

In the Multiparty Computation Protocol an initiator Alice and a responder Bob send messages encrypted with a homomorphic public key encryption operator e to a server who combines the encrypted data using an operator $*$. As a result of participating in the protocol, both Alice and Bob are supposed to receive a homomorphically encrypted version of $D_A * D_B$, where D_A is Alice's secret data and D_B is Bob's secret data, without either learning the other's secret. However, it is possible for Alice to accept data that did not come from Bob if she is not able to distinguish $D_A * D_B$ from nonsense. If she is able to, no authentication attack is possible. We specified two attack states: one in which Alice cannot reject nonsense, and one in which she can.

The protocol itself proceeds as follows:

1. $A \rightarrow B : sign(B; N_A; pk(e(D_A, pkey(A,B)), S), A)$
   A starts by encrypting her data first under the homomorphic public key, then under the server's public key. She then attaches a nonce and B's name, signs it, and sends it to B.

2. $B \rightarrow A : sign(N_A; N_B; pk(e(D_B, pkey(A,B)), S), B)$
   B sends a similar message to A, including both his and A's nonce.

3. $A \rightarrow S : sign(A; B; N_A; N_B; pk(e(D_A, pkey(A,B)), S);$
   $pk(e(D_B, pkey(A,B)), S), A)$
   A sends a signed message containing both nonces and both encrypted data sets to S.

4. $S \rightarrow A, B : sign(A; B; N_A; N_B;$
   $sign(e(D_A, pkey(A,B)) * e(D_B, pkey(A,B)), S)$
   The server combines both encrypted data sets using $*$ and sends the result to A and B. They can now decrypt it to obtain $D_A * D_B$.

The attack runs as follows:

[3]Of course, if the given equational theory only under-approximates the intended equational theory, there still may be an attack not detectable due to the under-approximation.

1. $A \to I(B) : sign(B; N_A; pk(e(D_A, pkey(A,B)), S), A)$
   A initiates the protocol with B.

2. $I \to B : sign(B; N_A; E, I)$
   I intercept's A's message, and uses it to create a message for B. The message E could or could not be A's encrypted data. This is irrelevant to the attack.

3. $B \to A : sign(N_A; N_B; pk(e(D_B, pkey(I,B)), S), B)$
   B believes that he is talking to I and sends the corresponding reply message. I forwards it to A.

4. $A \to S : sign(A; B; N_A; N_B;$
   $$pk(e(D_A, pkey(A,B)), S); pk(e(D_B, pkey(I,B)), S), A)$$
   A now forwards both encrypted data sets to the server S, who removes the outer layer of encryption, applies f, and sends the results back to A and B.

5. $S \to A, B : sign(A; B; N_A; NB;$
   $$sign(e(D_A, pkey(A,B)) * e(D_B, pkey(I,B)), S)$$
   If A now attempts to decrypt the result of S's computation with her private key corresponding to $pkey(A,B)$, she will get nonsense, because one of the data sets was encrypted with $pkey(I,B)$. Depending upon whether or not A can recognize that she has received nonsense, this can be used to prevent this attack.

We thus specify two versions of this protocol : one in which A verifies that she has received $e(X * Y,\ pkey(A,B))$ for some X and Y, and one in which she does not.

If, in addition, we assume that $*$ is an Abelian group operator, there are several attacks in which Bob can learn Alice's secret (and vice versa). In the first, Bob simply sends the unit 1 as his data and receives $D_A * 1 = D_A$ in return. In the second, Bob sends his correct data and receives $D_A * D_B$, and multiplies by $(D_B)^{-1}$ to obtain D_A. These attacks, although simple, were of interest to us because they follow from the Abelian group properties of $*$, and so we ran Maude-NPA on an attack state in which Bob learns Alice's secret, using the homormorphic encryption over an Abelian pre-group with associativity approximation. To demonstrate the associativity approximation's influence on performance, we investigated two such theories: $T_{APGAAH1_{\&}}$ is the theory $T_{APGH_{\&}}$ with the approximation equation: $(D_1 * D_2) * (D_1)^{-1} = D_2$. $T_{APGAAH2_{\&}}$ is the theory $T_{APGH_{\&}}$ with the approximation equation: $(D_1 * X) * (D_1)^{-1} = X$ with X of sort Msg, and D_1 and D_2 of sort Data, which is a subsort of Msg. Notice that we start from an associativity approximation that is just expressive enough for this specific protocol and then try a more expressive one.

Steps	T_{3H}		$T_{H_{\&}}$		$T_{APGH_{\&}}$	
	States	Time(ms)	States	Time(ms)	States	Time(ms)
Step 1	5	15941	8	1114538	11	4885493
Step 2	6	34298	9	1557783	13	4800678
Step 3	1	36514	2	2194422	2	8633163
Step 4	1	3084	1	6174	1	38997
Step 5	3	2409	3	5287	3	28363
Step 6	6	8106	6	29910	6	45339
Step 7	4	19615	4	103886	4	416986
Step ...	...	...	...	...	...	...
Step 15	5	44835	5	187119	5	321147

Table 4.3: Results for authentication of Bob to Alice

Steps	T_{3H}		$T_{H_{\&}}$		$T_{APGH_{\&}}$	
	States	Time(ms)	States	Time(ms)	States	Time(ms)
Step 1	3	26183	4	1308574	8	3513537
Step 2	2	13251	3	1382475	6	4867452
Step 3	1	8489	1	2203056	1	8218407
Step 4	0	2974	0	4933	0	2775

Table 4.4: Results for authentication of Bob to a stronger Alice

Note that, additional equations needed can be approximated using attacker rules when the theories themselves are not expressive enough. Approximation via attacker rules is less expressive than using equational theories, but in some cases, depending both on the theory and the protocol, they can be proven equivalent with respect to specific security properties.

We tried the authentication attack on the protocol specified with the theories T_{kH} (bounded homomorphism, variant complexity 4), $T_{H_{\&}}$ (multi set of keys with free operator, variant complexity 8), and $T_{APGH_{\&}}$ (Abelian pre-group,variant complexity 20). The results are given in Table 4.3 for the insecure version of the protocol and Table 4.4 for the second version of the protocol. We note that all the theories we obtained can be used for analyzing this attack, since the higher the variant complexity, the longer the analysis time, we thus consider only three equational theories. For the secrecy attack we investigated $T_{APGAAH1_{\&}}$ and $T_{APGAAH2_{\&}}$ as well as the theories T_{2AGH} (homomorphism between two Abelian groups, variant complexity 2276) and T_{2XOR} (homomomorphism between two exclusive-or theories, variant complexity 48), since Abelian group axioms are necessary for the secrecy attack. The results for $T_{APGAAH1_{\&}}$ and $T_{APGAAH2_{\&}}$ are given in Table 4.5. The attack patterns are specified in Maude-NPA as follows: the state `ATTACK-STATE(0)` indicates that Alice finished her strand without the corresponding Bob's strand and Alice cannot check the format of the received message.

```
eq ATTACK-STATE(0)
   = :: r, r' ::  *** Alice ***
     [ nil ,
     +(sign(b; n(a,r); pk(e(data(a,r'),pkey(a,b)),s),a)),
     -(sign(n(a,r); N; Y1 ,b )),
     +(sign(a; b; n(a,r); N; pk(e(data(a,r'),pkey(a,b)),s); Y1, a)),
     -(sign(a; b; n(a,r); N; X3 * X4, s )) |  nil ]
     || empty
     || nil
     || nil
     || never(
     :: r1, r2  :: ***Bob ***
     [ nil |
      -(sign(b; n(a,r); pk(e(data(a,r'),pkey(a,b)),s), a)),
     +(sign(n(a,r); n(b,r1); pk(e(data(b,r2),pkey(a,b)),s), b)),
       nil ]
     & S:StrandSet || IK:IntruderKnowledge)
```

The state `ATTACK-STATE(1)` checks the same authentication property as `ATTACK-STATE(0)` but with a stronger Alice who can tell whether she received nonsense or not.

```
 eq ATTACK-STATE(1)
   = :: r, r' ::  *** Alice ***
     [ nil ,
     +(sign(b; n(a,r); pk(e(data(a,r'),pkey(a,b)),s),a)),
     -(sign(n(a,r); N; Y1, b)),
     +(sign(a; b; n(a,r); N; pk(e(data(a,r'),pkey(a,b)),s); Y1, a)),
     -(sign(a; b; n(a,r); N; e(X3 * X4, pkey(a, b)), s)) |  nil ]
     || empty
     || nil
     || nil
     || never(
     :: r1, r2  :: ***Bob ***
     [ nil |
      -(sign(b; n(a, r); pk(e(data(a,r'),pkey(a,b)),s), a)),
      +(sign(n(a, r); n(b,r1); pk(e(data(b,r2),pkey(a,b)),s), b)),
       nil ]
     & S:StrandSet || IK:IntruderKnowledge)
```

Tables 4.3 and 4.4 show the number of states generated by Maude-NPA and the amount of time taken for each step of the backwards reachability analysis. In Table 4.3, for each theory Maude-NPA found the authentication attack in six steps. However, in each case Maude-NPA failed to terminate and kept generating five states after fifteen steps. Upon investigation, these appeared to be infinite paths that were not discarded by the Maude-NPA state space reduction techniques. As we can also see from the table, as the variant complexity of theories involved in the specification grows, the number of states and the time needed for Maude-NPA to find the attack also grows, as well as the time it takes to complete each step. Furthermore, the time it takes to complete a step increases with the variant complexity of the theory even when the number of states generated at the step is the same for all three theories. We conjecture that this is the result of Maude-NPA generating many states that are then removed by the state space reduction mechanisms. Greater variant complexity means that more failed states are generated as well as successful ones.

For the second attack analysis, when Alice can tell whether she received nonsense or not, we verified that there is no authentication attack between Alice and Bob. Table 4.4 shows the number of states and attacks generated by Maude-NPA in each step for the attack state with different theories described above. For each theory Maude-NPA terminated at Step 4. We note a similar relationship between between performance and variant complexity as in Table 4.3.

The secrecy attack is specified in Maude-NPA by following attack pattern:

```
eq ATTACK-STATE(2)
 = :: r, r' ::
   [ nil ,
   +(sign(i; N; pk(e(data(a,r'),pkey(a,i)),s),a)),
   -(sign(n(a,r); N; Y1, i)),
   +(sign(a; i; n(a,r); N; pk(e(data(a,r'),pkey(a,i)),s); Y1, a)),
   -(sign(a; i; n(a,r); N; e(X1, pkey(a, i)), s )) |  nil]
   || data(a, r') inI
```

For the secrecy attack we found that the theories T_{2AGH} and T_{2XOR} gave very discouraging results. For Theory T_{2AGH} Maude-NPA was not even able to complete Step 1, even for a simpler version of the protocol we constructed (see below). For theory T_{2XOR} Maude-NPA did a little better; it was able to complete Steps 1 and 2, but not Step 3. This is not surprising, given the high variant complexity of the theories. We did not investigate these two theories any further.

Steps	*Original*			
	$T_{APGAAH1_{\&}}$		$T_{APGAAH2_{\&}}$	
	States	Time	States	Time
Step 1	8	133571	8	250829
Step 2	11	7196213	17	8383711
Step 3	13	13728328	44	25724593
Step 4	8	35712864	106	199325826
Step 5	8	21233267		(timeout)
Step 6	11	30402427		
Step 7	23	40922662		
Step 8	35	62267212		
Step 9		(timeout)		

Steps	*Simplified*			
	$T_{APGAAH1_{\&}}$		$T_{APGAAH2_{\&}}$	
	States	Time	States	Time
Step 1	12	36965	12	66910
Step 2	14	38855	32	67577
Step 3	11	52049	72	323521
Step 4	7	34262	179	1486682
Step 5	10	21881	482	15140859
Step 6	8	29695		(timeout)
Step 7	8	18233		
Step 8	9	22508		
...	...	...		
Step 12	4	4534		

Table 4.5: Results for secrecy

Even with $T_{APGAAH1_{\&}}$ and $T_{APGAAH2_{\&}}$, Maude-NPA struggled to find the secrecy attack, as we can see in Table 4.5 for $T_{APGAAH2_{\&}}$. We thus tried Maude-NPA on a simpler version of the protocol to get a better idea of the performance tradeoffs, omitting the checks for authentication and freshness. This simplification is intended to reduce search space while keeping the part of the protocol that is of interest to us. Maude-NPA was able to find the two attacks in five steps for the simplified protocols. The search of the protocol with $T_{APGAAH1_{\&}}$ terminated after 12 steps and 4 possible attack sequences were found, while the one with $T_{APGAAH2_{\&}}$ took a much longer time for each search step and suffered from state explosion. Even so, it was able to produce 5 attack sequences before the explosion.

Since the associativity approximation of $T_{APGAAH1_{\&}}$ is more restrictive than that of $T_{APGAAH2_{\&}}$ (indeed it is a special case of it), less variants are generated for the same term, which reduced the state generation time. This result thus shows the tradeoff needed between a more general theory and performance in practice.

4.4.2 Homomorphic Hash Protocol.

In this protocol, an initiator Alice and a responder Bob communicate to agree on a secret N_s. The messages are encrypted using a shared key, together with a hash function for the integrity of the messages. But let us suppose that the hash function h has a fatal flaw: the hash function h is homomorphic. This flaw leads to a possible authentication attack. The protocol proceeds as follows:

1. $A \rightarrow B: \ A; N_A$
   The protocol starts by A sending her name and a nonce to B.

2. $B \rightarrow A: N_B; shk(N_s * N_A, K_{AB}); shk(h(N_B * N_s * N_A, K_{AB}), K_{AB})$
   After receiving the message from A, B generates his nonce and the intended secret N_s. B then responds with his nonce, together with two messages that are encrypted with a shared key. To guarantee the integrity, one of the encrypted messages includes a hash of both A and B's nonces and the secret.

3. $A \rightarrow B: shk(N'_A, K_{AB}); shk(h(N_B * N'_A, K_{AB}), K_{AB})$
   A confirms that she received the message.

Now taking advantage of the homomorphic property of the hash function, an intruder can pretend to be Alice and trick Bob into believing that he has agreed on a secret with Alice. The flaw is as follows:

1. $I(A) \rightarrow B: A; N_I$
   I pretends to be A, and send B A's name and its nonce.

2. $B \rightarrow I(A): N_B; shk(N_s; N_I, K_{AB}); shk(h(N_B * N_s * N_I, K_{AB}), K_{AB})$
   B responds to A, which is intercepted by I.

3. $I(A) \rightarrow B: shk(N_s; N_I, K_{AB}); shk(h(N_B * N_s * N_I, K_{AB}), K_{AB})$
   I sends B the encrypted part of the message that it intercepted in the previous step. Since h is homomorphic over $*$, we have $h(N_B * N_s * NA, \ K_{AB}) = h(N_B \ K_{AB}) * h(N_s * NA \ K_{AB})$. Therefore one can send the confirmation message without knowing K_{AB}. Depending on whether B can tell the difference between a single nonce and a combination of different nonces, B can be fooled into accepting the message he sent in the second step of the protocol as a confirmation.

We tried the authentication attack on the protocol specified with the theories T_{kH} (bounded homomorphism), $T_{H_\&}$ (multiset of keys with free operator), $T_{PGH_\&}$ (homomorphism over pre-group), $T_{APGH_{\&}}$ (homomorphism over Abelian pre-group), $T_{APGH_{\&}}$ (homomorphism over pre-Xor), and T_{2XORH} (homomorphism between two exclusive-or theories). We note that all the theories we obtained can be used for analyzing this attack. But Maude-NPA struggled to finish even the first step with the theory T_{2AGH} (homomorphism between two Abelian groups, variant complexity 2276) due to the high variant complexity, we therefore omit it here. The results are given in Table 4.6 for the protocol in which Bob cannot tell the difference between a single nonce and a combination of different nonces. In Table 4.7, we give the results for the protocol with a stronger Bob who can distinguish the difference between a single nonce and a combination of different nonces.

The attack pattern is specified in Maude-NPA as follows, which indicates that principal B, who is acting as the responder, finished his strand thinking that he agreed on a secret with A, but A did not actually participate in the communication.

```
eq ATTACK-STATE(0)
   = :: r, r' ::
     [ nil,  -(a ; Na),
             +(n(b, r) ; shk(h(n(b, r) * n(b, r') *  Na, mkey(a, b)),
               mkey(a, b)) ; shk(n(b, r') * Na, mkey(a, b))),
             -(shk(h(n(b,r) * Na', mkey(a, b)), mkey(a, b)) ;
                shk(Na', mkey(a, b))) | nil ]
     || empty
     || nil
     || nil
     || never(
     ( :: r1, r2 :: [nil | +(X), -(Y), +(Z), nil]
           & S:StrandSet || K:IntruderKnowledge)
     ( :: r1, r2 :: [nil | +(X), nil]
           & S:StrandSet  || K:IntruderKnowledge))
```

Table 4.6 shows the number of states generated by Maude-NPA and the amount of time taken for each step of the backwards reachability analysis. For each theory, except the theory T_{2XORH}, Maude-NPA found the authentication attack in two steps and terminated in six steps. For the protocol with the theory T_{2XORH}, Maude-NPA also found the authentication attack in two steps, but it was not able to terminate. As we can see from the tables, as the variant complexity of theories involved in the specification grows, the number of states and the time needed for Maude-NPA to find the attack also grows in general.

But we also noticed an exception with the theory $T_{PXorH_{\&}}$, which generates more states than the protocol using theory $T_{APGH_{\&}}$. Upon inspection, by the construction of the terms

Steps	T_{4H}		$T_{H_\&}$		$T_{PGH_\&}$	
	States	Time(ms)	States	Time(ms)	States	Time(ms)
Step 1	6	4225	3	1947	5	3174
Step 2	6	4613	4	2048	4	3497
Step 3	7	5825	6	2805	6	3097
Step 4	4	6866	8	6232	8	6938
Step 5	2	3190	4	5675	4	6096
Step 6	1	1692	1	3701	1	4032

Steps	$T_{APGH_\&}$		$T_{PXorH_\&}$		T_{2XorH}	
	States	Time(ms)	States	Time(ms)	States	Time(ms)
Step 1	5	2989	12	4640	14	10377
Step 2	4	3167	17	8767	15	18159
Step 3	6	2950	21	15340	19	22570
Step 4	8	6520	25	31086	23	42073
Step 5	4	5983	9	21238	13	35978
Step 6	1	3908	1	11924	15	31286
Step 7	1	0	1	0	20	101249
Step 8	1	0	1	0	...	timeout

Table 4.6: Results for authentication of A to B

in the protocol specification, the protocol using the theory $T_{PXorH_\&}$ tends to generate not only the states that are generated when using the theory $T_{APGH_\&}$, but also some twin states in which certain terms are unified. These twin states turn out to be unproductive states. By adding disequality constraints in the attack pattern, we were able to reduce half of the states while still terminating and found the stated authentication attack. This is also an example showing that the patterns in the protocol specification may also influence the performance.

For the second attack analysis, when B can tell whether he received a nonce or not, we verified that there is no authentication attack between A and B. Table 4.7 shows the number of states and attacks generated by Maude-NPA in each step for the attack state with different theories described above. For each theory Maude-NPA terminated within 4 steps. We note a similar relationship between performance and variant complexity as in Table 4.6.

Another interesting observation is that, if in the second step the message that B sends to A contains $h(N_s * N_A * N_B, K_{AB})$ instead, this attack can be prevented for theories in which $*$ is not commutative, even when B cannot tell the difference between a single nonce and a combination of different nonces; but this fix cannot prevent this attack for theories in which $*$ is commutative. The number of states generated by Maude-NPA and the amount of time taken for each step of the backwards reachability analysis are given in Table 4.8. For the protocols with theories T_{4H}, $T_{H_\&}$ and $T_{PGH_\&}$, Maude-NPA terminated in 5 steps without finding this authentication attack. For the protocols with theories $T_{APGH_\&}$ and $T_{PXorH_\&}$,

Steps	T_{4H}		$T_{H_\&}$		$T_{PGH_\&}$	
	States	Time(ms)	States	Time(ms)	States	Time(ms)
Step 1	1	1382	1	969	1	1043
Step 2	0	783	2	860	2	984
Step 3	0	0	1	1681	1	1826
Step 4	0	0	0	996	0	1077

Steps	$T_{APGH_\&}$		$T_{PXorH_\&}$		T_{2XorH}	
	States	Time(ms)	States	Time(ms)	States	Time(ms)
Step 1	1	984	4	2046	4	3771ms
Step 2	2	964	8	3015	6	5145
Step 3	1	1757	4	5204	2	8102
Step 4	0	1117	0	3317	0	3623

Table 4.7: Results for authentication of A to a stronger B

Maude-NPA found the authentication attack in 5 steps and terminated in 6 steps. For the protocols with the theory T_{2XorH}, Maude-NPA also found the authentication attack in 5 steps but did not terminate. We saw a similar relationship between performance and variant complexity as in Table 4.6.

Steps	T_{4H}		$T_{H_\&}$		$T_{PGH_\&}$	
	States	Time(ms)	States	Time(ms)	States	Time(ms)
Step 1	3	2747	2	1595	4	2811
Step 2	1	2118	3	1329	3	2721
Step 3	1	965	3	2496	3	2705
Step 4	0	906	1	2541	1	2750
Step 5	0	0	0	772	0	826

Steps	$T_{APGH_\&}$		$T_{PXorH_\&}$		T_{2XorH}	
	States	Time(ms)	States	Time(ms)	States	Time(ms)
Step 1	4	2962	8	3843	8	8354
Step 2	4	2788	18	6578	18	13805
Step 3	7	4177	22	17146	16	32024
Step 4	6	6802	18	23796	16	40088
Step 5	3	5208	5	15533	7	34454
Step 6	1	2331	1	5310	11	13037
Step 7	1	0	1	0	19	416986
Step 8	1	0	1	0	...	timeout

Table 4.8: Results for authentication of A to B

4.5 CONCLUDING REMARKS

The lack of FVP for H and AGH has made unification-based formal protocol analysis difficult to perform by extensible and generic methods such as variant-based unification. In this chapter we have addressed this problem by studying a hierarchy of theories for homomorphic encryption that are all FVP. The existence of finitary unification algorithms for these theories seems to be a new result in the area of unification theory. We have also introduced variant complexity as a metric and shown how it affects performance. One important lesson learned from our experiments in using these theories for protocol analysis is that there is a tradeoff between theory accuracy (typically at the cost of higher variant complexity) and efficiency of the analysis process. Our hierarchy allows users to choose the right balance for their problem to negotiate this tradeoff.

CHAPTER 5: STRAND SPACE WITH CHOICE VIA A PROCESS ALGEBRA SEMANTICS

5.1 INTRODUCTION

In the course of developing a specification language suitable for the formal analysis of cryptographic protocols, it has become clear that there are certain universal features that can best be handled by accounting for them directly in the syntax and semantics of the formal specification language, e.g., unguessable nonces, communication across a network controlled by an attacker, and support for the equational properties of cryptographic primitives. Thus a number of different specification languages have been developed that include these features.

At the same time, it is necessary to provide support for more commonly used constructs, such as *choice points* that cause the protocol to continue in different ways, and to do so in such a way that they are well integrated with the more specifically cryptographic features of the language. However, in their original form most of these languages do not support choice, or support it only in a limited way.

In particular the strand space model [25], one of the popular models designed for use in cryptographic protocol analysis, does not support choice in its original form; strands describe linear sequences of input and output messages, without any branching. One way of dealing with this limitation, and of formalizing strand spaces in general, has been to embed the strand space model in some other formal system that supports choice, e.g., event-based models for concurrency [52], Petri nets [53], or multi-set rewriting [54]. However, we believe that there is an advantage in introducing choice in the strand space model itself, while proving soundness and completeness with another formal system in order to validate the augmented model. This can allow us to concentrate on handling the different types of choice that commonly arise in cryptographic protocols.

5.1.1 Contributions

We address the problem of representing choice in the strand space model, particularly as it is used in the Maude-NPA cryptographic protocol analysis tool. We have identified a class of choices which includes finite branching and some cases of infinite branching. At the theoretical level, we provide a bisimulation result between the expected forwards execution semantics of the new process algebra and the original symbolic backwards semantics of

[1]This chapter is based on the paper [91], joint work with Santiago Escobar, Catherine A. Meadows, José Meseguer and Sonia Santiago.

Maude-NPA. This requires extra intermediate forwards and backwards semantics that are included in this chapter, together with all the proofs. What these results make possible is a sound and complete symbolic reachability analysis method for cryptographic protocols with choice *modulo* equational properties of the cryptographic functions satisfying the finite variant property (FVP). At the tool level, we have fully integrated the process algebra syntax, and its transformation into strands, and have developed new methods to specify attack states using the process notation in the recent release of Maude-NPA 3.0 (see [58]). Furthermore, we illustrate the expressive power and naturalness of adding choice to strand spaces with various examples, and show how it can be effectively used in formal analysis.

5.1.2 Choice in Maude-NPA

Previous to this work, Maude-NPA offered some ways of handling choice, but its scope was limited, and a uniform semantics of choice was lacking. Several kinds of branching could be handled by a protocol composition method in which a single parent strand is composed with one or more child strands. However, repurposing composition for branching has its limitations. First of all, it is possible to inadvertently introduce non-deterministic choice into what was intended to be deterministic choice by unwise choice of input and output parameters. Secondly, the limitation to pattern matching rules out certain types of deterministic choice conditioned on predicates that cannot be expressed this way, e.g., disequality predicates. Finally, implementation of choice via composition can be inefficient, since Maude-NPA must evaluate all possible child strands that match a parent strand.

Maude-NPA, in common with many other cryptographic protocol analysis tools, also offers a type of implicit choice that does not involve branching: non-deterministic choice of the values of certain variables. For example, a strand that describes an initiator communicating with a responder generally uses variables for both the initiator and responder names; this represents a non-deterministic choice of initiator and responder identities. However, the semantic implications of this kind of choice were not that well understood, which made it difficult to determine where it could safely be used. Clearly, a more unified treatment of choice was necessary, together with a formal semantics of choice.

In this work we have developed a taxonomy of choice features in which the categories of deterministic and non-deterministic choice are further subdivided. First of all, non-deterministic choice is subdivided into *explicit* and *implicit* non-deterministic choice. In explicit non-deterministic choice a role[2] chooses either one branch or another at a choice

[2]As further explained later, the behaviors of protocol participants, e.g., sender, receiver, server, etc., are described by their respective *roles*.

point non- deterministically. In implicit non-deterministic choice a logical *choice variable* is introduced which may be non-deterministically instantiated by the role. Deterministic choice is subdivided into (explicit) *if-then-else* choice and *implicit deterministic choice.* In if-then-else choice a predicate is evaluated. If the predicate evaluates to true one branch is chosen, and if it evaluates to false another branch is chosen. Deterministic choice with more than two choices can be modeled by nesting of if-then-else choices. In implicit deterministic choice, a term pattern is used as an implicit guard, so that only messages matching such pattern can be chosen, i.e., accepted, by the role. Although implicit deterministic choice can be considered a special case of if-then-else choice in which the second branch is empty, it is often simpler to treat it separately. Classifying choice in this way allows us to represent all possible behaviors of a protocol by a finite number of strands modeling possible executions, while still allowing the variables used in implicit non-deterministic and deterministic choice to be instantiated in a typically infinite number of ways.

5.1.3 A Motivating Example

In this section we introduce a protocol that we will use as a running example. It is a simplified version of the handshake protocol in TLS 1.3 [92], a proposed update to the TLS standard for client-server authentication. This protocol, like most other protocol standards, offers a number of different choices that are applied in different situations. In order to make the presentation and discussion manageable, we present only a subset here: the client chooses a Diffie-Hellman group, and proposes it to the server. The server can either accept it or request that the client proposes a different group. In addition, the server has the option of requesting that the client authenticates itself. We present the protocol at a high level similar to the style used in [92].

Example 5.1. We let a dashed arrow $\dashrightarrow$ denote an optional message, and an asterisk * denote an optional field.

1. $C \rightarrow S$: ClientHello, Key_Share
   The client sends a Hello message containing a nonce and the Diffie-Hellman group it wants to use. It also sends a Diffie-Hellman key share.

   - 1.1 $S \dashrightarrow C$: HelloRetryRequest
     The server may optionally reject the Diffie-Hellman group proposed by the client and request a new one.
   - 1.2 $C \dashrightarrow S$: DHGroup, Key_Share
     The client proposes a new group and sends a new key share.

2. $S \rightarrow C$: ServerHello, Key_Share,
   {AuthReq*},{CertificateVerify}, {Finished}
   The server sends its own Hello message and a Diffie-Hellman key share. It may optionally send an AuthReq to the client to authenticate itself with a public key signature from its public key certificate. It then signs the entire handshake using its own public key in the CertificateVerify field. Finally, in the Finished field it computes a MAC over the entire handshake using the shared Diffie-Hellman key. The {} notation denotes a field encrypted using the shared Diffie-Hellman key.

3. $C \rightarrow S$: {CertificateVerify*}, {Finished}
   If the client received an AuthReq from the server it returns its own CertificateVerify and Finished fields.

5.1.4 Plan of the chapter

The rest of this chapter is organized as follows. We first define the process algebra syntax and operational semantics in Section 5.2. In Section 5.3 we extend Maude-NPA's strand space syntax to include choice operators. The main bisimulation results between the expected forwards semantics of the process algebra in Section 5.2 and the original symbolic backwards strand semantics of Maude-NPA of Section 3 are Theorems 5.4 and 5.5. They are proved by introducing an intermediate semantics, a forward strand space semantics originally introduced in [60]. First, in Section 5.3 we extend the strand space model with constraints, since strands are the basis of both the forwards semantics and the backwards semantics of Maude-NPA. In Section 5.4 we augment the forwards strand space semantics of [60] with choice operators and operational semantic rules to produce a *constrained forwards semantics*. In Section 5.5 we prove bisimilarity of the process algebra semantics of Section 5.2 and the constrained forwards semantics of Section 5.4. In [60] the forwards strand space semantics was proved sound and complete w.r.t. the original symbolic backwards semantics of Maude-NPA and, therefore, such proofs had to be extended to handling constraints. In Section 5.6 we augment the original symbolic backwards semantics of Maude-NPA with choice operators and operational semantic rules to produce a *constrained backwards semantics*. In Section 5.7 we then prove that the constrained backwards semantics is sound and complete with respect to the constrained forwards semantics. By combining the bisimulation between the process algebra and the constrained forwards semantics on the one hand (Theorem 5.1) and the bisimulation between the constrained forwards semantics and the constrained backwards semantics on the other hand (Theorems 5.3 and 5.2) we obtain the main bisimulation results (Theorems 5.4 and 5.5). Finally, in Section 5.8 we describe how the process algebra has been fully integrated into Maude-NPA and show some experiments that we have run using Maude-NPA on various protocols exhibiting both deterministic and non-deterministic choice. In Section 5.9 we discuss related work. Finally, we conclude in Section 5.10.

5.2 A PROCESS ALGEBRA FOR PROTOCOLS WITH CHOICE

In this section we define a process algebra that extends the strand space model to naturally specify protocols exhibiting choice points. Throughout this chapter we refer to this process algebra as the *protocol process algebra*.

The rest of this section is organized as follows. First, in Section 5.2.1 we define the syntax of the protocol process algebra and state the requirements that a *well-formed process* must satisfy. Then in Section 5.2.2, we explain how *protocol specifications* can be defined in this process algebra. In Section 5.2.3 we then define the *operational semantics* of the protocol process algebra. Note that the operational semantics of Maude-NPA given in Section 3 corresponds to a symbolic *backwards* semantics, while in Section 5.2.3 we give a rewriting-based *forwards* semantics for process algebra. Sections 5.5 and 5.7 will relate these two semantics using a *bisimulation*.

5.2.1 Syntax of the Protocol Process Algebra

In the *protocol process algebra* the behaviors of both honest principals and the intruder are represented by *labeled processes*. Therefore, a protocol is specified as a set of labeled processes. Each process performs a sequence of actions, namely, sending or receiving a message, and may also perform deterministic or non-deterministic choices. The protocol process algebra's syntax is parameterized[3] by a sort Msg of messages and has the following syntax:

$$\begin{aligned}
ProcConf \ &::= LProc \mid ProcConf \ \& \ ProcConf \mid \varnothing \\
LProc \ &::= (Role, I, J) \ Proc \\
Proc \ &::= nilP \mid \ + Msg \mid \ - Msg \mid Proc \cdot Proc \mid \\
&\qquad Proc \ ? \ Proc \mid if \ Cond \ then \ Proc \ else \ Proc \\
Cond \ &::= Msg \ \neq \ Msg \mid Msg = \ Msg
\end{aligned}$$

[3]More precisely, as explained in Section 5.2.2, they are parameterized by a user-definable equational theory $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}})$ having a sort Msg of messages.

- *ProcConf* stands for a *process configuration*, that is, a set of labeled processes. The symbol & is used to denote set union for sets of labeled processes.
- *LProc* stands for a *labeled process*, that is, a process *Proc* with a label (Role, I, J). *Role* refers to the role of the process in the protocol (e.g., initiator or responder). I is a natural number denoting the identity of the process, which distinguishes different instances(sessions) of a process specification. J indicates that the action at stage J of the process specification will be the next one to be executed, that is, the first $J-1$ actions of the process for role *Role* have already been executed. Note that we omit I and J in the protocol specification when both I and J are 0.
- *Proc* defines the actions that can be executed within a process. $+\mathit{Msg}$, and $-\mathit{Msg}$ respectively denote sending out or receiving a message *Msg*. We assume a single channel, through which all messages are sent to or received by the intruder. "$\mathit{Proc} \cdot \mathit{Proc}$" denotes *sequential composition* of processes, where symbol $\_ . \_$ is associative and has the empty process *nilP* as identity. "*Proc* ? *Proc*" denotes an explicit *nondeterministic choice*, whereas "*if Cond then Proc else Proc*" denotes an explicit *deterministic choice*, whose continuation depends on the satisfaction of the constraint *Cond*.
- *Cond* denotes a constraint that will be evaluated in explicit deterministic choices. In this work we only consider constraints that are either equalities ($=$) or disequalities ($\neq$) between message expressions.

Let PS, QS, and RS be process configurations, and P, Q, and R be protocol processes. Our protocol syntax satisfies the following *structural axioms*:

$$PS \,\&\, QS = QS \,\&\, PS \qquad (5.1) \qquad\qquad PS \& \emptyset = PS \qquad (5.4)$$

$$(PS \,\&\, QS) \,\&\, RS = PS \,\&\, (QS \,\&\, RS) \qquad (5.2) \qquad\qquad P \cdot \mathit{nilP} = P \qquad (5.5)$$

$$(P \cdot Q) \cdot R = P \cdot (Q \cdot R) \qquad (5.3) \qquad\qquad \mathit{nilP} \cdot P = P \qquad (5.6)$$

The specification of the processes defining a protocol's behavior may contain some variables denoting information that the principal executing the process does not yet know, or that will be different in different executions. In all protocol specifications we assume three disjoint kinds of variables:

- ***fresh variables***: these are not really variables in the standard sense, but *names* for *constant values* in a data type $\mathsf{V}_{\mathsf{fresh}}$ of *unguessable* values such as nonces. A *fresh variable* r is always associated with a role $ro \in \mathit{Role}$ in the protocol. For each protocol session i, we associate to r a unique name $r.ro.i$ for a constant in the data type $\mathsf{V}_{\mathsf{fresh}}$.

What is assumed is that if $r.ro.i \neq r'.ro'.j$ (including the case $r.ro.i \neq r.ro.j$), the values interpreting $r.ro.i$ and $r'.ro'.j$ in $\mathsf{V}_{\mathsf{fresh}}$ are both *different* and *unguessable.* In particular, for role $ro \in Role$, the interpretation mapping $I : \{r.ro.i \mid i \in \mathbb{N}\} \to \mathsf{V}_{\mathsf{fresh}}$ is *injective* and *random.* In our semantics, a constant $r.ro.i$ denotes its (unguessable) interpretation $I(r.ro.i) \in \mathsf{V}_{\mathsf{fresh}}$. Throughout this chapter we will denote this kind of variables as $r, r_1, r_2, \ldots$.

- ***choice variables***: these are variables first appearing in a *sent message* $+M$, which can be substituted by any value arbitrarily chosen from a possibly infinite domain. A choice variable indicates an *implicit non-deterministic choice.* Given a protocol with choice variables, each possible substitution of these variables denotes a possible continuation of the protocol. We always denote choice variables by uppercase letters postfixed with the symbol "?" as a subscript, e.g., $A_?, B_?, \ldots$.

- ***pattern variables***: variables first appearing in a *received message* $-M$. These variables will be instantiated when matching sent and received messages. *Implicit deterministic choices* are indicated by terms containing pattern variables, since failing to match a pattern term may lead to the rejection of a message. A pattern term plays the implicit role of a guard, so that, depending on the different ways of matching it, the protocol can have different continuations. This kind of variables will be written with uppercase letters, e.g., $A, B, N_A, \ldots$.

Note that fresh variables are distinguished from other variables by having a specific sort Fresh. Choice variables and pattern variables can never have sort Fresh.

To guarantee the requirements on different kinds of variables that can appear in a given process, we consider only *well-formed* processes. We make this notion precise by defining a function $wf : Proc \to Bool$ checking whether a given process is well-formed. A labeled process is *well-formed* if the process it labels is well-formed. A process configuration is *well-formed* if all the labeled process in it are well-formed. The definition of wf uses an auxiliary function $shVar : Proc \to VarSet$, retrieving the "shared variables" of a process, i.e., the set of variables that show up in all branches. Below we define both functions, where P, Q, and R are processes, M is a message, and T is a constraint.

$$
\begin{aligned}
&shVar(+M \ \cdot P) = Var(M) \cup shVar(P)\\
&shVar(-M \ \cdot P) = Var(M) \cup shVar(P)\\
&shVar((if\, T \ then \ P \ else \ Q) \ \cdot R) = Var(T) \cup (shVar(P) \cap shVar(Q)) \cup shVar(R)\\
&shVar((P \ ? \ Q) \ \cdot R) = (shVar(P) \cap shVar(Q)) \cup shVar(R)
\end{aligned}
$$

$$\begin{aligned}
&\mathit{shVar}(\mathit{nilP}) = \varnothing\\
&\mathit{wf}(P \cdot +M) = \mathit{wf}(P) \qquad \mathit{if}\ \ (\mathit{Var}(M) \cap \mathit{Var}(P)) \subseteq \mathit{shVar}(P)\\
&\mathit{wf}(P \cdot -M) = \mathit{wf}(P) \qquad \mathit{if}\ \ (\mathit{Var}(M) \cap \mathit{Var}(P)) \subseteq \mathit{shVar}(P)\\
&\mathit{wf}(P \cdot (\mathit{if}\ \ T\ \mathit{then}\ Q\ \mathit{else}\ R)) = \mathit{wf}(P \cdot Q) \wedge \mathit{wf}(P \cdot R)\\
&\qquad\qquad \mathit{if}\ P \neq \mathit{nilP}\ \mathit{and}\ \ Q \neq \mathit{nilP}\ \mathit{and}\ \ \mathit{Var}(T) \subseteq \mathit{shVar}(P)\\
&\mathit{wf}(P \cdot (Q\ ?\ R)) = \mathit{wf}(P \cdot Q) \wedge \mathit{wf}(P \cdot R) \qquad \mathit{if}\ Q \neq \mathit{nilP}\ \mathit{or} R \neq \mathit{nilP}\\
&\mathit{wf}(P \cdot\ \mathit{nilP}) = \mathit{wf}(P)\\
&\mathit{wf}(\mathit{nilP}) = \mathit{True}.
\end{aligned}$$

Remark 5.1. Note that the well-formedness property implies that if a process begins with a deterministic choice action *if T then Q else R*, then all variables in T must be instantiated, and thus only one branch may be taken. For this reason, it is undesirable to specify processes that begin with such an action. Furthermore, note that the well-formedness property avoids explicit choices where both possibilities are the *nilP* process. That is, processes containing either *(if T then nil else nil)*, or *(nil ? nil)*, respectively.

We illustrate the notion of well-formed process below.

Example 5.2. The behavior of a Client initiating an instance of the handshake protocol from Example 5.1 with the Server, where the Server may or may not request the Client to authenticate itself, may be specified by the well-formed process shown below:

$$\begin{aligned}
(\mathit{Client})\ \ & + (\mathit{hs}; n(C_?, r_1); G_?; \mathit{gen}(G_?); \mathit{keyG}(G_?, C_?, r_2)) \cdot\\
& - (\mathit{hs}; N; G_?; \mathit{gen}(G_?); E; Z(\mathit{AReq}, G_?, E, C_?, r_1, S, HM)) \cdot\\
& \mathit{if}\ \ (\mathit{AReq} = \mathit{authreq})\\
& \mathit{then}\\
& + (e(\mathit{keyE}(G_?, E, C_?, r_1),\\
& \qquad \mathit{sig}(C, W(HM, \mathit{AReq}, S_?, G_?, E, C_?, r_1));\\
& \qquad \mathit{mac}(\mathit{keyE}(G_?, E, , C_?, r_1), W(HM, \mathit{AReq}, S, G_?, E, C_?, r_1)))) \cdot\\
& \mathit{else}\\
& + (e(\mathit{keyE}(G_?, E, C_?, r_2),\\
& \qquad \mathit{mac}(\mathit{keyE}(G_?, E, C_?, r_2), W(HM, \mathit{AReq}, S, G_?, E, C_?, r_1))))
\end{aligned}$$

where $KeyG$, Z and W are macros used to construct messages sent in the protocol. The variables $C_?$ and $G_?$ are choice variables denoting the client and Diffie-Hellman group respectively, and the variables r_1 and r_2 are fresh variables. All other variables are pattern variables. In particular, the variable $AReq$ is a pattern variable that can be instantiated to either *authreq* or *noauthreq*. The Client makes a deterministic choice whether or not to sign its next message with its digital signature, depending on which value of *AReq* it receives.

Example 5.3. The behavior of a Server who may or may not request a retry from a Client in an instance of the handshake protocol from Example 5.1 may be specified as follows:

$$\begin{aligned}(Server): \; & -(hs; N; G; gen(G); E)\cdot\\ & (((+(hs; retry)\cdot\\ & \quad -(hs; N'; G'; gen(G'); E')\cdot\\ & \quad +(hs; n(S_?, r1); G'; gen(G'); keyG(G', S_?, r_2);\\ & \qquad Z(AReq_?, G', E', S, r_2, S_?, HM)))\\ & ?\\ & (+(hs; n(S_?, r1); G; gen(G); keyG(G, S, r_2);\\ & \qquad Z(AReq_?, G, E, S, r_2, S_?, HM)))))\end{aligned}$$

In this case the server nondeterministically chooses to request or not to request a retry. In the case of a retry it waits for the retry message from the client, and then proceeds with the handshake message using the new key information from the client. In the case when it does not request a retry, it sends the handshake message immediately after receiving the client's Hello message. The variable r_2 is a fresh variable, while $S_?$ and $AReq_?$ are choice variables. $S_?$ denotes the name of the server, and $AReq_?$ is nondeterministically instantiated to *authreq* or *noauthreq*.

Example 5.4. The following process does not satisfy the well-formedness property.

$$\begin{aligned}(Resp) & -(pk(B, A; NA))\cdot\\ & (+(pk(A, 1; n(B, r))) \; ? \; +(pk(A, 2)))\;\cdot\\ & +(pk(C_?, n(B, r)))\end{aligned}$$

The problem with this process is the fresh variable r appearing in message $+(pk(C_?, n(B, r)))$, since

$$r \notin shVar(-(pk(B, A; NA)) \cdot (+(pk(A, 1; n(B, r))) \; ? \; +(pk(A, 2))))$$

more specifically, because it does not appear in message $+(pk(A, 2))$, but $r \in Var(-(pk(B, A; NA)) \cdot (+(pk(A, 1; n(B, r))) \; ? \; +(pk(A, 2))))$.

5.2.2 Protocol Specification in Process Algebra

Given a protocol $\mathcal{P}$, we define its specification in the protocol process algebra, written $\mathcal{P}_{PA}$, as a pair of the form $\mathcal{P}_{PA} = ((\Sigma_{PA_{\mathcal{P}}}, E_{PA_{\mathcal{P}}}), P_{PA})$, where $(\Sigma_{PA_{\mathcal{P}}}, E_{PA_{\mathcal{P}}})$ is an equational theory explained below, and P_{PA} is a term denoting a *well-formed* process configuration representing the behavior of the honest principals as well as the capabilities of the attacker. That is, $P_{PA} = (ro_1)P_1 \ \& \ \ldots \ \& \ (ro_i)P_i$, where each ro_k, $1 \leqslant k \leqslant i$, is either the role of an honest principal or identifies one of the capabilities of the attacker. P_{PA} cannot contain two processes with the same label, i.e., the behavior of each honest principal, and each attacker capability are represented by a *unique* process. $E_{PA_{\mathcal{P}}} = E_{\mathcal{P}} \cup E_{PA}$ is a set of equations with $E_{\mathcal{P}}$ denoting the protocol's cryptographic properties and E_{PA} denoting the properties of process constructors. The set of equations $E_{\mathcal{P}}$ is user-definable and can vary for different protocols. Instead, the set of equations E_{PA} is always the same for all protocols. $\Sigma_{PA_{\mathcal{P}}} = \Sigma_{\mathcal{P}} \cup \Sigma_{PA}$ is the signature defining the sorts and function symbols as follows:

- $\Sigma_{\mathcal{P}}$ is an order-sorted signature defining the sorts and function symbols for the messages that can be exchanged in protocol $\mathcal{P}$. However, independently of protocol $\mathcal{P}$, $\Sigma_{\mathcal{P}}$ must always have a sort Msg as the top sort in one of its connected components. We call a sort S a *data sort* iff it is either a subsort of Msg, or there is a message constructor $c : \mathsf{S_1 ... S ... S_n} \to \mathsf{S'}$, with $\mathsf{S'}$ a subsort of Msg. The specific sort Fresh for fresh variables is an example of a *data sort*. Choice and pattern variables have sort Msg or any of its subsorts.

- Σ_{PA} is an order-sorted signature defining the sorts and function symbols of the *process algebra infrastructure*. Σ_{PA} corresponds exactly to the BNF definition of the protocol process algebra's syntax in Section 5.2.1. Although it has a sort Msg for messages, it leaves this sort totally unspecified, so that different protocols $\mathcal{P}$ may use completely different message constructors and may satisfy different equational properties $E_{\mathcal{P}}$. Therefore, Σ_{PA} will be the same signature for any protocol specified in the process algebra. More specifically, Σ_{PA} contains the sorts for process configurations ($\mathsf{ProcConf}$), labeled processes (LProc), processes (Proc), constraints (Cond), and messages(Msg), as well as the subsort relations $\mathsf{LProc} < \mathsf{ProcConf}$. Furthermore, the function symbols in Σ_{PA} are also defined according to the BNF definition.

Therefore, the syntax $\Sigma_{PA_{\mathcal{P}}}$ of processes for $\mathcal{P}$ will be in the union signature $\Sigma_{PA} \cup \Sigma_{\mathcal{P}}$, consisting of the protocol-specific syntax $\Sigma_{\mathcal{P}}$, and the generic process syntax Σ_{PA} through the shared sort Msg.

5.2.3 Operational Semantics of the Protocol Process Algebra

Given a protocol $\mathcal{P}$, a *state* of $\mathcal{P}$ consists of a set of (possibly partially executed) *labeled processes*, and a set of terms in the intruder's knowledge $\{IK\}$. That is, a state is a term of the form $\{LP_1 \,\&\, \cdots \,\&\, LP_n \mid \{IK\}\}$. Given a state St of this form, we abuse notation and write $LP_k \in St$ if LP_k is a labeled process in the set $LP_1 \,\&\, \cdots \,\&\, LP_n$.

The intruder knowledge IK models the *single* channel through which all messages are sent and received. We consider an active attacker who has complete control of the channel, i.e, can read, alter, redirect, and delete traffic as well as create its own messages by means of *intruder processes*. That is, the purpose of some $LP_k \in St$ is to perform message-manipulation actions for the intruder.

State changes are defined by a set $R_{PA_{\mathcal{P}}}$ of *rewrite rules*, such that the rewrite theory $(\Sigma_{PA_{\mathcal{P}}+State}, E_{PA_{\mathcal{P}}}, R_{PA_{\mathcal{P}}})$ characterizes the behavior of protocol $\mathcal{P}$, where $\Sigma_{PA_{\mathcal{P}}+State}$ extends $\Sigma_{PA_{\mathcal{P}}}$ by adding state constructor symbols. We assume that a protocol's execution begins with an empty state, i.e., a state with an empty set of labeled processes, and an empty intruder knowledge. That is, the initial state is always of the form $\{\emptyset \mid \{empty\}\}$. Each transition rule in $R_{PA_{\mathcal{P}}}$ is labeled with a tuple of the form (ro, i, j, a, n), where:

- ro is the role of the labeled process being executed in the transition.
- i denotes the identifier of the labeled process being executed in the transition. Since there can be more than one process instance of the same role in a process state, i is used to distinguish different instances, i.e., ro and i together uniquely identify a process in a state.
- j denotes the process' step number since its beginning.
- a is a ground term identifying the action that is being performed in the transition. It has different possible values: "$+m$" or "$-m$" if the message m was sent (and added to the intruder's knowledge) or received, respectively; "m" if the message m was sent but did not increase the intruder's knowledge, "?" if the transition performs an explicit non-deterministic choice, or "T" if the transition performs a explicit deterministic choice.
- n is a number that, if the action that is being executed is an explicit choice, indicates which branch has been chosen as the process continuation. In this case n takes the value of either 1 or 2. If the transition does not perform any explicit choice, then $n = 0$.

Below we describe the set of transition rules that define a protocol's execution in the protocol process algebra, that is, the set of rules $R_{PA_{\mathcal{P}}}$. Note that in the transition rules shown below, PS denotes the rest of labeled processes of the state (which can be the empty set $\varnothing$).

- The action of *sending a message* is represented by the two transition rules below. Since we assume that the intruder has complete control of the network, it can learn any message sent by other principals. Rule (PA++) denotes the case in which the sent message is added to the intruder's knowledge. Note that this rule can only be applied if the intruder has not already learnt that message. Rule (PA+) denotes the case in which the intruder chooses not to learn the message, i.e., the intruder's knowledge is not modified, and, thus, no condition needs to be checked. Since choice variables denote messages that are nondeterministically chosen, all (possibly infinitely many) admissible ground substitutions for the choice variables are possible behaviors.

$$
\begin{array}{ll}
\{(ro,i,j)\ (+M\cdot P)\ \&\ PS \mid \{IK\}\} & \\
\longrightarrow_{(ro,i,j,+M\sigma,0)} \{(ro,i,j+1)\ P\sigma\ \&\ PS \mid \{M\sigma{\in}\mathcal{I}, IK\}\} & \\
\textit{if } (M\sigma{\in}\mathcal{I}) \notin IK & \\
\textit{where } \sigma \textit{ is a ground substitution binding choice variables in } M & \text{(PA++)}
\end{array}
$$

$$
\begin{array}{ll}
\{(ro,i,j)\ (+M\cdot P)\ \&\ PS \mid \{IK\}\} & \\
\longrightarrow_{(ro,i,j,M\sigma,0)} \{(ro,i,j+1)\ P\sigma\ \&\ PS \mid \{IK\}\} & \\
\textit{where } \sigma \textit{ is a ground substitution binding choice variables in } M & \text{(PA+)}
\end{array}
$$

- As shown in the rule below, a process can *receive a message* matching a pattern M if there is a message M' in the intruder's knowledge, i.e., a message previously sent either by some honest principal or by some intruder process, that matches the pattern message M. After receiving this message the process will continue with its variables instantiated by the matching substitution, which takes place modulo the equations $E_{\mathcal{P}}$. Note that the intruder can "delete" a message via choosing not to learn it (executing Rule PA+ instead of Rule PA++) or not to deliver it (failing to execute Rule PA-).

$$
\begin{array}{ll}
\{(ro,i,j)\ (-M\cdot P)\ \&\ PS \mid \{M'{\in}\mathcal{I}, IK\}\} & \\
\longrightarrow_{(ro,i,j,-M\sigma,0)} \{(ro,i,j+1)\ P\sigma\ \&\ PS \mid \{M'{\in}\mathcal{I}, IK\}\} & \\
\textit{if } M' =_{E_{\mathcal{P}}} M\sigma & \text{(PA-)}
\end{array}
$$

- The two transition rules shown below define the operational semantics of *explicit deterministic choices*. That is, the operational semantics of an *if* T *then* P *else* Q expression. More specifically, rule (PAif1) describes the *then* case, i.e., if the constraint T is satisfied, the process will continue as P. Rule (PAif2) describes the *else* case, that is, if the constraint T is *not* satisfied, the process will continue as Q. Note that, since we only consider well-formed processes, these transition rules will only be applied if $j \geqslant 1$. Note also that since T has been fully substituted by the time the if-then-else is executed, and the constraints that we considered in this chapter are of the form $m \neq_{E_{\mathcal{P}}} m'$ or $m =_{E_{\mathcal{P}}} m'$, the satisfiability of T can be checked by checking whether the corresponding ground equality or disequality holds.

$$\begin{aligned}&\{(ro,i,j)\ ((\mathit{if}\,T\ \mathit{then}\ P\ \mathit{else}\ Q)\ \cdot R)\ \&\ PS \mid \{IK\}\}\\ &\longrightarrow_{(ro,i,j,T,1)} \{(ro,i,j+1)\ (P\cdot R)\ \&\ PS \mid \{IK\}\}\ \ if\ T \end{aligned} \quad \text{(PAif1)}$$

$$\begin{aligned}&\{(ro,i,j)\ ((\mathit{if}\,T\ \mathit{then}\ P\ \mathit{else}\ Q)\ \cdot R)\ \&\ PS \mid \{IK\}\}\\ &\longrightarrow_{(ro,i,j,T,2)} \{(ro,i,j+1)\ (Q\cdot R)\ \&\ PS \mid \{IK\}\}\ \ if\ \neg T \end{aligned} \quad \text{(PAif2)}$$

- The two transition rules below define the semantics of *explicit non-deterministic choice* $P\ ?\ Q$. In this case, the process can continue either as P, denoted by rule (PA?1), or as Q, denoted by rule (PA?2). Note that this decision is made non-deterministically.

$$\begin{aligned}&\{(ro,i,j)\ ((P\ ?\ Q)\cdot R)\ \&\ PS \mid \{IK\}\}\\ &\longrightarrow_{(ro,i,j,?,1)} \{(ro,i,j+1)\ (P\cdot R)\ \&\ PS \mid \{IK\}\}\end{aligned} \quad \text{(PA?1)}$$

$$\begin{aligned}&\{(ro,i,j)\ ((P\ ?\ Q)\cdot R)\ \&\ PS \mid \{IK\}\}\\ &\longrightarrow_{(ro,i,j,?,2)} \{(ro,i,j+1)(Q\cdot R)\ \&\ PS \mid \{IK\}\}\end{aligned} \quad \text{(PA?2)}$$

- The transition rules shown below describe the *introduction of a new process* from the specification into the state, which allows us to support an unbounded session model. Recall that fresh variables are associated with a role and an identifier. Therefore, whenever a new process is introduced: (a) the largest process identifier (i) will be increased by 1, and (b) new names will be assigned to the fresh variables in the new process. The function $\mathit{MaxProcId}(PS, ro)$ in the transition rule below is used to get the largest process identifier (i) of role ro in the process configuration PS. The substitution $\rho_{ro,i+1}$ in the transition rule below takes a labeled process and assigns new names to the fresh variables according to the label. More specifically, $(ro, i+1, 1)\ P_k(r_1, \ldots, r_n)\rho_{ro,i+1} = (ro, i+1, 1)\ P_k(r_1, \ldots, r_n)\{r_1 \mapsto r_1.ro.i+1, \ldots, r_n \mapsto r_n.ro.i+1\}$. In a process state, a role name together with an identifier uniquely identifies a process. Therefore, there is a unique subset of fresh names for each process in the state. In the rest of this chapter we will refer to this kind of substitutions as *fresh substitutions*.

$$\left\{\begin{array}{l}
\forall\ (ro)\ P_k \in P_{PA} \\
\{PS \mid \{IK\}\} \\
\longrightarrow_{(ro,i+1,1,A,Num)} \{(ro, i+1, 2)\ P'_k\ \&\ PS \mid \{IK'\}\} \\
\textsf{IF}\ \{(ro, i+1, 1)\ P_k\rho_{ro,i+1} \mid \{IK\}\} \\
\longrightarrow_{(ro,i+1,1,A,Num)} \{(ro, i+1, 2)\ P'_k\ \mid \{IK'\}\} \\
\textit{where } \rho_{ro,i+1} \textit{ is a fresh substitution,} \\
\ i = \mathit{MaxProcId}(PS, ro)
\end{array}\right\} \tag{PA\&}$$

Note that A denotes the action of the state transition, and can be of any of the forms explained above. The function *MaxProcId* is defined as follows:

$$\begin{array}{ll}
\mathit{MaxProcId}(\emptyset, ro) = 0 & \\
\mathit{MaxProcId}((ro, i, j)P\&PS, ro) = \mathit{max}(\mathit{MaxProcId}(PS, ro), i) & \\
\mathit{MaxProcId}((ro', i, j)P\&PS, ro) = \mathit{MaxProcId}(PS, ro) & \textit{if } ro \neq ro'
\end{array}$$

where PS denotes a process configuration, P denotes a process, and ro, ro' denote role names.

Therefore, the behavior of a protocol in the process algebra is defined by the set of transition rules $R_{PA_\mathcal{P}} = \{(\text{PA++}), (\text{PA+}), (\text{PA-}), (\text{PAif1}), (\text{PAif2}), (\text{PA?1}), (\text{PA?2})\} \cup (\text{PA\&})$.

Our main result is a bisimulation between the state space generated by the transition rules $R^{-1}_{B_\mathcal{P}}$, associated to the symbolic backwards semantics of Section 3, and the transition rules $R_{PA_\mathcal{P}}$ above, associated to the forwards semantics for process algebra. This is nontrivial, since there are three major ways in which the two semantics differ. The first is that processes "forget" their past, while strands "remember" theirs. The second is that Maude-NPA uses backwards search, while the process algebra proceeds forwards. The third is that Maude-NPA performs symbolic reachability analysis using terms with variables, while the process algebra considers only ground terms.

We systematically relate these different semantics by introducing an intermediate semantics, a forward strand space semantics extending that in [60]. First, in Section 5.3 we extend the strand space model with constraints, since strands are the basis of both the forwards semantics and the backwards semantics of Maude-NPA. In Section 5.4 we augment the forwards strand space semantics of [60] with choice operators and operational semantic rules to produce a *constrained forwards semantics*. In Section 5.5 we prove bisimilarity of the process algebra semantics of Section 5.2 and the constrained forwards semantics of Section 5.4. In [60] the forwards strand space semantics was proved sound and complete w.r.t. the original symbolic backwards semantics of Maude-NPA. But now such proofs have to be extended to handle constraints. In Section 5.6 we also augment the original symbolic backwards semantics of Maude-NPA with choice operators and operational semantic rules to produce a *constrained backwards semantics*. In Section 5.7, we then prove that the constrained backwards semantics is sound and complete with respect to the constrained forwards semantics. By combining the bisimulation between the process algebra and the constrained forwards semantics on the one hand, and the bisimulation between the constrained forwards semantics and the constrained backwards semantics on the other hand, we obtain the main bisimulation result.

Besides provgiving a detailed semantic account of how the strand model can be extended with choice features, the key practical importance of these bisimulation results is that, with the relatively modest extensions to Maude-NPA described in Section 5.8.1 and supported by its recent 3.0 release, sound and complete analysis of protocols with choice features specified in process algebra is made possible.

5.3 CONSTRAINED PROTOCOL STRANDS WITH CHOICE

To specify and analyze protocols with choices in Maude-NPA, in this section we extend Maude-NPA's strand notation by adding new symbols to support explicit choices. We refer to the strands in this extended syntax as *constrained protocol strands*.

In Section 5.3.1 we describe the syntax for constrained protocol strands. Then, in Section 5.3.2 we define a mapping from a protocol specification in the protocol process algebra, as described in Section 5.2.2, to a specification based on constrained protocol strands.

5.3.1 Constrained Protocol Strands Syntax

In this section we extend Maude-NPA's syntax by adding *constrained messages*, which are terms of the form $\{Cstr, Num\}$, where $Cstr$ is a constraint, and Num is a natural number that identifies the continuation of the protocol's execution, among the two possibilities after an explicit choice point. More specifically, we extend the signature of strand set in the Maude-NPA's syntax as follows:

- A new sort Cstr represents the constraints allowed in constrained messages, containing three symbols: (i) $? \ : \to \mathsf{Cstr}$, (ii) $\_{=}\_ \ : \ \mathsf{Msg}\ \mathsf{Msg} \to \mathsf{Cstr}$, and (iii) $\_{\neq}\_ \ : \ \mathsf{Msg}\ \mathsf{Msg} \to \mathsf{Cstr}$.
- A new sort $\mathsf{CstrMsg}$ for constrained messages, such that $\mathsf{CstrMsg} < \mathsf{SMsg}$, where SMsg is an existing Maude-NPA sort denoting signed messages (i.e., messages with + or -). Therefore, now a strand is a sequence of output, input and constrained messages.
- A new operator $\{\_,\_\} : \mathsf{Cstr}\ \mathsf{Nat} \to \mathsf{CstrMsg}$ constructs constrained messages.

We refer to this extended signature as $\Sigma_{CstrSS_{\mathcal{P}}}$. Note that the protocol signature $\Sigma_{\mathcal{P}}$ is contained in Maude-NPA's signature $\Sigma_{SS_{\mathcal{P}}}$, and therefore in $\Sigma_{CstrSS_{\mathcal{P}}}$. Furthermore, in the constrained semantics we allow each honest principal or intruder capability strand to be *labeled* by the "role" of that strand in the protocol (e.g., `(Client)` or `(Server)`). Therefore, strands are now terms of the form $(ro, i)[u_1, \ldots, u_n]$, where ro denotes the role of the strand in the protocol, i is a unique identifier distinguishing different instances of strands of the same role, and each u_i can be a sent or received message, i.e., a term of the form $M^{\pm}$, or a constraint message of the form $\{Cstr, Num\}$. We often omit i, or both ro and i for clarity when they are not relevant.

5.3.2 Protocol Specification using Constrained Protocol Strands

The behavior of a protocol involving choices can be specified using the syntax presented in Section 5.3.1 as described below.

Definition 5.1 (Constrained protocol strand specification)**.** Given a protocol $\mathcal{P}$, we define its specification by means of constrained protocol strands, written $\mathcal{P}_{CstrSS}$, as a tuple of the form $\mathcal{P}_{CstrSS} = ((\Sigma_{CstrSS_{\mathcal{P}}}, E_{SS_{\mathcal{P}}}), P_{CstrSS})$, where $\Sigma_{CstrSS_{\mathcal{P}}}$ is the protocol's signature (see Section 5.3.1), and $E_{SS_{\mathcal{P}}} = E_{\mathcal{P}} \cup E_{SS}$ is a set of equations as we defined in Chapter 3, where $E_{\mathcal{P}}$ denotes the protocol's cryptographic properties and E_{SS} denotes the protocol-independent properties of constructors of strands. That is, the set of equations $E_{\mathcal{P}}$ may vary depending on different protocols, but the set of equations E_{SS} is always the same for all protocols. P_{CstrSS} is a set of constrained protocol strands as defined in Section 5.3.1, representing the behavior of the honest principals as well as the capabilities of the attacker. That is, P_{CstrSS} is a set of labeled strands of the form: $P_{\mathit{CstrSS}} = \{(\mathit{ro_1})[u_{1,1}, \ldots, u_{1,n_1}]\ \&\ \ldots\ \&\ (\mathit{ro_m})[u_{m,1}, \ldots, u_{m,n_m}]\}$, where, for each $\mathit{ro_k}$ such that $1 \leqslant k \leqslant i$, $\mathit{ro_k}$ is either the role of an honest principal, or identifies one of the capabilities of the attacker. We note that P_{CstrSS} *may contain several strands with the same label*, each defining one of the possible paths of such a principal.

The protocol specification described above can be obtained by *transforming* a specification in the process algebra of Section 5.2.2 as follows. Given a protocol $\mathcal{P}$, its specification in the process algebra P_{PA}, consists of a set of *well-formed* labeled processes. We transform a term denoting a set of labeled processes into a term denoting a set of constrained protocol strands by the mapping *toCstrSS*. The intuitive idea is that, since our process contains no recursion, each process can be "deconstructed" as a set of constrained protocol strands, where each such strand represent a possible execution path of the process.

The mapping *toCstrSS* is specified in Definition 5.2 below.

Definition 5.2 (Mapping labeled processes *toCstrSS*)**.** Given a labeled process LP and a process configuration LPS, we define the mapping $\mathit{toCstrSS} : \mathcal{T}_{\Sigma_{PA_\mathcal{P}}}(\mathcal{X}) \ \rightarrow \mathcal{T}_{\Sigma_{\mathit{CstrSS}_\mathcal{P}}}(\mathcal{X})$ recursively as follows:

$$\begin{aligned}
&\mathit{toCstrSS}(LP\ \&\ LPS) = \mathit{toCstrSS^*}(LP, \mathit{nilP})\ \&\ \mathit{toCstrSS}(LPS)\\
&\mathit{toCstrSS}(\varnothing) = \varnothing
\end{aligned}$$

where $\varnothing$ is the empty set of strands. *toCstrSS** is an auxiliary mapping that maps a term denoting a labeled process to a term that denotes a set of constrained protocol strands. It takes two arguments: a labeled process, and a temporary store that keeps a sequence of messages. More specifically, $\mathit{toCstrSS^*} : \mathcal{T}_{\Sigma_{PA_\mathcal{P}}}(\mathcal{X}) \ \times \mathcal{T}_{\Sigma_{\mathit{CstrSS}_\mathcal{P}}}(\mathcal{X}) \rightarrow \mathcal{T}_{\Sigma_{\mathit{CstrSS}_\mathcal{P}}}(\mathcal{X})$ is defined as follows:

$$\begin{aligned}
&\mathit{toCstrSS^*}((ro, i, j)\ \mathit{nilP}, L) = (ro, i)\ [\ L\ ]\\
&\mathit{toCstrSS^*}((ro, i, j)\ +M\ .\ P, L) = \mathit{toCstrSS^*}((ro, i, j)\ P, (L,\ +M))\\
&\mathit{toCstrSS^*}((ro, i, j)\ -M\ .\ P,\ L) = \mathit{toCstrSS^*}((ro, i, j)\ P, (L,\ -M))\\
&\mathit{toCstrSS^*}((ro, i, j)\ \ (\mathit{if}\ T\ \mathit{then}\ P\ \mathit{else}\ Q)\ .\ R, L)\\
&\qquad =\ \mathit{toCstrSS^*}((ro, i, j)\ P\ .\ R, (L,\ \{T, 1\}))\ \&
\end{aligned}$$

$$\begin{aligned}
&\qquad\quad \mathit{toCstrSS^*}((ro,i,j)\ Q\ .\ R,(L,\ \{\neg T,2\}))\\
&\mathit{toCstrSS^*}((ro,i,j)\ (P\ ?\ Q)\ .\ R,L)\\
&\qquad = \ \mathit{toCstrSS^*}((ro,i,j)\ P\ .\ R,(L,\ \{?,1\}))\ \&\\
&\qquad\quad \mathit{toCstrSS^*}((ro,i,j)\ Q\ .\ R,(L,\{?,2\}))
\end{aligned}$$

where P, Q, and R denote processes, M is a message, T is a constraint, and L denotes a list of messages, i.e., input, output or constraint messages.

Note that *toCstrSS* does not modify output and input messages, since messages are actually terms in $\mathcal{T}_{\Sigma_\mathcal{P}}(\mathcal{X})$ in both the protocol process algebra, and the constrained forwards semantics. *toCstrSS* can be used both as a map between specifications, and as a map from process configurations and strand sets appearing in states.

We illustrate the *toCstrSS* transformation with the example below.

Example 5.5. If we apply the mapping *toCstrSS* to the process in Example 5.3 we obtain the following term which denotes a set of strands:

$$\begin{aligned}
&(\mathit{Server})\ [\ \{?,1\},\\
&\qquad -(hs;N;G;gen(G);E),\\
&\qquad +(hs;retry),\\
&\qquad -(hs;N';G';gen(G');E')),\\
&\qquad +(hs;n(S_?,r1);G';gen(G');keyG(G',S_?,r_2);\\
&\qquad\qquad Z(AReq_?,G',E',S,r_2,S_?,HM))]\ \&\\
&(\mathit{Server})\ [\ \{?,2\},\\
&\qquad -(hs;N;G;gen(G);E),\\
&\qquad +(hs;n(S_?,r1);G;gen(G);keyG(G,S,r_2);\\
&\qquad\qquad Z(AReq_?,G,E,S,r_2,S_?,HM))]
\end{aligned}$$

A protocol specification in the protocol process algebra can then be transformed into a specification of that protocol in the constrained protocol strands described below using *toCstrSS*.

Definition 5.3 (Specification transformation)**.** Given a protocol $\mathcal{P}$ and its protocol process algebra specification $\mathcal{P}_{PA} = ((\Sigma_{PA_\mathcal{P}}, E_\mathcal{P} \cup E_{PA}), P_{PA})$, with $P_{PA} = (ro_1)P_1\& \ldots \&(ro_n)P_n$, its specification by means of constrained protocol strands is $\mathcal{P}_{CstrSS} = ((\Sigma_{CstrSS_\mathcal{P}}, E_\mathcal{P} \cup E_{SS})$, $P_{CstrSS})$ with $P_{CstrSS} = \mathit{toCstrSS}(P_{PA})$.

5.4 CONSTRAINED FORWARDS STRAND SEMANTICS

In this section we extend Maude-NPA's rewriting-based forwards semantics in [60] by adding new transition rules for constrained messages. We refer to this extended forwards semantics as *constrained forwards strand semantics*. We show that the process algebra semantics and the constrained forwards strand semantics are label bisimilar. Therefore, protocols exhibiting choices can be specified and executed in an equivalent way in both semantics.

In the constrained forwards strand semantics, state changes are defined by a set $R_{CstrF_{\mathcal{P}}}$ of *rewrite rules*, so that the rewrite theory $(\Sigma_{CstrSS_{\mathcal{P}}}, E_{SS_{\mathcal{P}}}, R_{CstrF_{\mathcal{P}}})$ characterizes the behaviors of protocol $\mathcal{P}$.

The set of transition rules $R_{CstrF_{\mathcal{P}}}$ is an extension of the transition rules $R_{F_{\mathcal{P}}}$ in [60]. The transition rules are generated from the protocol specification. A *state* consists of a multiset of partially executed strands and a set of terms denoting the intruder's knowledge. The main differences between the sets $R_{CstrF_{\mathcal{P}}}$ and $R_{F_{\mathcal{P}}}$ are: (i) new transition rules are added in $R_{CstrF_{\mathcal{P}}}$ to appropriately deal with constraint messages, (ii) strands are labeled with the role name, together with the identifier for distinguishing different instances, as explained in Section 5.3.1, (iii) transitions are also labeled, similarly as in the protocol process algebra, (iv) the global counter for generating fresh variables is deleted from the state. Instead, special unique names are assigned to fresh variable, which simplifies our notation.

In the constrained forwards strand semantics we label each transition rule similarly as in Section 5.2.3, that is, using labels of the form (ro, i, j, a, n), where ro, i, a, and n are as explained in Section 5.2.3, and j in this case is the position of the message that is being exchanged in the state transition. Also, similar to Section 5.2.3, for transitions that send out messages containing choice variables, all (possibly infinitely many) admissible ground substitutions for the choice variables are possible behaviors. A similar mechanism for distinguishing different fresh variables is used as that explained in Section 5.2.3. Since messages are introduced into strands in the state incrementally, we instantiate the fresh variables incrementally as well. Recall that fresh variables always first show up in a sent message. Therefore, each time a sent message is introduced into a strand in the state, we assign new names to the fresh variables in the message being introduced. The function *MaxStrId* for getting the max identifier for a constrained strand of a certain role is similar to *MaxProcId* in Section 5.2.3.

Since now messages in a strand can be sent or received messages, i.e., terms of the form m^+ or m^-, as well as constraint messages $\{Cstr, Num\}$, we represent them in the rules below simply as terms of the form u_i when their exact form is not relevant. We will use the precise form of the message when disambiguation is needed.

Before explaining the new transition rules for constraint messages, we show how the transition rules in [60] are labeled.

The constrained forwards strand semantics extends Maude-NPA's forwards semantics in [60] by adding transition rules to handle constraint messages, i.e, messages of the form $\{Cstr, Num\}$, where Num can be either 1 or 2. First, we add the two transition rules below for the cases when such a constrained message comes from explicit choices. Note that, as a consequence of well-formedness, the constraints introduce no new variables, and since the constraints that we consider are of the form $m \neq_{E_{\mathcal{P}}} m'$ or $m =_{E_{\mathcal{P}}} m'$, the satisfiability of $Cstr$ can be checked by checking whether the corresponding ground equality or disequality holds.

$$\left\{\begin{array}{l}\forall\ (ro)\ [u_1,\ldots,u_{j-1},u_j^+,u_{j+1},\ldots,u_n] \in P_{CstrSS} \wedge j{>}1 :\\ \{SS\,\&\,\{IK\}\,\&\,(ro,i)\ [u_1,\ldots,u_{j-1}]\}\\ \quad\rightarrow_{(ro,i,j,(u_j\rho_{ro,i}\sigma)^+,0)}\\ \{SS\,\&\,\{u_j\rho_{ro,i}\sigma{\in}\mathcal{I}, IK\}\,\&\,(ro,i)\ [u_1,\ldots,u_{j-1},(u_j\rho_{ro,i}\sigma)^+]\}\\ \qquad\qquad \mathsf{IF}\ (u_j\rho_{ro,i}\sigma{\in}\mathcal{I}) \notin IK\\ \textit{where } \sigma \textit{ is a ground substitution binding choice variables in } u_j,\\ \rho_{ro,i} = \{r_1 \mapsto r_1.ro.i,\ldots,r_n \mapsto r_n.ro.i\} \textit{ is a fresh substitution.}\end{array}\right\} \tag{F++}$$

$$\left\{\begin{array}{l}\forall\ (ro)\ [u_1,\ldots,u_{j-1},u_j^+,u_{j+1},\ldots,u_n] \in P_{CstrSS} \wedge j{>}1 :\\ \{SS\,\&\,\{IK\}\,\&\,(ro,i)\ [u_1,\ldots,u_{j-1}]\}\\ \quad\rightarrow_{(ro,i,j,u_j\rho_{ro,i}\sigma,0)}\\ \{SS\,\&\,\{IK\}\,\&\,(ro,i)\ [u_1,\ldots,u_{j-1},(u_j\rho_{ro,i}\sigma)^+]\}\\ \textit{where } \sigma \textit{ is a ground substitution binding choice variables in } u_j,\\ \rho_{ro,i} = \{r_1 \mapsto r_1.ro.i,\ldots,r_n \mapsto r_n.ro.i\} \textit{ is a fresh substitution.}\end{array}\right\} \tag{F+}$$

$$\left\{\begin{array}{l}\forall\ (ro)\ [u_1^+,\ldots,u_n] \in P_{CstrSS}:\\ \{SS\,\&\,\{IK\}\} \rightarrow_{(ro,i+1,j,(u_1\rho_{ro,i+1}\sigma)^+,0)}\\ \{SS\,\&\,(ro,i+1)\ [(u_1\rho_{ro,i+1}\sigma)^+]\,\&\,\{u_1\rho_{ro,i+1}\sigma{\in}\mathcal{I},IK\}\}\\ \quad\textsf{IF}\ (u_1\rho_{ro,i+1}\sigma{\in}\mathcal{I})\notin IK\\ \textit{where } \sigma \textit{ is a ground substitution binding choice variables in } u_1,\\ i = \textit{MaxStrId}(SS,ro),\\ \rho_{ro,i+1} = \{r_1 \mapsto r_1.ro.i+1,\ldots,r_n \mapsto r_n.ro.i+1\} \textit{ is a fresh substitution.}\end{array}\right\} \qquad \text{(F++\&)}$$

$$\left\{\begin{array}{l}\forall\ (ro)\ [u_1^+,\ldots,u_n] \in P_{CstrSS}:\\ \{SS\,\&\,\{IK\}\} \rightarrow_{(ro,i+1,j,u_1\rho_{ro,i+1}\sigma,0)}\\ \quad\{SS\,\&\,(ro,i+1)\ [(u_1\rho_{ro,i+1}\sigma)^+]\,\&\,\{IK\}\}\\ \textit{where } \sigma \textit{ is a ground substitution binding choice variables in } u_1,\\ i = \textit{MaxStrId}(SS,ro),\\ \rho_{ro,i+1} = \{r_1 \mapsto r_1.ro.i+1,\ldots,r_n \mapsto r_n.ro.i+1\} \textit{ is a fresh substitution.}\end{array}\right\} \qquad \text{(F+\&)}$$

$$\left\{\begin{array}{l}\forall\ (ro)\ [u_1,\ldots,u_{j-1},u_j^-,u_{j+1},\ldots,u_n] \in P_{CstrSS} \wedge j>1:\\ \{SS\ \&\{u_j{\in}\mathcal{I},IK\}\,\&\,(ro,i)\ [u_1,\ldots,u_{j-1}]\}\\ \rightarrow_{(ro,i,j,u_j^-,0)}\\ \{SS\,\&\,\{u_j{\in}\mathcal{I},IK\}\,\&\,(ro,i)\ [u_1,\ldots,u_{j-1},u_j^-]\}\end{array}\right\} \qquad \text{(F-)}$$

$$\left\{\begin{array}{l}\forall(ro)\ [u_1^-,u_2,\ldots,u_n] \in P_{CstrSS}:\\ \{SS\,\&\,\{u_1{\in}\mathcal{I},IK\}\}\\ \rightarrow_{(ro,i+1,1,u_1^-,0)} \{SS\ \&\ (ro,i+1)\ [u_1^-]\,\&\,\{u_1{\in}\mathcal{I},IK\}\}\\ \quad\textit{where } i = \textit{MaxStrId}(SS,ro)\end{array}\right\} \qquad \text{(F-\&)}$$

$$\left\{\begin{array}{l}\forall\ (ro)\ [u_1,\ldots,u_{j-1},\{Cstr,Num\},u_{j+1},\ldots,u_n] \in P_{CstrSS}\\ \wedge j > 1 :\\ \{SS\ \&\{IK\}\ \&\ (ro,i)\ [u_1,\ldots,u_{j-1}]\}\\ \rightarrow_{(ro,i,j,T,Num)}\\ \{SS\ \&\ \{IK\}\ \&\ (ro,i)\ [u_1,\ldots,u_{j-1},\{Cstr,Num\}]\}\\ \qquad \mathsf{IF}\ Cstr\end{array}\right\} \tag{Fif}$$

$$\left\{\begin{array}{l}\forall\ (ro)\ [u_1,\ldots,u_{j-1},\{?,Num\},u_{j+1},\ldots,u_n] \in P_{CstrSS}\\ \wedge j > 1 :\\ \{SS\ \&\{IK\}\ \&\ (ro,i)\ [u_1,\ldots,u_{j-1}]\}\\ \rightarrow_{(ro,i,j,?,Num)}\\ \{SS\ \&\ \{IK\}\ \&\ (ro,i)\ [u_1,\ldots,u_{j-1},\{?,Num\}]\}\end{array}\right\} \tag{F?}$$

The following set of transition rules adds to the state a new strand whose first message is a constraint message of the form $\{?, Num\}$:

$$\left\{\begin{array}{l}\forall\ (ro)\ [\,\{?,Num\},u_2,\ldots,u_n] \in P_{CstrSS} :\\ \{SS\ \&\ \{IK\}\}\\ \rightarrow_{(ro,i+1,1,?,Num)}\\ \{SS\ \&\ (ro,i+1)\ [\,\{?,Num\}\,]\ \&\ \{IK\}\}\\ \quad where\ i = MaxStrId(SS,ro)\end{array}\right\} \tag{F?\&}$$

Definition 5.4. Let $\mathcal{P}$ be a protocol with signature $\Sigma_{CstrSS_{\mathcal{P}}}$ and equational theory $E_{SS_{\mathcal{P}}}$. We define the *constrained forwards rewrite theory characterizing* $\mathcal{P}$ as $(\Sigma_{CstrSS_{\mathcal{P}}}, E_{SS_{\mathcal{P}}}, R_{CstrF_{\mathcal{P}}})$ where $R_{CstrF_{\mathcal{P}}} =$ (F++) $\cup$ (F+) $\cup$ (F++&) $\cup$ (F+&) $\cup$ (F-) $\cup$ (F-&) $\cup$ (Fif) $\cup$ (F?) $\cup$ (F?&).

5.5 BISIMULATION BETWEEN CONSTRAINED FORWARDS STRAND SEMANTICS AND PROCESS ALGEBRA SEMANTICS

In this section we show that the process algebra semantics and the constrained forwards strand semantics are label bisimilar. We first define PA-State and FW-State, the respective notions of state in each semantics.

Definition 5.5 (PA-State)**.** Given a protocol $\mathcal{P}$, a *PA-State* of $\mathcal{P}$ is a state in the protocol process algebra semantics that is *reachable* from the initial state. The initial PA-State is $P_{init} = \{\varnothing \mid \{empty\}\}$.

Definition 5.6 (FW-State)**.** Given a protocol $\mathcal{P}$, a *FW-State* of $\mathcal{P}$ is a state in the constrained forwards strand semantics that is *reachable* from the initial state. The initial FW-State is $F_{init} = \{\varnothing \ \& \ \{empty\}\}$.

The bisimulation relation is defined based on reachability, i.e., if a PA-State and a FW-State are in the relation $\mathcal{H}$, then they both can be reached from their corresponding initial states by the same label sequence. Note that we only consider states that are reachable from the initial states.

Let us first define the notation of label sequence that we will use throughout.

Definition 5.7 (Label Sequence)**.** An ordered sequence α of transition labels is defined by using $\_.\_$ as an associative concatenation operator with *nil* as an identity. The length of a label sequence α is denoted by $|\alpha|$. Given a label sequence α, we denote by $\alpha|_{(ro,i)}$ the sub-sequence of labels in α that have ro as role name, and i as identifier, i.e., labels of the form $(ro, i, \_, \_, \_)$ ($\_$ is a shorthand for denoting any term).

Definition 5.8 (Relation $\mathcal{H}$)**.** Given a protocol $\mathcal{P}$, the relation $\mathcal{H}$ is defined as: $\mathcal{H} = \{(Pst, Fst) \in \textit{PA-State} \times \textit{FW-State} \mid \exists \textit{ label sequence } \alpha \textit{ s.t. } P_{init} \rightarrow_{\alpha} Pst,\ F_{init} \rightarrow_{\alpha} Fst\}$.

Recall that a process can be "deconstructed" by the mapping *toCstrSS* into a set of constrained protocol strands, each representing a possible execution path. If a PA-State *Pst* and a FW-State *Fst* are related by $\mathcal{H}$, then an important observation is that there is a *duality* between individual processes in *Pst* and strands in *Fst*: if there is a process in the *Pst* describing a role's *continuation* in the future, there will be a corresponding strand in *Fst* describing the part of the process that has *already been executed*, and vice versa. Another observation is that, since the intruder's knowledge is extracted from the communication history, following the definition of $\mathcal{H}$, the states *Pst* and *Fst* have the same communication history. Therefore, they have the same intruder's knowledge. We formalize these observations in Lemmas 5.1 and 5.2. These lemmas then lead us to the main result that $\mathcal{H}$ is a bisimulation relation.

We now define the relation $\mathcal{H}_{LP\_Str}$, which relates a possibly partially executed labeled process and a constrained strand. This relation defines the *duality* relation between a labeled process and a constrained strand. If a labeled process *LP* is related to a constrained strand *Str* by the relation $\mathcal{H}_{LP\_Str}$, then: (i) *LP* and *Str* denote the behavior of the same role with the same identity in the same protocol, and (ii) for any strand Str_{LP}, Str_{LP} denotes a possible execution path of LP iff Str followed by Str_{LP} forms a valid possible execution path of the protocol.

Definition 5.9 (Relation $\mathcal{H}_{LP\_Str}$)**.** Given a protocol $\mathcal{P}$, and a possibly partially executed labeled process LP of $\mathcal{P}$, a possibly partially executed constrained strand Str of $\mathcal{P}$, then $(LP, Str) \in \mathcal{H}_{LP\_Str}$ iff

$$\begin{aligned} toCstrSS(LP) = \&\{(ro,i)[u_{j+1},\dots,u_n]\rho_{ro,i}\theta \mid \exists\ ground\ substitution\ \theta \\ \exists(ro)[u_1,\dots u_j,u_{j+1},\dots,u_n] \in \mathcal{P}_{Cstr}\ s.t.\ Str = (ro,i)[u_1,\dots u_j]\rho_{ro,i}\theta\} \end{aligned}$$

where $\&\{S_1, S_2, \dots, S_n\}$ is a shorthand for a term $S_1 \& S_2 \& \dots \& S_n$ denoting a set of strands. $\rho_{ro,i} = \{r_1 \mapsto r_1.ro.i, \dots, r_m \mapsto r_m.ro.i\}$ for fresh variables $r_1, \dots, r_m$ in $[u_1, \dots u_j, u_{j+1}, \dots, u_n]$.

Example 5.6. Following Examples 5.3 and 5.5, we show a process LP and a strand Str that are related by the relation $\mathcal{H}_{LP\_Str}$. LP (resp. Str) is the labeled process (resp. constrained strand) of the Server role after making the first explicit nondeterministic choice.

$$\begin{aligned} LP =& (Server, 1, 2)\ \sigma(+(hs; retry) \cdot -(hs; N'; G'; gen(G'); E') \cdot \\ & + (hs; n(S_?, r1); G'; gen(G'); keyG(G', S_?, r_2); \\ & Z(AReq_?, G', E', S, r_2, S_?, HM))) \\ Str =& (Server, 1)\ \sigma[\ \{?, 1\}, -(hs; N; G; gen(G); E)] \end{aligned}$$

where σ is a ground substitution instantiating the pattern variables N, G, and E.

We then lift the duality relation between individual processes and strands to a duality relation between a PA-State and a FW-State.

Definition 5.10 (Relation $\mathcal{H}_{PS\_FS}$)**.** Let $Pst = \{LP_1 \& \dots \& LP_n \mid \{IK\}\}$ be a PA-State and $Fst = \{Str_1 \& \dots \& Str_m \& \{IK'\}\}$ be a FW-State, if $(Pst, Fst) \in \mathcal{H}_{PS\_FS}$, then:

(i) For each labeled process $LP_k \in Pst$, $1 \leqslant k \leqslant n$, there exists a strand $Str_{k'} \in Fst$, $1 \leqslant k' \leqslant m$, such that $(LP_k, Str_{k'}) \in \mathcal{H}_{LP\_Str}$.

(ii) For each strand $Str_{k'} \in Fst$, $1 \leqslant k' \leqslant m$, there exists a labeled process $LP_k \in Pst$, $1 \leqslant k \leqslant n$, such that $(LP_k, Str_{k'}) \in \mathcal{H}_{LP\_Str}$.

The lemma below states that the relation $\mathcal{H}$ induces the *duality* relation $\mathcal{H}_{PS\_FS}$.

Lemma 5.1. *Let $Pst = \{LP_1\&\ldots\&LP_n \mid \{IK\}\}$ be a PA-State and $Fst = \{Str_1\&\ldots\&Str_m\&\{IK'\}\}$ be a FW-State, if $(Pst, Fst) \in \mathcal{H}$, i.e., exists a label sequence α such that $P_{init} \rightarrow_\alpha Pst$, and $F_{init} \rightarrow_\alpha Fst$, then $(Pst, Fst) \in \mathcal{H}_{PS\_FS}$.*

Proof. We first prove property (i). If $|\alpha| = 0$, since both the strand set and the process configuration are empty, the statement is vacuously true.

Now suppose that $|\alpha| > 0$. Then, without loss of generality, assume there exists a labeled process $LP_k = ((ro, i, j)\ P_k)$ in Pst, with $i, j \geqslant 1$. Then there is at least one label in α of the form $(ro, i, \_, \_, \_)$ ($\_$ is a short hand for any content), therefore, there is a strand $St_{k'}$ in Fst of the form $(ro, i)[v_1, \ldots, v_{j'}]$.

We then show that the above-mentioned LP_k and $Str_{k'}$ are related by $\mathcal{H}_{LP\_Str}$, i.e., $(LP_k, Str_{k'}) \in \mathcal{H}_{LP\_Str}$. Since the state Fst is reachable from the initial state by the label sequence α, and $Str_{k'} \in Fst$, $[v_1, \ldots, v_{j'}]$ denotes exactly the sequence of messages in the unique sequence of labels $\alpha|_{(ro,i)}$. Moreover, $j' = j - 1$.

Since the process state Pst is reachable from the initial state P_{init} by label sequence α, there exists a unique process $(ro)P_{spec}$ in the specification P_{PA}, and LP_k represents all possible behaviors of $(ro)P_{spec}$ after the sequence of transitions $\alpha|_{(ro,i)}$. Therefore, $toCstrSS(LP_k) =$

$$\begin{aligned}
&\&\{(ro, i)[u_j, \ldots, u_n]\rho_{ro,i}\theta \mid \\
&\exists \textit{ ground substitution } \theta \\
&\exists (ro)[u_1, \ldots, u_{j-1}, u_j, \ldots, u_n] \in toCstrSS((ro)P_{spec}) \\
&s.t.\ (ro, i)[u_1, \ldots, u_{j-1}]\rho_{ro,i}\theta = (ro, i)[v_1, \ldots, v_{j-1}]\}
\end{aligned}$$

By the correspondence between protocol specifications defined in definition 5.3 , $\mathcal{P}_{CstrF} = toCstrSS(P_{PA})$. Also note that $(ro)P_{spec}$ is the only process in P_{PA} that has ro as its role name, therefore, $toCstrSS((ro)P_{spec}) = \{(ro)[u_1, \ldots, u_n] \mid (ro)[u_1, \ldots, u_n] \in \mathcal{P}_{CstrF}\}$. Therefore, $toCstrSS(LP_k) =$

$$\begin{aligned}
&\&\{(ro, i)[u_j, \ldots u_n]\rho_{ro,i}\theta \mid \\
&\exists \textit{ ground substitution } \theta \\
&\exists (ro)[u_1, \ldots, u_{j-1}, u_j, \ldots, u_n] \in \mathcal{P}_{CstrF} \\
&s.t.\ (ro, i)[u_1, \ldots, u_{j-1}]\rho_{ro,i}\theta = (ro, i)[v_1, \ldots, v_{j-1}]\}.
\end{aligned}$$

Therefore, $(LP_k, Str_{k'}) \in \mathcal{H}_{LP\_Str}$.

The proof for property (ii) is similar to the one for property (i).

Lemma 5.2 below formalizes the observation that the equivalence of label sequence implies the same intruder knowledge.

Lemma 5.2. *Given a PA-State Pst and a FW-State Fst such that $(Pst, Fst) \in \mathcal{H}$, i.e., there exists a label sequence α such that $P_{init} \to_\alpha Pst$ and $F_{init} \to_\alpha Fst$, then the contents of intruder knowledge in Pst and in Fst are syntactically equal.*

Proof. In both semantics the only transition rules that add new elements to the intruder's knowledge are the ones whose label is of the form $(ro, i, j, +m, n)$. Therefore, given the two states Pst and Fst as described above, their intruder's knowledge can be computed from the sequence of labeled transitions α as $IK(Pst) = \{m \in \mathcal{I} \mid (\_, \_, \_, +m, \_) \in \alpha\} = IK(Fst)$.

Based on the lemmas above, we can now show that the relation $\mathcal{H}$ is a bisimulation. Since the proof of Theorem 5.1 requires a somewhat lengthy case analysis, it has been moved to Appendix B.

Theorem 5.1 (Bisimulation)**.** *$\mathcal{H}$ is a bisimulation.*

5.6 CONSTRAINED BACKWARDS STRAND SEMANTICS

In this section we extend Maude-NPA's symbolic backwards semantics with rules for constrained messages of the form described in Section 5.3.1, so that it can analyze protocols exhibiting explicit choices. We refer to this extended backwards semantics as *constrained backwards strand semantics*. We then show that the *constrained backwards strand semantics* is sound and complete with respect to the constrained forwards strand semantics presented in Section 5.4, and the process algebra semantics presented in Section 5.2. This result allows us to use Maude-NPA for analyzing protocols exhibiting choice, including both implicit and explicit choices, and in particular any protocol specified using the *protocol process algebra*.

The strand space model used in the constrained backwards strand semantics is the same as the one already used in Maude-NPA [16], except for the following differences:

- Maude-NPA explores *constrained states* as defined in [93], that is, states that have an associated constraint store. More specifically, a *constrained state* is a pair $\langle St, \Psi \rangle$ consisting of a state expression St and a *constraint*, i.e., a set Ψ understood as a conjunction $\Psi = \bigwedge_{i=1}^{n} u_i \neq v_i$ of disequality constraints.

- Strands are now of the form $[u_1, \ldots, u_i \mid u_{i+1}, \ldots u_n]$, where each u_k can be of one of these forms: (i) m^+ if it is a sent message, (ii) m^- if it is a received message, or (iii) $\{Cstr, Num\}$ if it is a constrained message.

State changes are described by a set $R^{-1}_{CstrB_{\mathcal{P}}}$ of *rewrite rules*, so that the rewrite theory $(\Sigma_{CstrSS_{\mathcal{P}}}, E_{SS_{\mathcal{P}}}, R^{-1}_{CstrB_{\mathcal{P}}})$ characterizes the behavior of protocol $\mathcal{P}$ modulo the equations $E_{SS_{\mathcal{P}}}$ for *backwards* execution. The set of rules $R^{-1}_{CstrB_{\mathcal{P}}}$ is obtained as follows. First, we adapt the set of rules $R^{-1}_{B_{\mathcal{P}}}$ in Chapter 3 to constrained states, which is an embedding of rules in $R^{-1}_{B_{\mathcal{P}}}$. Their forwards version is shown below:

$$\begin{aligned}&\langle\{SS \,\&\, (ro)[L \mid M^-, L'] \,\&\, \{M{\in}\mathcal{I}, IK\}\}, \Psi\rangle\\ &\to \langle\{SS \,\&\, (ro)[L, M^- \mid L'] \,\&\, \{M{\in}\mathcal{I}, IK\}\}, \Psi\rangle\end{aligned} \tag{B-}$$

$$\begin{aligned}&\langle\{SS \,\&\, (ro)[L \mid M^+, L'] \,\&\, \{IK\}\}, \Psi\rangle\\ &\to \langle\{SS \,\&\, (ro)[L, M^+ \mid L'] \,\&\, \{IK\}\}, \Psi\rangle\end{aligned} \tag{B+}$$

$$\begin{aligned}&\langle\{SS \,\&\, (ro)[L \mid M^+, L'] \,\&\, \{M{\notin}\mathcal{I}, IK\}\}, \Psi\rangle\\ &\to \langle\{SS \,\&\, (ro)[L, M^+ \mid L'] \,\&\, \{M{\in}\mathcal{I}, IK\}\}, \Psi\rangle\end{aligned} \tag{B++}$$

$$\begin{aligned}&\forall\ (ro)[l_1, u^+, l_2] \in \mathcal{P} :\\ &\langle\{\{SS \,\&\, (ro)[\,l_1 \mid u^+, l_2\,] \,\&\, \{u{\notin}\mathcal{I}, IK\}\}, \Psi\rangle\\ &\to \langle\{SS \,\&\, \{u{\in}\mathcal{I}, IK\}\}\}, \Psi\rangle\end{aligned} \tag{B\&}$$

where L and L' are variables denoting a list of strand messages, IK is a variable for a set of intruder facts ($m{\in}\mathcal{I}$ or $m{\notin}\mathcal{I}$), SS is a variable denoting a set of strands, and l_1, l_2 denote a list of strand messages.

Then, we define new transition rules for constrained messages. That is, we add the reversed version of the following rules:

$$\begin{aligned}&\langle\{SS \,\&\, \{IK'\} \,\&\, (ro)[L \mid \{?, Num\}, L']\}, \Psi\rangle\\ &\to \langle\{SS \,\&\, \{IK'\} \,\&\, (ro)[L, \{?, Num\} \mid L']\}, \Psi\rangle\end{aligned} \tag{B?}$$

$$\begin{aligned}&\langle\{SS \,\&\, \{IK\} \,\&\, (ro)[L \mid \{M =_{E_{\mathcal{P}}} M, Num\}, L']\}, \Psi\rangle\\ &\to \langle\{SS \,\&\, \{IK\} \,\&\, (ro)[L, \{M =_{E_{\mathcal{P}}} M, Num\} \mid L']\}, \Psi\rangle\end{aligned} \tag{Bif=}$$

$$\begin{aligned}&\langle\{SS \,\&\, \{IK\} \,\&\, (ro)[L \mid \{M \neq M', Num\}, L']\}, (\Psi \wedge M \neq M')\rangle\\ &\to \langle\{SS \,\&\, \{IK\} \,\&\, (ro)[L, \{M \neq M', Num\} \mid L']\}, \Psi\rangle\\ &\text{if } (\Psi \wedge M \neq_{E_{\mathcal{P}}} M') \text{ is satisfiable in } \mathcal{T}_{\Sigma_{CstrSS_{\mathcal{P}}}/E_{\mathcal{P}}}(\mathcal{X})\end{aligned} \tag{Bif$\neq$}$$

Rule (B?) processes a constraint message denoting an explicit non-deterministic choice with constant "?". The constraint store is not changed and no satisfiability check is required.

Rules (Bif=) and (Bif≠) deal with constrained messages associated to explicit deterministic choices. Since the only constraints we allow in explicit deterministic choices are equalities and disequalities, rule (Bif=) is for the case when the constraint is an equality, rule (Bif≠) is for the case when the constraint is a disequality. The equality constraint is solved by $E_{\mathcal{P}}$-unification. The constraint in a *constrained state* is therefore a *disequality constraint*, i.e., $\Psi = \bigwedge_{i=1}^{n} u_i \neq_{E_{\mathcal{P}}} v_i$. The *semantics* of such a constrained state, written $[\![\langle St, \Psi\rangle]\!]$ is the set of all ground substitution instances of the form:

$$[\![\langle St, \Psi\rangle]\!] = \{St\theta \mid \theta \in [\mathcal{X} \to \mathcal{T}_{\Sigma_{\mathcal{P}}}] \wedge u_i\theta \neq_{E_{\mathcal{P}}} v_i\theta, 1 \leqslant i \leqslant n\}$$

The disequality constraints are then solved the same way as in [93].

Definition 5.11. Let $\mathcal{P}$ be a protocol with signature $\Sigma_{CstrSS_{\mathcal{P}}}$ and equational theory $E_{\mathcal{P}}$. We define the *constrained backwards rewrite theory characterizing* $\mathcal{P}$ to be $(\Sigma_{CstrSS_{\mathcal{P}}}, E_{SS_{\mathcal{P}}}, R^{-1}_{CstrB_{\mathcal{P}}})$, where $R^{-1}_{CstrB_{\mathcal{P}}}$ is the result of reversing the rewrite rules {(B-), (B+), (B++), (B?), (Bif=), (Bif≠)} ∪ (B&).

5.7 SOUNDNESS AND COMPLETENESS OF CONSTRAINED BACKWARDS STRAND SEMANTICS

The soundness and completeness proofs generalize the proofs in [60]. Recall that the state in the constrained states of constrained backwards strand semantics is a symbolic strand state, i.e., a state with variables. A state in the forwards strand semantics is a ground strand state, i.e., a state without variables. The lifting relation defines the instantiation relation between symbolic and ground states.

We define a symbolic state and a ground state as follows.

Definition 5.12 (Symbolic Strand State)**.** Given a protocol $\mathcal{P}$, a *symbolic* strand state S of $\mathcal{P}$ is a term of the form:

$$\begin{aligned} S = \{ & :: r_{1_1}, \ldots, r_{m_1} :: [u_{1_1}, \ldots u_{i_1-1} \mid u_{i_1}, \ldots, u_{n_1}] \ \& \\ & \vdots \\ & :: r_{1_k}, \ldots, r_{m_k} :: [u_{1_k}, \ldots, u_{i_k-1} \mid u_{i_k}, \ldots, u_{n_k}] \ \& \ SS \\ & \{w_1 \in \mathcal{I}, \ldots, w_m \in \mathcal{I}, \ w'_1 \notin \mathcal{I}, \ldots, w'_{m'} \notin \mathcal{I}, IK\}\} \end{aligned}$$

where for each $1 \leqslant j \leqslant k$, there exists a strand $[m_{1_j}, \ldots m_{i_j-1}, m_{i_j}, \ldots, m_{n_j}] \in P_{CstrSS}$ and a substitution $\rho_j : \mathcal{X} \to \mathcal{T}_{\Sigma_\mathcal{P}}(\mathcal{X})$ such that $m_{1_j}\rho_j =_{E_\mathcal{P}} u_{1_j}, \ldots, m_{n_j}\rho_j =_{E_\mathcal{P}} u_{n_j}$, SS is a variable denoting a (possibly empty) set of strands, and IK is a variable denoting a (possibly empty) set of intruder's knowledge facts.

Definition 5.13 (Ground Strand State)**.** Given a protocol $\mathcal{P}$, a *ground* strand state s of $\mathcal{P}$ is a term without variables of the form:

$$\begin{aligned} s = \{&[u_{1_1}, \ldots u_{i_1-1}] \mathbin{\&} \cdots \mathbin{\&} [u_{1_k}, \ldots, u_{i_k-1}] \mathbin{\&} \\ &\{w_1 \in \mathcal{I}, \ldots, w_m \in \mathcal{I}\}\ \} \end{aligned}$$

where for each $1 \leqslant j \leqslant k$, there exists a strand $[m_{1_j}, \ldots m_{i_j-1}, m_{i_j}, \ldots, m_{n_j}] \in P_{CstrSS}$ and a substitution $\rho_j : \mathcal{X} \to \mathcal{T}_{\Sigma_\mathcal{P}}$ such that $m_{1_j}\rho_j =_{E_\mathcal{P}} u_{1_j}, \ldots, m_{i_j}\rho_j =_{E_\mathcal{P}} u_{i_j}$.

The lifting relation in [60] is extended with constraints and constrained messages. Note that the u_i in the definition below can be sent messages, received messages, or constrained messages.

Definition 5.14 (Lifting Relation)**.** Given a protocol $\mathcal{P}$, a constrained symbolic strand state $CstrS = \langle S, \Psi \rangle$ and a ground strand state s, we say that s *lifts* to $CstrS$, or that $CstrS$ *instantiates* to s with a *ground* substitution $\theta : (Var(S) - \{SS, IK\}) \to \mathcal{T}_{\Sigma_\mathcal{P}}$, written $CstrS >^\theta s$ iff

- for each strand $:: r_1, \ldots, r_m :: [u_1, \ldots u_{i-1} \mid u_i, \ldots, u_n]$ in S, there exists a strand $[v_1, \ldots v_{i-1}]$ in s such that $\forall 1 \leqslant j \leqslant i-1$, $v_j =_{E_\mathcal{P}} u_j\theta$.
- for each positive intruder fact $w \in \mathcal{I}$ in S, there exists a positive intruder fact $w' \in \mathcal{I}$ in s such that $w' =_{E_\mathcal{P}} w\theta$, and
- for each negative intruder fact $w \notin \mathcal{I}$ in S, there is no positive intruder fact $w' \in \mathcal{I}$ in s such that $w' =_{E_\mathcal{P}} w\theta$.
- $E_\mathcal{P} \models \Psi\theta$.

In the following we show the soundness and completeness of transitions in constrained backwards strand semantics w.r.t. the constrained forwards strand semantics by proving two lemmas stating the completeness and soundness of one-step transition in the constrained backwards strand semantics w.r.t. the constrained forwards strand semantics. The soundness and completeness result directly follows these two lemmas. In the proofs we consider only transition rules added in both semantics to deal with explicit choices, that is, rules (Fif) $\cup$

$$\begin{tikzcd}
CstrS \arrow[d, dotted, ">^{\theta}"'] & CstrS' \arrow[l, dotted] \arrow[d, ">^{\theta'}"] \\
s \arrow[r] & s'
\end{tikzcd}$$

Figure 5.1: Lemma 5.3

(F?) $\cup$ (F?&) in the constrained forwards strand semantics and rules $\{(\text{B?}), (\text{Bif=}), (\text{Bif}\neq)\}$ in the constrained backwards strand semantics. The proof of the soundness and completeness of one-step transitions performed in the constrained backwards strand semantics using rules $\{(\text{B-}), (\text{B+}), (\text{B++})\} \cup (\text{B\&})$ w.r.t to one-step transitions performed in the constrained forwards strand semantics using rules (F++) $\cup$ (F+) $\cup$ (F++&) $\cup$ (F+&) $\cup$ (F-) $\cup$ (F-&) is the same as in [60], since in these transitions no constraint is involved. Note that although in [60], *Choice Variables* were not defined explicitly, the proof extends to strands with choice variables naturally, since the lifting relation between a ground state and a symbolic state does not need to be changed to cover choice variables. Since the strand labels are irrelevant for the results of this section, we will omit them to simplify the notation from now on. Also, we include the fresh substitution in the substitutions and do not separate the fresh substitutions explicitly.

Extending the proofs in [60], we first prove how the lifting of a ground state to a symbolic state induces a lifting of a forwards rewriting step in the forwards semantics to a backwards narrowing step in the backwards semantics, i.e., the completeness of one-step transition. The lemma below extends the lifting lemma in [60] to strands with constrained messages.

Lemma 5.3 (Lifting Lemma)**.** *Given a protocol $\mathcal{P}$, two ground strand states s and s', a constrained symbolic strand state $CstrS' = \langle S', \Psi' \rangle$ and a substitution θ' s.t. $s \to s'$ and $CstrS' >^{\theta'} s'$, then there exists a constrained symbolic strand state $CstrS = \langle S, \Psi \rangle$ and a substitution θ s.t. $CstrS >^{\theta} s$ and either $CstrS \overset{\mu}{\leftsquigarrow} CstrS'$ or $CstrS = CstrS'$.*

The Lifting Lemma is illustrated by Figure 5.1.

Proof. As has been explained before, we only need to consider the new rules: (Fif), (F?), (F?&). The proof in [60] is structured by cases, some of which having specific requirements on intruder knowledge, or involve changes made to the intruder knowledge. Since all the new rules we are considering do not have specific requirements on the intruder knowledge, and do not change the intruder knowledge either, the cases that we need to consider are the following (cases e and f in the proof in [60]), which involve the appearance or non-appearance of certain strand(s):

e: There is a strand $[u_1, \ldots, u_{j-1}, u_j, \ldots, u_n]$ in P_{CstrSS}, $n \geqslant 1$, $1 \leqslant j \leqslant n$, and a substitution ρ such that $[u_1, \ldots, u_{j-1}, u_j]\rho$ is a strand in s' and $[u_1, \ldots, u_{j-1}, u_j \mid u_{j+1}, \ldots, u_n]\rho$

is a strand in $S'\theta'$.

f: There is a strand $[u_1, \ldots, u_{j-1}, u_j, \ldots, u_n]$ in P_{CstrSS}, $n \geqslant 1$, $1 \leqslant j \leqslant n$, and a substitution ρ such that $[u_1, \ldots, u_{j-1}, u_j]\rho$ is a strand in s' but $[u_1, \ldots, u_{j-1}, u_j \mid u_{j+1}, \ldots, u_n]\rho$ is not a strand in $S'\theta'$.

Now we consider for the forward rewrite rule application in the step $s \rightarrow s'$.

- Given ground states s and s' s.t. $s \rightarrow s'$ using a rule in set (Fif), then there exists a ground substitution τ, variables SS' and IK', and strand $[u_1, \ldots, u_{j-1}, \{T, Num\}, u_{j+1}, \ldots, u_n]$ in P_{CstrSS}, such that $s = \{SS'\tau \& \{IK'\tau\} \& (ro)[u_1\tau, \ldots, u_{j-1}\tau]\}$, and $s' = \{SS'\tau \& \{IK'\tau\} \& [u_1\tau, \ldots, u_{j-1}\tau, \{T\tau, Num\}]\}$ and $T\tau =_{E_\mathcal{P}} true$. Since there exists a substitution θ' s. t. $CstrS' >^{\theta'} s'$, we consider the following two cases:
  - Case (e) The strand appears in $S'\theta'$. More specifically, $[u_1\sigma, \ldots, u_{j-1}\sigma, \{T\sigma, Num\} \mid u_{j+1}\sigma, \ldots, u_n\sigma]$ is a strand in S' s.t. $\sigma\theta' =_{E_\mathcal{P}} \tau$. If the constraint T is an equality constraint, since $T\tau =_{E_\mathcal{P}} T\sigma\theta' =_{E_\mathcal{P}} true$, and by the lifting relation, $E_\mathcal{P} \models \Psi'\theta'$, rule (Bif=) can be applied for the backwards narrowing $CstrS' \overset{\mu}{\leftsquigarrow} CstrS$, and $CstrS >^{\theta} s$ such that $\mu\theta =_{E_\mathcal{P}} \theta'$. If the constraint T is a disequality constraint, since $T\tau =_{E_\mathcal{P}} T\sigma\theta' =_{E_\mathcal{P}} true$, and by the lifting relation, $E_\mathcal{P} \models \Psi'\theta'$, we have $E_\mathcal{P} \models T\sigma\theta' \wedge \Psi'\theta'$. Therefore, rule (Bif≠) can be applied for the backwards narrowing, and $CstrS >^{\theta} s$.
  - Case (f) The strand does not appear in $S'\theta'$. Then θ' makes S' a valid symbolic strand state of s, i.e., $S = S'$ and $CstrS' >^{\theta'} s$.

- Given ground strand states s and s' s.t. $s \rightarrow s'$ using a rule in set (F?), then we consider the following two applicable cases:
  - Case (e) The strand appears in $S'\theta'$ and thus we can perform a backwards narrowing step from $CstrS'$ with rule (B?), i.e., $CstrS' \rightsquigarrow CstrS$, and $CstrS >^{\theta'} s$.
  - Case (f) The strand does not appear in $S'\theta'$. Then θ' makes $CstrS'$ as a valid constraint symbolic state of s, i.e., $CstrS = CstrS'$ and $CstrS >^{\theta'} s$.

- Given states s and s' s.t. $s \rightarrow s'$ using a rule in set (F?&), the proof is similar with using a rule in the set (F?).

Theorem 5.2 below then follows straightforwardly.

$$\begin{tikzcd}
CstrS \arrow[d, ">^{\theta}"'] & CstrS' \arrow[l, squiggly] \arrow[d, dotted, ">^{\theta'}"] \\
s \arrow[r, dotted] & s'
\end{tikzcd}$$

Figure 5.2: Lemma 5.4

Theorem 5.2 (Completeness). *Given a protocol $\mathcal{P}$, two ground strand states s, s_0, a constrained symbolic strand state CstrS and a substitution θ s.t. (i) s_0 is an initial state, (ii) $s_0 \rightarrow^n s$, and (iii) $CstrS >^{\theta} s$. Then there exists a constrained symbolic initial strand state $CstrS_0$, two substitutions μ and θ', and $k \leqslant n$, s.t. $CstrS_0 \overset{k}{\leftsquigarrow}_{\mu} CstrS$, and $CstrS_0 >^{\theta'} s_0$.*

The Soundness Theorem from [60] can also be extended to constrained backwards and forwards strand semantics. We first show that Lemma 2 in [60], which states the soundness of one-step transition, still holds after extending it to constrained states. The Soundness Theorem then follows straightforwardly.

Lemma 5.4. *Given a protocol $\mathcal{P}$, two constrained symbolic states $CstrS = \langle S, \Psi \rangle$ and $CstrS' = \langle S', \Psi' \rangle$, a ground strand state s and a ground substitution θ, if $CstrS \overset{\mu}{\leftsquigarrow} CstrS'$ and $CstrS >^{\theta} s$, then there exists a ground strand state s' and a ground substitution θ' such that $s \rightarrow s'$, and $CstrS' >^{\theta'} s'$.*

Lemma 5.4 is illustrated by the Figure 5.2.

Proof. We only need to consider the new rules: rule (Bif=), (Bif≠) and (B?).
1) If $CstrS \overset{\mu}{\leftsquigarrow} CstrS'$ using rule (B?), then there are associated rules in the sets (F?) and (F?&).
2) If $CstrS \overset{\mu}{\leftsquigarrow} CstrS'$ using rule (Bif=), there is a strand $[u_1\sigma, \dots, u_{j-1}\sigma \mid \{(u = v)\sigma, Num\}, u_{j+1}\sigma, \dots, u_n\sigma]$ in S, $[u_1\sigma', \dots, u_{j-1}\sigma', \{(u = v)\sigma', Num\} \mid u_{j+1}\sigma', \dots, u_n\sigma']$ in S' s.t. $\sigma =_{E_{\mathcal{P}}} \sigma'\mu$, $\Psi =_{E_{\mathcal{P}}} \Psi'\mu$ and $u\sigma =_{E_{\mathcal{P}}} v\sigma$, where $[u_1, \dots, u_{j-1}, \{u = v, Num\}, u_{j+1}, \dots, u_n]$ is a strand in P_{CstrSS}. Since $CstrS >^{\theta} s$, there is a ground strand $[u_1\sigma\theta, \dots, u_{j-1}\sigma\theta]$ in s, and $E_{\mathcal{P}} \models \Psi\theta$. Therefore, $E_{\mathcal{P}} \models \Psi'\mu\theta$ and $u\sigma\theta =_{E_{\mathcal{P}}} v\sigma\theta$. By rule (Fif), $s \rightarrow s'$, and $CstrS' >^{\mu\theta} s'$.

If $CstrS \overset{\mu}{\leftsquigarrow} CstrS'$ using rule (Bif≠), there is a strand $[u_1\sigma, \dots, u_{j-1}\sigma \mid \{(u \neq v)\sigma, Num\}, u_{j+1}\sigma, \dots, u_n\sigma]$ in S , $[u_1\sigma', \dots, u_{j-1}\sigma', \{(u \neq v)\sigma', Num\} \mid u_{j+1}\sigma', \dots, u_n\sigma']$ in S' s.t. $\sigma =_{E_{\mathcal{P}}} \sigma'\mu$ and $\Psi =_{E_{\mathcal{P}}} \Psi'\mu \wedge (u \neq v)\sigma'\mu$, where $[u_1, \dots, u_{j-1}, \{u \neq v, Num\}, u_{j+1}, \dots, u_n]$ is a strand in P_{CstrSS}. Since $CstrS >^{\theta} s$, there is a ground strand $[u_1\sigma\theta, \dots, u_{j-1}\sigma\theta]$ in s, and $E_{\mathcal{P}} \models \Psi\theta$. Therefore, $E_{\mathcal{P}} \models \Psi'\mu\theta \wedge (u \neq v)\sigma'\mu\theta$. By rule (Fif), $s \rightarrow s'$, and $CstrS' >^{\mu\theta} s'$.

The Soundness Theorem below shows that the backwards symbolic reachability analysis is *sound* with respect to the forwards rewriting-based strand semantics.

Theorem 5.3 (Soundness)**.** *Given a protocol $\mathcal{P}$, two constrained symbolic strand states $CstrS_0, CstrS'$, an initial ground strand state s_0 and a substitution θ s.t. (i) $CstrS_0$ is a symbolic initial state, and (ii) $CstrS_0 \overset{*}{\leftsquigarrow} CstrS'$, and (iii) $CstrS_0 >^{\theta} s_0$. Then there exists a ground strand state s' and a substitution θ', s.t. (i) $s_0 \rightarrow^{*} s'$, and (ii) $CstrS' >^{\theta'} s'$.*

The soundness and completeness results in Theorems 5.3 and 5.2 together with the bisimulation proved in Theorem 5.1 show that the backwards symbolic reachability analysis is *sound* and *complete* with respect to the process algebra semantics.

Theorem 5.4 (Soundness)**.** *Given a protocol $\mathcal{P}$, two constrained symbolic strand states $CstrS_0, CstrS$, the initial FW-State F_{init}, a substitution θ, and the initial PA-State P_{init} s.t. (i) $CstrS_0$ is a symbolic initial strand state, and (ii) $CstrS_0 \overset{*}{\leftsquigarrow}_{\mu} CstrS$, and (iii) $CstrS_0 >^{\theta} F_{init}$. Then there exists a FW-State Fst such that $CstrS >^{\theta'} Fst$, and therefore, there is a PA-State Pst such that Pst $\mathcal{H}$ Fst.*

Theorem 5.5 (Completeness)**.** *Given a protocol $\mathcal{P}$, a PA-State Pst, a FW-State Fst, a constrained symbolic strand state CstrS s.t. (i) Pst $\mathcal{H}$ Fst, (ii) $CstrS >^{\theta'} Fst$. Then there is a backwards symbolic execution $CstrS_0 \overset{*}{\leftsquigarrow}_{\mu} CstrS$ s.t. $CstrS_0$ is a symbolic initial strand state as defined in Chapter 3, and $CstrS_0 >^{\theta} F_{init}$.*

5.8 PROTOCOL EXPERIMENTS

In this section we describe some experiments that we have performed on protocols with choice. We have fully integrated the process algebra syntax, and its transformation into strands, and have developed new methods to specify attack states using the process notation in the recent release of Maude-NPA 3.0 (see [58]).

5.8.1 Integration of the Protocol Process Algebra in Maude-NPA

We have fully implemented the process algebra notation in Maude-NPA. Strands represent each role behavior as a linear sequence of message outputs and inputs but processes represent each role behavior as a possibly non-linear sequence of message outputs and inputs. The honest principal specification is specified in the process algebra syntax. In order for Maude-NPA to accept process specifications, we have replaced the section `STRANDS-PROTOCOL` from the Maude-NPA protocol template by a new section `PROCESSES-PROTOCOL`; see [58] for details. The intruder capabilities as well as the states generated by the tool still use the strand syntax.

Attack patterns may be specified using the process algebra syntax, under the label `ATTACK-PROCESS`, or strand syntax, under the label `ATTACK-STATE`. We describe how they are specified in the process algebra syntax below. An attack pattern describes a state consisting of zero or more processes that must have executed, and zero or more terms in the intruder knowledge. It may also contain *never patterns*, that is, descriptions of processes that must *not* be executed at the time the state is reached. Never patterns can be used to reason about authentication properties, e.g., can Alice execute an instance of the protocol, apparently with Bob, without Bob executing an instance of the protocol with Alice?

Note that processes in an attack pattern cannot contain explicit nondeterminism (?) or explicit deterministic choice (if), since one and only one behavior is provided in an attack pattern. This is achieve by requiring that any constraint c appearing in an attack pattern must be *strongly irreducible*, that is, it must not only be irreducible, but for any irreducible substitution σ to the variables of c, σc must be irreducible as well.

That is, imagine a process i the form

$$-(m1)\ .\ +(m2)\ .\ if\ exp1\ =\ exp2\ then\ +(m3)\ else\ +(m4)$$

where each of the expressions $exp1$ and $exp2$ can evaluate to yes or no depending on the substitutions applied to them.

Then in the attack pattern one must specify one and only one of the following possibilities

$$\begin{array}{l}
-(m1)\ .\ +(m2)\ .\ yes = yes\ .\ +(m3)\\
-(m1)\ .\ +(m2)\ .\ yes \neq no\ .\ +(m4)\\
-(m1)\ .\ +(m2)\ .\ no = no\ .\ +(m3)\\
-(m1)\ .\ +(m2)\ .\ no \neq yes\ .\ +(m4)
\end{array}$$

Finally, never patterns must satisfy a stronger condition: the entire never pattern must be strongly irreducible. This condition is inherited from the original one in Maude-NPA.

5.8.2 Choice of Encryption Type

This protocol allows either public key encryption or shared key encryption to be used by Alice to communicate with Bob. Alice initiates the conversation by sending out a message containing the chosen encryption mode, then Bob replies by sending an encrypted message containing his session key. The encryption mode is chosen nondeterministically by Alice. Therefore, it exhibits an *explicit nondeterministic choice*. Below we show the protocol description: the first one reflects the case in which public key encryption (denoted by $PubKey$) is chosen.

1. $A \rightarrow B : A; B; \mathit{PubKey}$
2. $B \rightarrow A : \mathit{pk}(A, B; SK)$
3. $A \rightarrow B : \mathit{pk}(B, A; SK; N_A\}$
4. $B \rightarrow A : \mathit{pk}(A, B; N_A)$

The second one reflects the case in which a shared key encryption (denoted by $SharedKey$) is chosen.

1. $A \rightarrow B : A; B; \mathit{SharedKey}$
2. $B \rightarrow A : \mathit{shk}(\mathit{key}(A, B), B; SK)$
3. $A \rightarrow B : \mathit{shk}(\mathit{key}(A, B), A; SK; N_A)$
4. $B \rightarrow A : \mathit{shk}(\mathit{key}(A, B), B; N_A)$

Note that A and B are names of principals, SK denotes the session key generated by B, and N_A denotes a nonce generated by A.

There are different ways of specifying this protocol as two process expressions. We have chosen to treat the encryption mode as a choice variable which can be either public key encryption or shared key encryption, and then the receiver will perform an explicit deterministic choice depending on the value of this choice variable. The process specification is as follows:

$$
\begin{array}{ll}
(\mathit{Init}) & ((+(A_? \,;\, B_? \,;\, \mathit{PubKey}) \cdot -(\mathit{pk}(A_?, B_? \,;\, SK)) \\
 & ? \\
 & (+(A_? \,;\, B_? \,;\, \mathit{SharedKey}) \cdot -(e(\mathit{key}(A_?, B_?), B_? \,;\, SK)) \\
(\mathit{Resp}) & -(A \,;\, B \,;\, \mathit{TEnc}) \cdot \\
 & \mathit{if}\ \mathit{TEnc} = \mathit{PubKey} \\
 & \quad \mathit{then}\ (+(\mathit{pk}(A, B \,;\, \mathit{skey}(A, B, r'))) \\
 & \quad \mathit{else}\ (+(e(\mathit{key}(A, B), B \,;\, \mathit{skey}(A, B, r'))))
\end{array}
$$

We analyzed whether the intruder can learn the session key generated by Bob, when either the public key encryption or shared key encryption is chosen, assuming both principals are honest.

```
--- initiator accepts session key for shared key encryption and
--- intruder learns it
  eq ATTACK-PROCESS(2)
   = -(a ; b ; mode) .
     (mode neq pubkey) .
     +(she(key(a, b), skey(b,r))) .
     -(she(key(a, b), skey(b,r) ; N)) .
     +(she(key(a, b), N))
     || skey(b,r) inI
     || nil   [nonexec] .

--- initiator accepts session key for public key encryption and
--- intruder learns it
  eq ATTACK-PROCESS(3)
   = -(a ; b ; mode) .
     (mode eq pubkey) .
     +(pk(a, b ; skey(b, r))) .
     -(pk(b, a ; skey(b,r) ; N)) .
     +(pk(a, b ; N))
     || skey(b,r) inI
     || nil  [nonexec] .
```

For this property, Maude-NPA terminated without any attack being found for any of the two attack states.

5.8.3 Rock-Paper-Scissors

To evaluate our approach on protocols with explicit deterministic choices, we have used a simple protocol which simulates the famous Rock-Paper-Scissors game, in which Alice and Bob are the two players of the game. In this game, Alice and Bob commit to each other their hand shapes, which are later on revealed to each other after both players committed their hand shapes. The result of the game is then agreed upon between the two players according to the rule: rock beats scissors, scissors beats paper and paper beats rock. They finish by verifying with each other that they both reached the same conclusion. Thus, at the end of the protocol each party should know the outcome of the game and whether or not the other party agrees to the outcome. This protocol exhibits *explicit deterministic choice*, because the result of the game depends on the evaluation of the committed hand shapes according to the game's rule. Note that this protocol also exhibits *implicit nondeterministic choice*, since the hand shape of the players are chosen by the players during the game.

The protocol proceeds as follows. First, both initiator and responder choose their hand shapes and send them to each other using a secure commitment scheme. Next, they both send each other the nonces that are necessary to open the commitments. Each of them then compares the two hand shapes and decides if the initiator wins, the responder wins, or there is a tie. The initiator then sends the responder the outcome. When the responder receives the initiator's verdict, it compares it against its own. It responds with "finished" if it agrees with the initiator and "cheater" if it doesn't. All messages are signed and encrypted, and the initiator's and responder's nonces are included in the messages concerning the outcome of the game. The actual messages sent and choices made are described in more detail below.

1. $A \rightarrow B : pk(B, sign(A, commit(N_A, X_A)))$
2. $B \rightarrow A : pk(A, sign(B, commit(N_B, X_B)))$
3. $A \rightarrow B : pk(B, sign(A, N_A))$
4. $B \rightarrow A : pk(A, sign(B, N_B))$

5.
   ```
   $if\ (X_A\ beats\ X_B)$
   $then\ R = Win$
   $else\ if\ (X_B\ beats\ X_A)$
       $then\ R = Lose$
       $else\ if\ (X_B\ =\ X_A)$
           $then\ R =\ Tie$
           $else\ nilP$
   ```
6. $A \rightarrow B : pk(B, sign(A, N_A; N_B; R))$

7.
   ```
   $if\ (R = Win \& X_A\ beats\ X_B)$
             $or\ (R\ =\ Lose\ \&\ X_B\ beats\ X_A)$
             $or\ (R\ =\ Tie\ \&\ X_A\ =\ X_B)$
       $then\ B \rightarrow A : pk(A, sign(B, N_A; N_B;\ finished\ ))$
       $else\ B \rightarrow A : pk(A, sign(B, N_A; N_B;\ cheater\ ))$
   ```

One interesting feature of the Rock-Scissors-Paper protocol is that, in order to verify that the commitment has been opened successfully, i.e., that the nonce received is the nonce used to create the commitment, one must verify that the result of opening it is well-typed, i.e., that it is equal to “rock”, “scissors”, or “paper”. This can be done via the evaluation of predicates. First, we create a sort Item and declare the constants “rock”, “scissors”, and “paper” to be of sort Item. Then we create a variable $X : Item$ of sort Item. We then define a predicate *item*? such that $item?X : Item$ evaluates to true. Since only terms of sort Item can be unified with $X : Item$, this predicate can be used to check whether or not a term is of sort Item. The process specification for the initiator and the responder is as follows.

$$
\begin{array}{ll}
(Initiator) & +(pk(B, sig(A, com(n(A,r), XA))))\ .\\
& -(pk(A, sig(B, ComXB)))\ .\\
& +(pk(B, sig(A, n(A,r))))\ .\\
& -(pk(A, sig(B, NB)))\ .\\
& (if\ ((item?\ open(NB, ComXB))\ eq\ ok)\\
& then\ if\ ((XA\ beats\ open(NB, ComXB))\ eq\ ok)\\
& \qquad then\ +(pk(B, sig(A, n(A,r)\ ;\ win)))\\
& \qquad else\ if\ ((open(NB, ComXB)\ beats\ XA)\ eq\ ok)\\
& \qquad\quad then\ +(pk(B, sig(A, n(A,r)\ ;\ lose)))\\
& \qquad\quad else\ +(pk(B, sig(A, n(A,r)\ ;\ tie)))\\
& else\ nilP)\ .\\
& -(pk(A, sig(B, n(A,r); NB))\ ;\ S{:}Status)
\end{array}
$$

$$
\begin{array}{ll}
(Responder) & -(pk(B, sig(A, ComXA)))\ .\\
& +(pk(A, sig(B, com(n(B,r), XB))))\ .\\
& -(pk(B, sig(A, NA)))\ .\\
& +(pk(A, sig(B, n(B,r))))\ .\\
& -(pk(B, sig(A, NA; R)))\ .\\
& (if\ ((item?\ open(NA, ComXA))\ eq\ ok)\ then\\
& if\ (R\ eq\ win)\\
& then\ if\ ((open(NA, ComXA)\ beats\ XB)\ eq\ ok)\\
& \qquad then\ +(pk(A, sig(B, NA; n(B,r)))\ ;\ finished)\\
& \qquad else\ +(pk(A, sig(B, NA; n(B,r)))\ ;\ cheater)
\end{array}
$$

```
else if (R eq lose)
      then if ((XB beats open(NA, ComXA)) eq ok)
          then + (pk(A, sig(B, NA; n(B, r))) ; finished)
            else + (pk(A, sig(B, NA; n(B, r))) ; cheater)
      else if (R eq tie)
            then if (XB eq open(NA, ComXA))
                then + (pk(A, sig(B, NA; n(B, r))) ; finished)
                  else + (pk(A, sig(B, NA; n(B, r))) ; cheater)
            else nilP
else nilP)
```

We first tried to see whether the protocol can simulate the game successfully, so we asked for different scenarios in which the player Alice or Bob can win in a round of the game. Maude-NPA was able to generate the expected scenarios, and it did not generate any others. We then gave Maude-NPA a secrecy attack state, in which the intruder, playing the role of initiator against an honest responder, attempts to guess its nonce before the responder receives its commitment.

```
eq ATTACK-PROCESS(1) =
   -(pk(b,sig(i, ComXA:ComMsg))) .
   +(pk(i,sig(b, com(n(b, r:Fresh), XB:Item)))) .
   -(pk(b,sig(i, NA:Nonce)))
   || n(b, r:Fresh) inI
   || nil [nonexec] .
```

Finally we specified an authentication attack state in which we asked if a responder could complete a session with an honest initiator with the conclusion that the initiator had carried out its rule faithfully, without that actually having happened.

```
eq ATTACK-PROCESS(2) =
  -(pk(b, sig(a, ComXA))) .
  +(pk(a, sig(b, com(n(b,r), XB)))) .
  -(pk(b, sig(a, NA))) .
  +(pk(a, sig(b, n(b, r)))) .
  -(pk(b, sig(a, NA ; win))) .
  ((item? open(NA, ComXA)) eq ok) .
```

```
(win eq win) .
((open(NA, ComXA) beats XB) eq ok) .
+(pk(a, sig(b, NA ; n(b,r))) ; finished)
|| empty
|| never(
   +(pk(b, sig(a, ComXA))) .
   -(pk(a, sig(b, com(n(b,r), XB)))) .
   +(pk(b, sig(a, NA))) .
   -(pk(a, sig(b, n(b,r)))) .
   (ok eq ok) .
   (ok eq ok) .
   +(pk(b, sig(a, NA ; win))) .
   -(pk(a, sig(b, NA ; n(b,r))) ; finished)
   || empty  ) [nonexec] .
```

For both of these attack states Maude-NPA finished its search without finding any attacks.

5.8.4 TLS

In Section 5.1.3 we introduced a simplified version of the handshake protocol in TLS 1.3 [92]. Even this simplified version produced a very large search space, because of the long list of messages and the concurrent interactions of a big amount of choices. We are however able to check the correctness of our specification by producing legal executions in Maude-NPA. Unlike TLS 1.3, we intentionally introduced a "downgrade attack" in our version in which the attacker can trick the principals into using a weaker crypto system. However, we have not yet been able to produce this attack because of the very deep and wide analysis tree (i.e., long reachability sequences with many branches) that is produced. We are currently investigating more efficient ways of managing list processing.

5.9 RELATED WORK

As we mentioned in the introduction, there is a considerable amount of work on adding choice to the strand space model that involves embedding it into other formal systems, including event-based models for concurrency [52], Petri nets [53], or multi-set rewriting [54]. Crazzolara and Winskel model nondeterministic choice as a form of composition, where a conflict relation is defined between possible child strands so that the parent can compose with only one potential child. In [53] Fröschle uses a Petri net model to add branching to strand space *bundles*, which represent the concurrent execution of strand space roles. Note that we have taken the opposite approach of representing bundles as traces of non-branching strands, where a different trace is generated for each choice taken. Although this results in more bundles during forward execution, it makes little difference in backwards execution, and is more straightforward to implement in an already existing analysis tool.

We also note that deterministic choice has been included in the applied pi calculus for cryptographic protocols [94], another widely used formal model, based on Milner's pi calculus [95]. The applied pi calculus includes the rule $\textit{if } M = N \textit{ then } P \textit{ else } Q$, where P and Q are terms. This is similar to our syntax for deterministic choice. However our long-term plan is to add other types or predicates as well (e.g., $M \textit{ subsumes } N$) ; indeed our approach extends to any type of predicate that can be evaluated on a ground state. Although the applied pi calculus in its original form does not include nondeterministic choice, both nondeterministic and probabilistic choice have been added in subsequent work [96].

In addition, Olarte and Valencia show in [97] how a cryptographic protocol modeling language can be expressed in their universal timed concurrent constraint programming (utcc) model, a framework that extends the timed concurrent constraint programming model to express mobility. The language does not support choice, but utcc does. It seems that it would not be difficult to extend the language to incorporate the utcc choice mechanisms.

The Tamarin protocol analysis tool [98] includes deterministic branching, which was used extensively in the analysis of TLS 1.3 [99]. In particular, it includes an optimization for roles of the form $P.(\textit{if } T \textit{ then } Q \textit{ else } R).S$; when backwards search is used, it is sometimes possible to capture such an execution in terms of just one strand until the conditional is encountered, thus reducing the state space. Our approach produces two strands, but since the process algebra semantics makes it easy to tell whether or not R behaves "essentially" the same no matter if P or Q is chosen, we believe that we have a pathway for including such a feature if desired.

5.10 CONCLUDING REMARKS

We have presented an extension to the strand space model that allows for both deterministic and nondeterministic choice, together with an operational semantics for choice in strand spaces that not only provides a formal foundation for choice, but allows us to implement it directly in the Maude-NPA cryptographic protocol analysis tool. In particular, we have applied Maude-NPA to several protocols that rely on choice in order to validate our approach. This work not only provides a choice extension to strand spaces, but extends them in other ways as well. First of all, it provides a process algebra for strand spaces. In addition, the process algebra semantics provides a new specification language for Maude-NPA that we believe is more natural for users than the current strand-space specification language.

CHAPTER 6: MODULAR VERIFICATION OF SEQUENTIAL COMPOSITION FOR PRIVATE CHANNELS

6.1 INTRODUCTION

Security protocols often depend on other protocols to generate the keys and other values they use to communicate securely. This can often be modeled in terms of protocol composition: the protocol receiving the keys runs as a subroutine of the protocol that generates the keys. But examining composed protocols can be unwieldy, leading to state space explosion.

Researchers have developed two primary ways of approaching this problem. One is to think of the parent protocol as creating *secure channels* that the child protocol can use for secure communication. Another approach is to perform modular verification, in which the two protocols are analyzed separately, and conclusions about the composition are made based on the results.

Secure channels and modular verification are closely related, and indeed we can think of them as two different kinds of decomposition, the main difference being that the channels specified in one approach are left implicit in the other. However, they have usually been treated separately. But the close relationship raises the possibility that techniques developed for one can be used for another.

In this chapter we take advantage of this relationship to make use of a construct used by Cheval, Cortier, and Le Morvan [101] to reason about secure channels to prove results about modular decomposition. This construct is called an *encapsulation*. It is a combination of different cryptographic operators that are used to provide functionality such as confidentiality and authentication. But its usefulness for modular composition lies in the fact that it is impossible for encapsulations to leak the keys used to construct them, so that any leakage that occurs must be the fault of the protocol providing the keys. We can thus think of an encapsulation together with the keys used to implement it as providing a secure communication channel, although unlike [101], we do not define the properties of the channels explicitly; rather they are verified implicitly by the separate analyses of the parent protocol and the child protocol.

Example 6.1. We use the Passive Authentication (PA) protocol [101] as a running example, which provides an authentication mechanism proving that the content of the RFID chip in an E-passport is authentic. We describe below the PA protocol, between a passport (B) and a

[1]This chapter is based on the paper [100], joint work with Santiago Escobar, Catherine A. Meadows and José Meseguer.

reader (A). A secure channel between principals A and B is represented as an encapsulation expression $\mathcal{E}(M, \overrightarrow{K})$ where M is the payload message and $\overrightarrow{K}$ is a list of yet unspecified keys.

$$\begin{aligned}
&A \rightarrow B : \mathcal{E}(read, \overrightarrow{K}) \\
&B \rightarrow A : \mathcal{E}(data; sign(h(data), sk(B)), \overrightarrow{K})
\end{aligned}$$

In this protocol, $data$ is the passport information, h is a hash function, $sign(M, K)$ is signing message M with key K, and $sk(B)$ is the signing key of B.

A protocol using channels is parametric in several ways: (i) what keys $\overrightarrow{K}$ are actually needed by such encapsulation, and (ii) how is the encapsulation $\mathcal{E}$ actually obtained using cryptographic primitives. In [101], the encapsulation associated to protocol PA is achieved by using symmetric encryption and MAC with two respective keys, K_1 and K_2. The encapsulation can be seen as a macro:

$$\mathcal{E}(M, K_1, K_2) = senc(M, K_1); mac(senc(M, K_1), K_2).$$

To obtain the actual keys K_1 and K_2, we first run a key establishment protocol that is sequentially composed with PA. The cryptographic protocol analysis tool Maude-NPA [60] was extended in [48] with a specification language, a semantics, and automatic verification methods that support sequential protocol composition. We use this extension in this chapter.

Example 6.2. Let us consider the Basic Access Control (BAC) protocol [101] for access control on private data, which can be used as the key establishment protocol for the PA protocol.

$$\begin{aligned}
&A \rightarrow B : challenge \\
&B \rightarrow A : N_B \\
&A \rightarrow B : senc(N_A; N_B; K_A, K_e), mac(senc(N_A; N_B; K_A, K_e), K_m) \\
&B \rightarrow A : senc(N_B; N_A; K_B, K_e), mac(senc(N_B; N_A; K_B, K_e), K_m)
\end{aligned}$$

where $senc(M, K_e)$ denotes symmetric encryption of message M using a shared key K_e and $mac(M, K_m)$ denotes MAC using a shared key K_m.

The sequential protocol composition $BAC\,;\ PA$ is then achieved by running one protocol after the other and passing information from BAC to PA.

Our contributions. The main contribution of this chapter is a *modular verification* methodology. Given parametric specifications of a key establishment protocol P and a protocol Q providing private channel communication, security and authenticity properties of their sequential composition $P\ ;\ Q$ can be reduced to: (i) verification of corresponding properties for P, and (ii) verification of corresponding properties for an *abstract version* Q^{α} of Q in which keys inherited by Q from P have been suitably abstracted. The semantic basis of this methodology is provided by two *simulation relations* H_P and H_Q of the form $P \xleftarrow{H_P} P\ ;\ Q \xrightarrow{H_Q} Q^{\alpha}$ that essentially "project out" states of $P\ ;\ Q$ to, respectively, states of P and states of Q^{α}. This then ensures that given an attack state for $P\ ;\ Q$, if no attack of types (i), resp. (ii), exists for P, resp. Q, no such attack of the specified kind exists for $P\ ;\ Q$. In addition we offer tool support via the Maude-NPA cryptographic protocol analysis tool [60]. Furthermore, we are able to easily handle algebraic properties of P, Q, and the encapsulation $\mathcal{E}$, (e.g. Diffie-Hellman exponentiation used in the IKEv1 protocol; see Section 6.6) as long as they satisfy the *finite variant property*, which is also supported by Maude-NPA.

Plan of the chapter. The rest of this chapter is organized as follows. In Section 6.2 we recall protocol composition in Maude-NPA, and present the specification of the motivating example. In Section 6.3 we present an approach to the modeling of channels with security assumptions in Maude-NPA. In Section 6.4 we present a modular method for reducing reachability properties of $P;Q$ to corresponding properties for P and Q's abstraction Q^{α}. In Section 6.5 we explain how this modular methodology is supported by Maude-NPA. In Section 6.6 we present some example protocols we analyzed using our modular approach. We discuss some related work in Section 6.7 and conclude in Section 6.8.

6.2 PROTOCOL COMPOSITION IN MAUDE-NPA

Recall that, to support sequential protocol composition, strands can be extended with *synchronization messages* that are intended for reasoning about child protocols that use information received from parent protocols, e.g. a session key establishment protocol that uses a master key received from a master key establishment protocol. Intuitively, sequential composition of two strands describes a situation in which one strand (the *child*), can only execute after another strand (*the parent*) has completed its execution. Each composition of two strands is obtained by *matching the output parameters of the parent strand with the input parameters of the child strand* in a user-specified way.

Example 6.3. The PA protocol of Example 6.1 is specified in Maude-NPA by two strands associated to the reader (A) and passport (B) roles. To be composable with a previous key establishment protocol, these strands have *input parameters* (inside a synchronization message's curly braces). The passport strand is as follows, where Prev2 is a variable of sort Role leaving the parent strand unspecified:

$$
\begin{aligned}
(PA.B) :: r' :: [&nil \mid \{Prev2 \to PA.B \;;;\; 1{-}1 \;;;\; A; B; K_1; K_2\},\\
&-(\mathcal{E}(read, K_1, K_2)),\\
&+(\mathcal{E}(data(B, r'); sign(h(data(B, r')), sk(B)), K_1, K_2)), nil]
\end{aligned}
$$

Example 6.4. The BAC protocol of Example 6.2 is specified in Maude-NPA by two strands associated to the reader (A) and passport (B) roles. To be usable for sequential composition with another protocol, these strands have *output parameters* (inside a synchronization message's curly braces). The strand associated to the passport (B) is as follows:

$$
\begin{aligned}
(BAC.B) :: r, r_2 :: \;\; [&nil \mid\\
&-(challenge),\; +(n(B, r)),\\
&-(senc(N_A; n(B, r); K_A, ke(B, A));\\
&\qquad mac(senc(N_R; n(B, r); K_A, ke(B, A)), km(B, A))),\\
&+(senc(n(B, r); N_A; key(B, r_2), ke(B, A));\\
&\qquad mac(senc(n(B, r); N_A; key(B, r_2), ke(B, A)), km(B, A))),\\
&\{BAC.B \to PA.B \;;;\; 1{-}1 \;;;\; A; B;\\
&\quad f_1(K_A, key(B, r_2)); f_2(K_A, key(B, r_2))\}, nil]
\end{aligned}
$$

Example 6.5. Given the protocol specification of PA in Example 6.3 and of BAC in Example 6.4, both containing synchronization messages, a concrete state where a strand of the passport role in BAC has been synchronized with a strand of the passport role in PA looks as follows:

$$
\begin{aligned}
&(BAC.B)\\
&[\ldots, \{BAC.B{\to}PA.B;;1{-}1;;A;B;f_1(K_A, key(B, r_2)); f_2(K_A, key(B, r_2))\} \mid nil]\ \&\\
&(PA.B)\\
&[nil, \{BAC.B{\to}PA.B;;1{-}1;;A;B;f_1(K_A, key(B, r_2)); f_2(K_A, key(B, r_2))\} \mid \ldots]
\end{aligned}
$$

6.3 PROTOCOL SPECIFICATION WITH ENCAPSULATIONS

In this section we define the specification of secure channels using encapsulations with protocol composition. The specification consists of the composition of two main protocols: (i) a key establishment protocol (P) as the parent protocol, and (ii) the key application protocol (Q) as the child protocol. Private communication in protocol Q is achieved by means of encapsulations (patterns constructed with cryptographic primitives) taking advantage of the keys established by protocol P. The relation between the protocol P and protocol Q is defined by the map ρ^P, which relates the principals in Q to a *list of keys* that are established in P. The map ρ^P defines the interface for composing protocols P and Q. We will first show in Section 6.3.1 the specification of the protocol Q with encapsulation. The map ρ^P is defined in Section 6.3.2. In Section 6.3.3, we define the admissible composition of protocols P and Q.

Notation. In the rest of this chapter we will use S to denote states, SS to denote the set of strands in a state, and IK to denote the intruder knowledge set in a state. Different subscripts are used to distinguish states, strands and intruder knowledge of different protocols. The subscript ; (resp. P, resp. Q^α) denotes protocol $P;_\rho Q$ (resp. P, resp. Q^α). For example, $S_;$ denotes a state of protocol $P;_\rho Q$. Given a set of strands SS, we denote by $SS|_Q$ and $SS|_P$ the strands in the strand set SS that are (partial) instances of strands in the protocol Q and protocol P respectively. $SS|_{HS}$ denotes the strands in SS that are instances of honest protocol strands (without the input/output parameters), and $SS|_{DY}$ denotes the strands in SS that are instances of Dolev-Yao strands.

6.3.1 Protocol Q with Encapsulations

In this section, to specify and analyze protocols using encapsulations in Maude-NPA, we add special sorts and operations to the Maude-NPA message notation that we introduced in Section 6.2. Messages sent and received through encapsulations are denoted by terms constructed using these special sorts and operations.

A protocol Q providing private communication is such that all input and output messages in a strand are actually sent through (possibly different) encapsulations. Its specification has the form $Q = ((\Sigma_Q, E_Q), SSpec_Q)$, where the signature Σ_Q is a disjoint union of $\Sigma_{Ch} \uplus \Sigma_{Q_0}$ and where:

1. Σ_{Ch} is the signature of *encapsulation operators* used in the private channels, which

have the general form:

$$\mathcal{E} : \mathsf{QMsg} \times \mathsf{QKey} \times \cdots \times \mathsf{QKey} \to \mathcal{E}\mathsf{Msg}$$

where QMsg denotes the sort of *payload messages* sent through the encapsulation, QKey denotes the sort of *keys* used in the encapsulation, and $\mathcal{E}\mathsf{Msg}$ is the sort of *encapsulated messages*, representing message transmission. There is also a general sort Msg of Maude-NPA for messages, and subsort inclusions $\mathsf{QMsg} < \mathcal{E}\mathsf{Msg} < \mathsf{Msg}$ and $\mathsf{QKey} < \mathsf{Msg}$. Note that the encapsulation is parameterized by sort QKey.

2. Σ_{Q_0} is the signature of *cryptographic functions* used to *actually achieve the encapsulation*, i.e., to give concrete meaning to the operator $\mathcal{E}$ in Σ_{Ch}. All operators in Σ_{Q_0} are of the general form $f : \mathsf{S_1} \times \cdots \times \mathsf{S_n} \to \mathcal{E}\mathsf{Msg}$, where $\{\mathsf{S_1}, \ldots, \mathsf{S_n}\} \subseteq \{\mathsf{QKey}, \mathcal{E}\mathsf{Msg}\}$.

3. E_Q is a disjoint union $E_Q = E_{\mathcal{E}} \uplus E_{Q_0} \uplus B_{Q_0}$ where $E_{\mathcal{E}}$ are the *definitional extensions* of the operators $\mathcal{E}$ in Σ_{Ch}. These are defined as equations having the form: $\mathcal{E}(M, X_1, \ldots, X_n) = t(M, X_1, \ldots, X_n)$ with $t(M, X_1, \ldots, X_n)$ being a Σ_{Q_0}-term of sort $\mathcal{E}\mathsf{Msg}$. B_{Q_0} is a set of Σ_{Q_0}-axioms and E_{Q_0} are Σ_0-equations that are convergent (confluent and terminating) modulo B_{Q_0} and have finite variant property.

The encapsulation equations $E_{\mathcal{E}}$ define how cryptographic primitives are used to implement private communication. We assume that the protocol Q is specified using protocol composition to define how the extra parameters of an encapsulation (principal identifiers and keys) are bound.

Example 6.6. In Example 6.3, the encapsulation symbol $\mathcal{E} : \mathsf{QMsg} \times \mathsf{QKey} \times \mathsf{QKey} \to \mathcal{E}\mathsf{Msg}$ for a secure communication is defined as the definitional extension $\mathcal{E}(M, K_1, K_2) = senc(M, K_1); mac(senc(M, K_1), K_2)$ in $E_{\mathcal{E}}$. The two keys K_1 and K_2 are generated by a key establishment protocol for the passport (B) and the reader (A). The signature Σ_{Ch} contains the encapsulation operator $\mathcal{E}$, and Σ_{Q_0} contains the operator *senc* for symmetric encryption and the operator *mac* for MAC. The encryption/decryption cancellation of *senc* is represented in E_{Q_0}.

We assume the following *admissibility requirements* on the *definitional extension* equations $E_{\mathcal{E}}$ of the encapsulation operators $\mathcal{E}$:

- certain positions in the term $t_{\mathcal{E}}$ associated by $E_{\mathcal{E}}$ to the encapsulation operator $\mathcal{E}$ are designated as *non-payload* positions and are not of sort QMsg;

- it is impossible for the intruder to learn a term that only appears in non-payload positions in such a term $t_{\mathcal{E}}$, and
- QKey is not a subsort of QMsg, and expressions of sort QKey only appear in non-payload positions in $t_{\mathcal{E}}$.

For example, in $senc(M, K_1); mac(senc(M, K_1), K_2)$, the non-payload positions are 1.2, 2.1.2, and 2.2, i.e., the positions where K_1 and K_2 occur.

6.3.2 The Key Implementation Map ρ

We assume that a *key establishment protocol* P in the composition $P;_\rho Q$ is specified as $P = ((\Sigma_P, E_P \cup B_P), SSpec_P)$, where Σ_P is the signature of protocol P. All message sorts in P are subsorts of PMsg, including QKey. QKey denotes the sort for keys that are being established in protocol P and will be used for implementing channels in protocol Q. Note that terms of sort QKey are therefore allowed to show up in any messages in protocol P. $E_P \cup B_P$ defines the algebraic properties of protocol P. $SSpec_P$ is a set of strands specifying both the capabilities of the attacker and the behavior of the honest principals.

Given a protocol Q, from the actual protocol composition P ; Q with a key establishment protocol P it is possible to derive a *key implementation map* ρ^P that defines the list of keys established by the protocol P. Indeed, we will denote a protocol composition as $P \;;_\rho\; Q$ to emphasize the specific key map ρ. The keys are assigned considering not only the type of desired communication, but also the names and roles of the intended sender and the receiver of each protocol. More specifically, a key implementation map ρ^P of protocol P is an indexed family of mappings $\rho^P = \{P_{Role} \in \mathcal{T}_{\Sigma_{P,\mathsf{Role}}} \mid \rho^{P_{Role}}\}$ where $\rho^{P_{Role}}$ denotes the keys established by the role P_{Role}. Each $\rho^{P_{Role}}$ takes two arguments: the first argument is a triple denoting the sender's name, sender's role in protocol P and sender's role in protocol Q; the second argument is a triple denoting the receiver's name, receiver's role in protocol P and receiver's role in protocol Q. To simplify the presentation, we will refer to the triple as $\widehat{A}$ with A denoting a name when the roles are understood from the context.

Example 6.7. Following Example 6.2, the key implementation map is:

$$\begin{aligned}
&\rho^{BAC.B}((B, BAC.B, PA.B), (A, BAC.A, PA.A)) \\
&\qquad = \{K^{PA.B}_{1_{B,A}} \mapsto f_1(K_A, key(B, r_2)), K^{PA.B}_{2_{B,A}} \mapsto f_2(K_A, key(B, r_2))\}, \\
&\rho^{BAC.A}((B, BAC.B, PA.B), (A, BAC.A, PA.A)) \\
&\qquad = \{K^{PA.A}_{1_{B,A}} \mapsto f_1(key(A, r_1), K_B),\ K^{PA.A}_{2_{B,A}} \mapsto f_2(key(A, r_1), K_B)\}
\end{aligned}$$

where A and B are variables denoting the reader and passport respectively, and K_A, K_B are variables denoting keys. $BAC.A, PA.A, BAC.B, PA.B$ are constants denoting roles. Although the keys in $\rho^{BAC.B}$ and $\rho^{BAC.A}$ are syntactically different, keys can have common instances, e.g. $f_1(key(A, r_1), key(B, r_2))$, which captures the idea that different roles may have different views of the same message.

We assume the following *admissibility requirements* on ρ:

- **Q keys are actual P keys of sort QKey.** For any $\rho^{Rol_P}(\widehat{A}, \widehat{B}) = \{\overrightarrow{X}^{Rol_Q}_{A,B} \mapsto \overrightarrow{K}\}$, $Var(\overrightarrow{K}) \subseteq Var(\mathcal{P}_P|_{Rol_P})$, where $\overrightarrow{K}$ denotes a list of QKeys.
- **Keys must include some random value of sort Fresh.** For any $\rho^{Rol_P}(\widehat{A}, \widehat{B}) = \{\overrightarrow{X}^{Rol_Q}_{A,B} \mapsto \overrightarrow{K}\}$, $\forall K_i \in \overrightarrow{K}$, there exists $r{:}\mathsf{Fresh}$ s.t. $r \in Var(K_i)$.
- **Keys are disjoint.** No two keys bound by ρ^{Rol_P} are unifiable.
- **ρ is injective.** For $\rho^{Rol_P}(\widehat{A}, \widehat{B}) = \{\overrightarrow{X}^{Rol_Q}_{A,B} \mapsto \overrightarrow{K}\}$ and $\rho^{Rol_P{}'}(\widehat{A'}, \widehat{B'}) = \{\overrightarrow{X}^{Rol_Q{}'}_{A',B'} \mapsto \overrightarrow{K'}\}$, we require that there exists a substitution θ such that $(\widehat{A}, \widehat{B})\theta = (\widehat{A'}, \widehat{B'})\theta$ iff there exists σ such that $\overrightarrow{K}\theta\sigma =_{E_P \cup B_P} \overrightarrow{K'}\theta\sigma$ (resp. $(\overrightarrow{K}\theta\sigma)^{-1} =_{E_P \cup B_P} \overrightarrow{K'}\theta\sigma$ for asymmetric keys).

Since the key implementation map is derived from the protocol composition, the admissibility requirement that Q keys are actual P keys of sort QKey is based on the requirements on sequential protocol composition. Also because the keys passed from P to Q are intended for implementing private channels, we thus consider it a good practice to ensure the uniqueness of keys generated in different sessions of the key exchange protocol and the disjointness of keys used for different purpose or shared between different principals.

Note that the mapping ρ can be made more precise by performing analysis in protocol P, as long as it covers all possible keys that can be synchronized from protocol P to Q in a reachable state. In order to allow all possible behaviors, key terms used in protocol specifications can often be quite general. If it can be later verified by performing analysis in P that in reality only certain key patterns can be passed to protocol Q, we then can consider only these more concrete key patterns when analyzing protocol Q.

The encapsulation equations $E_{\mathcal{E}}$ and the above map ρ^P jointly define a translation function:

$$(\mathcal{E}, \rho^P) : \mathcal{T}_{\Sigma_Q}(X)_{\mathsf{ChMsg}} \to \mathcal{T}_{\Sigma_{\mathsf{QMsg}} \cup \Sigma_{\mathsf{QKey}} \cup \Sigma_{Q_0}}(X)$$

which models the actual composition of all the strands involved between the key establishment protocol P and the protocol Q with private communication

$$\mathcal{E}(M, {X_1}_{A,B}^{Rol_Q}, \ldots, {X_{n_{ch_{op}}}}_{A,B}^{Rol_Q})\rho_{ch_{op}}^{Rol_P}(\widehat{A}, \widehat{B})$$

The key implementation formalizes the synchronization of parameters in the sequential protocol composition $P;_\rho Q$ in Maude-NPA. The keys generated by protocol P are transferred to the key parameters in protocol Q by synchronizing the output parameters of the parent protocol P and input parameters of child protocol Q according to ρ^P.

Example 6.8. The input/output parameters of the composed protocol $BAC;_\rho PA$ are as follows:

$$\begin{aligned}
&(BAC.B)\ [nil \mid \ \ldots, \{BAC.B \to PA.B; ; 1-1; ; A; B;\\
&\qquad\qquad f_1(Kr, key(B, r_2)); f_2(Kr, key(B, r_2))\}, nil]\ \&\\
&(PA.B)\ [nil \mid \{BAC.B \to PA.B; ; 1-1; ; A; B; K_1; K_2\}, \ldots, nil]
\end{aligned}$$

6.3.3 The Admissible Composition $P;_\rho Q$

The specification of the composed protocol $P;_\rho Q$ is defined as: $P;_\rho Q = ((\Sigma_{P;_\rho Q}, E_{P;_\rho Q} \cup B_{P;_\rho Q}), SSpec_{P;_\rho Q})$ where $\Sigma_{P;_\rho Q} = \Sigma_Q \cup \Sigma_P$, $E_{P;_\rho Q} = E_Q \cup E_P$, $B_{P;_\rho Q} = B_Q \cup B_P$, $SSpec_{P;_\rho Q} = SSpec_P \cup SSpec_Q$. The synchronization of input/output parameters of honest strands are captured in the key implementation map ρ. To simplify the notation, we will use $E_;$ for $E_{P;_\rho Q}$ and $B_;$ for $B_{P;_\rho Q}$. We will refer to the child protocol in the composition $P;_\rho Q$ as Q^ρ in the rest of this chapter.

We summarize the *syntactic disjointness assumptions* between protocols P and Q^ρ implied by this setup as follows:

- all the messages in honest protocol strands of protocol Q^ρ are encapsulated, i.e., are of sort $\mathcal{E}\mathsf{Msg}$, and all messages in protocol P are of a sort PMsg or subsorts of a sort PMsg.

- all the messages that can be shared by protocol P and Q^ρ must be of sorts QKey, a new sort Shared or subsorts of Shared. The sort Shared is introduced to denotes the messages in $\Sigma_{\mathsf{PMsg}} \cap \Sigma_{\mathsf{QMsg}}$. And we assume that P and Q^ρ agree on the shared messages, i.e., $(\Sigma_{\mathsf{Shared}},$
$E_{\mathsf{Shared}} \cup B_{\mathsf{Shared}}) = (\Sigma_P, E_P \cup B_P) \cap (\Sigma_{\mathsf{QMsg}}, E_{\mathsf{QMsg}} \cup B_{\mathsf{QMsg}})$. We assume that $\mathsf{Shared} < \mathsf{Public}$.

Note that QKey is not a subsort of $\mathcal{E}\mathsf{Msg}$. Therefore the keys generated by protocol P cannot be unified with encapsulations or payload messages in Q^ρ, i.e., messages of sort QMsg. By the disjointness feature of this setup, all information synchronized through input-output parameters in the protocol $P;_\rho Q$ are of sort QKey, Shared or subsort of Shared. This setup is inspired by some of the common assumptions: (i) in the child protocol of $P;_\rho Q$, the keys used to implement encapsulations will never be sent in plain text or as a payload message; (ii) protocols P, Q and encapsulations in $\mathcal{E}$ have disjoint encryption subterms.

Remark 6.1. As usual in Maude-NPA, we do not fix the primitives that can be used in either P or Q. Moreover, P and Q do not have to use disjoint cryptographic primitives, as long as the operators can be correctly overloaded.

Definition 6.1 (Admissible Composition)**.** A composition $P;_\rho Q$ is *admissible* if the protocol Q, the key implementation ρ and the protocol P satisfy the requirements in Section 6.3.1, 6.3.2 and 6.3.3 respectively.

In the rest of this chapter, we assume that the composed protocol $P;_\rho Q$ is *admissible.*

6.4 BEHAVIORAL DECOMPOSITION OF $P;_\rho Q$

In this section we show that the admissible composed protocol $P;_\rho Q$ can be analyzed by analyzing the protocols P and Q^α (an abstraction of Q^ρ) separately. Two *simulation relations* H_P and H_Q are defined and proved in Sections 6.4.1 and 6.4.2, respectively. The reachable states in protocol $P;_\rho Q$ are related to reachable states of protocol P and reachable states of protocol Q^α via H_P and H_Q, respectively. Therefore, for any reachable attack state in protocol $P;_\rho Q$, there are corresponding attack states that are reachable in protocols P and Q^α. This then ensures that given an attack state for $P;_\rho Q$, if no attack exists for P, resp. Q^α, then no such attack exists for $P;_\rho Q$.

6.4.1 Simulation of $P;_\rho Q$ by P

In this section we show that, given an admissible composed protocol $P;_\rho Q$, the protocol P can simulate protocol $P;_\rho Q$, so that any QKey can be generated by $P;_\rho Q$ if and only if it can be generated by P.

We first define the simulation relation below. The relation $\mathcal{H}_P$ essentially projects out from a state in the protocol $P;_\rho Q$ the strands that are instances of protocol P and the subset of the intruder knowledge that is in the signature of protocol P. Note that we only consider states that are reachable from initial states.

Definition 6.2 (Relation $\mathcal{H}_P$)**.** Given an admissible composed protocol $P;_\rho Q$ and the key establishment protocol P in $P;_\rho Q$, let $S_; = SS_; \,\&\, IK_;$ denote a state that is reachable from the initial state in protocol $P;_\rho Q$, and $S_P = SS_P \,\&\, IK_P$ denote a state that is reachable from the initial state in protocol P, the relation $\mathcal{H}_P$ between reachable states in $P;_\rho Q$ and in P is defined by the definitional equivalence $(S_;, S_P) \in \mathcal{H}_P$ if and only if:

(i) $SS_;|_P|_{HS} =_{E_P \cup B_P} SS_P|_{HS}$, and $SS_;|_P|_{DY} \subseteq_{E_P \cup B_P} SS_P|_{DY}$,

(ii) $IK_;|_{\Sigma_P} =_{E_P \cup B_P} IK_P$

We prove below that the relation $\mathcal{H}_P$ is indeed a simulation relation. Before giving the simulation proofs, we first show in the following lemma that the protocol decomposition preserves the equality relation of terms in the signature Σ_P, which is essential for the simulation results below.

Lemma 6.1. *Given an admissible composed protocol $P;_\rho Q$ and the key establishment protocol P in $P;_\rho Q$, for any two terms $t, t' \in \mathcal{T}_{\Sigma_P}(X)$, $t =_{E_P \cup B_P} t'$ iff $t =_{E_; \cup B_;} t'$.*

Proof. We recall that $E_; = E'_P \uplus E_{\mathsf{Shared}} \uplus E'_Q$, $B_; = B'_P \uplus B_{\mathsf{Shared}} \uplus B'_Q$, where $E_P = E'_P \uplus E_{\mathsf{Shared}}$, $E_Q = E_{\mathsf{Shared}} \uplus E'_Q$, $B_P = B'_P \uplus B_{\mathsf{Shared}}$, $B_Q = B_{\mathsf{Shared}} \uplus B'_Q$. By the disjointness of Σ_Q and Σ_P, for any rule $l \to r$ in $E_;$, if there exists a position pos in t, axioms $B \subseteq B_;$, and substitution σ such that $t|_{pos} =_B l\sigma$, then $l \to r \in E_P$, $B \subseteq B_P$ and $t[r\sigma]_{pos} \in \mathcal{T}_{\Sigma_P}(X)$. We therefore have that $t \to_{E_P,B_P} t_1$ iff $t \to_{E_;,B_;} t_1$. Together with the coherence, confluence and termination of $\to_{E_P,B_P}$ and $\to_{E_;,B_;}$, by successively applying the above argument, we have $t!_{E_P,B_P} =_{B_P} t!_{E_;,B_;}$. Therefore $t!_{E_P,B_P} =_{B_P} t'!_{E_P,B_P}$ iff $t!_{E_;,B_;} =_{B_;} t'!_{E_;,B_;}$, which implies $t =_{E_P \cup B_P} t'$ iff $t =_{E_; \cup B_;} t'$.

Corollary 6.1. *Given an admissible composed protocol $P;_\rho Q$ and the key establishment protocol P in $P;_\rho Q$, we have the following equality between sets of terms:*

$$\{t \in \mathcal{T}_{\Sigma_{P;_\rho Q}}(X) \mid t =_{E_; \cup B_;} t' \text{ and } t' : \mathsf{QKey}\}$$
$$=\{t \in \mathcal{T}_{\Sigma_{P;_\rho Q}}(X) \mid t =_{E_P \cup B_P} t' \text{ and } t' : \mathsf{QKey}\}$$

We refer to this set as $[T_{\mathsf{QKey}}]_{E_P \cup B_P}$ in the rest of this chapter.

We prove that $\mathcal{H}_P$ is a simulation in the following theorem. The intuition that the relation $\mathcal{H}_P$ is a simulation is that, by the disjointness assumption, terms generated by protocol Q^ρ cannot be used in (i.e., cannot match) any messages of protocol P, except for terms of sorts QKey, and Shared or subsorts of Shared. By the protocol construction, Q^ρ cannot reveal QKey, and the shared messages can be generated by Dolev-Yao transitions in protocol P.

Theorem 6.1 (Soundness from $P;_\rho Q$ to P)**.** *Given an admissible composed protocol $P;_\rho Q$, the key establishment protocol P, and the key application protocol Q, if a state $S_;$ is reachable from the initial state of protocol $P;_\rho Q$, then there exists a state S_P that is reachable from the initial state in protocol P such that $(S_;, S_P) \in \mathcal{H}_P$.*

Proof. We prove this theorem by induction on the number of steps that are taken to reach state $S_;$. Suppose $S_{;_{init}} \rightarrow^n S_;$ with n being the minimum number of steps to reach $S_;$. We prove that there exists a state S_P such that $S_{P_{init}} \rightarrow^* S_P$ and $(S_;, S_P) \in \mathcal{H}_P$.

Base case. $n = 0$: Since the initial states of both systems consist of the empty set of strands and the empty set of intruder knowledge, proving that $(S_{;_{init}}, S_{P_{init}}) \in \mathcal{H}_P$ is straightforward.

Induction Step. Suppose that the theorem is true for $n = k$. We show that the theorem is true for $n = k + 1$. Let $S'_;$ be the state such that $S_{;_{init}} \rightarrow^k S'_; \rightarrow_{rl} S_;$. Then, according to the induction hypothesis, there exists a state S'_P such that $S_{P_{init}} \rightarrow^* S'_P$ and $(S'_;, S'_P) \in \mathcal{H}_P$. We then prove by case analysis on the transition rule rl that is applied in the $k + 1$-th induction step that there exists a state S_P such that $S_{P_{init}} \rightarrow^* S_P$ and $(S_;, S_P) \in \mathcal{H}_P$.

1. If rl is of the form: $[L]\&SS\&\{IK\} \rightarrow [L, +M]\&SS\&\{IK, M\}$ if $M \notin IK$, and rl is generated from a strand in protocol specification of P, then there exists a strand $[l] \in S'_;$, the strand $[l, +m] \in S_;$, a strand $[L, +M, L']$ in protocol specification of P, and a ground substitution σ such that $[l] =_{E_; \cup B_;} [L]\sigma$, and $m =_{E_; \cup B_;} M\sigma$. Since $[L, +M, L']$ is a strand in the protocol specification of P, all variables in $[L, +M, L']$ have sorts in Σ_P. Also, by the disjointness of the signatures of protocol P and Q, the range of the ground substitution σ only contains terms having sorts in $\mathcal{T}_{\Sigma_P}$. Since $[l] \in S'_;$ and $(S'_;, S'_P) \in \mathcal{H}_P$, according to the definition of the relation $\mathcal{H}_P$, there exists a strand $[l_p]$ in the state S'_P such that $[l_p] =_{E_P \cup B_P} [l]$. Since $[l] =_{E_; \cup B_;} [L]\sigma$, by Lemma 6.1, $[l_p] =_{E_P \cup B_P} [L]\sigma$. Also by Lemma 6.1, $m =_{E_p \cup B_p} M\sigma$. Since m is not in the intruder knowledge of state $S'_;$, according to the definition of the relation $\mathcal{H}_P$, m is not in the intruder knowledge of state S'_P. Thus $S'_P \rightarrow S_P$ by applying the rule rl with the ground substitution σ, and therefore $(S_;, S_P) \in \mathcal{H}_P$.

2. The cases where rl is generated according to a strand in protocol P and is of the form $[L]\&SS\&\{IK\} \rightarrow [L, +m]\&SS\&\{IK\}$, or $SS\&\{IK\} \rightarrow [+M]\&SS\&\{IK, M\}$, or $SS\&\{IK\} \rightarrow [+M]\&SS\&\{IK\}$ can be proved in a way similar to the previous case.

3. If rl is of the form: $[L]\&SS\&\{IK, M\} \rightarrow [L, -M]\&SS\&\{IK, M\}$, and rl is generated from a strand in protocol P, then there exists a strand $[l] \in S'_;$, a message m in the intruder knowledge of state $S'_;$, a strand $[L, -M, L']$ in protocol P, and a ground substitution σ such that $[l] =_{E_; \cup B_;} [L]\sigma$ and $m =_{E_; \cup B_;} M\sigma$. Since $[L, -M, L']$ is

a strand in the protocol P, all variables in $[L, -M, L']$ having sorts in Σ_P. Also, by the disjointness of the signatures of protocol P and Q, the range of the ground substitution σ only contains ground terms in Σ_P. Since $[l] \in S'_{;}$ and $(S'_{;}, S'_P) \in \mathcal{H}_P$, according to the definition of the relation $\mathcal{H}_P$, there exists a strand $[l_p]$ in S'_P such that $[l_p] =_{E_P \cup B_P} [l]$. Since $m \in \mathcal{T}_{\Sigma_P}$ and is in the intruder knowledge of state $S'_{;}$, according to the definition of the relation $\mathcal{H}_P$, there exists m' in the intruder knowledge of state S'_P such that $m =_{E_P \cup B_P} m'$. Since $[l] =_{E_{;} \cup B_{;}} [L]\sigma$ and $m =_{E_{;} \cup B_{;}} M\sigma$, by Lemma 6.1, $l_p =_{E_P \cup B_P} L\sigma$ and $m' =_{E_P \cup B_P} M\sigma$. Thus $S'_P \to S_P$ by applying the rule rl with the ground substitution σ, and therefore $(S_{;}, S_P) \in \mathcal{H}_P$.

4. The case where rl is generated according to a strand in protocol P and is of the form $SS\&\{IK, M\} \to [-M]\&SS\&\{IK, M\}$ can be proved in a way similar to the previous case.

5. If rl is of the form: $[L]\&SS\&\{IK\} \to [L, +M]\&SS\&\{IK, M\}$ if $M \notin IK$, and rl is generated from a strand in protocol Q, then there exists a term m that is in the intruder knowledge of the state $S_{;}$ but not in $S'_{;}$, such that $m =_{E_{;} \cup B_{;}} M\theta$ for some ground substitution θ.

    - If $m =_{E_{;} \cup B_{;}} m'$ with m' of sort Shared or any subsort of Shared, according to the assumption that sort Shared is a subsort of sort Public, m' can be generated by the intruder itself (i.e., Dolev-Yao strands) in protocol P. Therefore $S'_P \to_{DY} S_P$ with m' in the intruder knowledge of state S_P, and the transition uses only Dolev-Yao strands. Thus $(S_{;}, S_P) \in \mathcal{H}_P$.
    - Otherwise $(S_{;}, S'_P) \in \mathcal{H}_P$.

6. For other cases, $(S_{;}, S'_P) \in \mathcal{H}_P$. The synchronization steps are covered by this case.

It is now straightforward to show that the admissible composed protocol $P;_\rho Q$ can simulate protocol P.

Theorem 6.2 (Completeness from $P;_\rho Q$ to P)**.** *Given an admissible composed protocol $P;_\rho Q$, the key establishment protocol P, and the key application protocol Q, if a state S is reachable from the initial state in protocol P, the state S is also reachable from the initial state of protocol $P;_\rho Q$.*

Proof. Since the protocol P is the parent protocol of the protocol $P ;_\rho Q$, the protocol $P ;_\rho Q$ can perform exactly the same sequence of transitions from the initial state that the protocol P performs to reach state S.

By Theorems 6.1 and 6.2, we can conclude that any authentication or secrecy property of the QKeys holds in P if and only if it holds in $P ;_\rho Q$. In particular, this holds for the following *secrecy enforcing* property.

Definition 6.3. A protocol P is *Secrecy-Enforcing* if any state where an honest principal has accepted a list of QKeys to communicate with another honest principal and the intruder learns any of the secret QKeys is unreachable in P.

That is, the state $S_P = SS_P \& IK_P$ cannot be reached from the initial states in protocol P, if there exists a strand $[l]$ in the set of strands SS_P in S_P such that $[l]$ is an instance of honest protocol strands in protocol P, the participants in $[l]$ do not include the intruder, but the set of intruder knowledge IK_P is not disjoint with the set of secret QKeys generated from $[l]$.

In the rest of this chapter, we assume that the key establishment protocol P is *Secrecy-Enforcing*. Note that the intruder can still learn keys from communications in P, either because some of the participants of the protocol are compromised or because the keys were not intended to be secret in the first place.

6.4.2 Simulation of $P ;_\rho Q$ by Q^α

In this section we show that the reachability properties of the protocol $P ;_\rho Q$ can be simulated by an abstraction of Q^ρ, denoted Q^α, where actual keys from an admissible P are replaced by their abstraction using a mapping α. The inherited keys become in some sense independent of the protocol P. The properties we show in this section will allow us to reason about properties of $P ;_\rho Q$, especially properties about secret payload messages, by analyzing the protocol Q^α.

The key abstraction mapping α can be different for each specific protocol. Usually it is enough to consider the abstraction of the key patterns that can be synchronized from protocol P to protocol Q in any reachable states, as specified by the mapping ρ. Let $t(x_1, \ldots, x_n)$ denote any such key pattern. We assume that all such patterns are *strongly irreducible*, i.e., that they, and all their instances by irreducible substitutions, are irreducible by equations E_P modulo axioms B_P. Note that in Q the sort QKey is *parametric*: it is instantiated to different data types for keys depending on the choice of P. The central intuition about the abstract protocol Q^α is that the keys that are instances of each key pattern $t(x_1,\ldots,x_n)$ in ρ are abstracted by *constructor terms* using some new constructors Ω, possibly modulo some axioms B_α. Furthermore, key terms that cannot unify with any of the key patterns, are mapped to a special constant in Ω, e.g., c. In this way, Q^α terms of sort QKey become *constructor terms* up to B_α-equivalence, thus greatly simplifying their representation. Different choices of key abstraction are possible (see Example 6.9 below for a concrete illustration). Here is the general method:

Definition 6.4 (Key Abstraction)**.** A *key abstraction* α of protocol Q^ρ by protocol Q^α is defined as follows: (i) new key-building constructors Ω and axioms B_Ω are specified, including a special constant c in Ω, (ii) for each key pattern $t(x_1,\ldots,x_n)$ in ρ, a constructor Ω-term $u_t(x_{i_1},\ldots,x_{i_k})$ with $\{x_{i_1},\ldots,x_{i_k}\} \subseteq \{x_1,\ldots,x_n\}$ is chosen, and (iii) the key abstraction α is defined by equations $\alpha(t(x_1,\ldots,x_n)) = u_t(x_{i_1},\ldots,x_{i_k})$ for each key pattern $t(x_1,\ldots,x_n)$ in ρ, plus an additional "otherwise" default equation "$\alpha(t) = c \ \ [otherwise]$" (that applies to any t that does not $E_P \cup B_P$-unify with any of the key patterns) subject to the requirements: For each key pattern $t(x_1,\ldots,x_n)$ in ρ,

$$t(x_1,\ldots,x_n) =_{E_P \cup B_P} t(x'_1,\ldots,x'_n) \ \Rightarrow\ u_t(x_{i_1},\ldots,x_{i_k}) =_{B_\Omega} u_t(x'_{i_1},\ldots,x'_{i_k})$$

which ensures that α maps $E_P \cup B_P$-equivalence classes of keys in Q^ρ to B_Ω-equivalence classes of keys in Q^α.

The key abstraction should be such that the abstract key implementation map for Q^α obtained by composing ρ with α satisfies the *admissibility requirements* as the original map ρ (see Section 6.3).

Example 6.9. An example key abstraction α for a reader and a passport's keys in a concrete reachable state of the protocol PA^ρ according to the key mapping is:

$$\alpha(f_1(key(reader, r_1), key(passport, r_2))) = k_1(reader, passport, r_1),$$
$$\alpha(f_2(key(reader, r_1), key(passport, r_2))) = k_2(reader, passport, r_2)$$

where $\Omega = \{k_1, k_2, c\}$ and $B_\alpha = \varnothing$. This α satisfies the admissibility requirements, since it ensures that every key is abstracted differently by using the participants of BAC and a fresh variable that is involved in the key.

Note that the protocol Q^α also extends Q with extra Dolev-Yao strands on abstract keys such that if the intruder was able to generate or learn some QKey k from protocol

P, then Q^{α} can generate the abstract version $\alpha(k)$, e.g., a Dolev-Yao strand of the form $:: nil :: [nil \mid +(k_1(i, A, r_?)),\ nil]$ or $[nil \mid +(c),\ nil]$. We refer to this set of extra Dolev-Yao strands as DY_{Ω}.

The key abstraction α is homomorphically extended to terms. We denote by $t\alpha$ the term $t[\alpha(t_1), \ldots, \alpha(t_n)]_{Pos_1, \ldots, Pos_n}$ where $\{t_1, \ldots, t_n\}$ is the set of top QKey subterms, and $Pos_i = \{p_{ij} \in Pos(t) \mid t|_{p_{ij}} =_{E_P \cup B_P} t_i\}$. Given a substitution α, we use $\theta\alpha$ to denote the substitution such that $\{x \mapsto t\alpha \mid x \mapsto t \in \theta\}$. Note that it is possible that a QKey term can contain other QKey terms as subterms, but we do not need to consider those subterms that are not at the top QKey position, since Q is parameterized by sort QKey.

Example 6.10. Consider the key abstraction map α of Example 6.9 and the following strand Str in a state of protocol $BAC;_{\rho} PA$:

$$[-(\mathcal{E}(Read, f_1(key(reader, r_1), key(passport, r_2)),$$
$$f_2(key(reader, r_1), key(passport, r_2)))))]$$

The set of top QKey subterms in this strand is:

$$\{f_1(n(reader, r_1), n(passport, r_2)),\ f_2(n(reader, r_1), n(passport, r_2))\}$$

and the result of applying α to Str is:

$$[-(\mathcal{E}(Read, k_1(reader, passport, r_1), k_2(reader, passport, r_2)))]$$

Note that, for the sake of brevity, the encapsulation operator $\mathcal{E}$ has not been expanded into its definition in this example.

Given a QKey abstraction α of a protocol Q^{ρ}, the protocol Q^{α} with abstract QKeys is thus specified as $Q^{\alpha} = ((\Sigma_Q \cup \Omega, E_{Q^{\alpha}} \cup B_{Q^{\alpha}}), SSpec_{Q^{\alpha}})$, where Σ_Q is the signature of protocol Q and Ω is the set of constructors for abstracted keys as defined above. $E_{Q^{\alpha}} = E_Q$ and $B_{Q^{\alpha}} = B_Q \cup B_{\Omega}$. $SSpec_{Q^{\alpha}}$ is a set of strands specifying both the capabilities of the attacker and the behavior of the honest principals. $SSpec_{Q^{\alpha}} = SSpec_Q|_{HS}\rho\alpha \cup SSpec_Q|_{DY} \cup \Omega_{DY}$, i.e., the strands in the specification of the protocol Q^{ρ} are adapted to use abstracted keys as follows: (i) the input parameters are removed from the strands of honest principals, and the keys are instantiated by the corresponding abstracted keys according to the key implementation map ρ and the key abstraction α, (ii) the intruder's strands are extended with the extra strands denoting intruder's capability on generating abstracted keys.

$$\begin{tikzcd}
t\alpha \arrow[r, dotted, "E_{\mathsf{Shared}},B_{\mathsf{Shared}}\cup B_\alpha"'] & t!_{E_P,B_P}\alpha \arrow[r, dotted, "E_Q,B_{Q^\alpha}"'] & w \arrow[r, equal, "B_{Q^\alpha}"'] & (t\alpha)!_{E_;,B_;} \\
 & t!_{E_P,B_P} \arrow[u, "\alpha"] \arrow[r, "E_Q,B_Q"] & u \arrow[u, "\alpha"] \arrow[dr, equal, "B_;"] & \\
t \arrow[uu, "\alpha"] \arrow[ur, "E_P,B_P"'] \arrow[rrr, "E_;,B_;"'] & & & t!_{E_;,B_;} \arrow[uu, "\alpha"']
\end{tikzcd}$$

Figure 6.1: Proof of Lemma 6.2

We first show in the following lemma that the decomposition and key abstraction map preserves equality of terms of sort or subsort of $\mathcal{E}\mathsf{Msg}$ in the composed protocol.

Lemma 6.2. *Given an admissible composed protocol $P;_\rho Q$, and an admissible* QKey *abstraction α, for any terms $t, t' \in \mathcal{T}_{\Sigma_{P;_\rho Q}}(X)_{\mathcal{E}\mathsf{Msg}}$, if $t =_{E_; \cup B_;} t'$, then $t\alpha =_{E_{Q^\alpha} \cup B_{Q^\alpha}} t'\alpha$.*

Proof. Since $E_{Q^\alpha} = E_Q$, we only need to prove: if $t =_{E_; \cup B_;} t'$, then $t\alpha =_{E_Q \cup B_{Q^\alpha}} t'\alpha$. The proof is based on the following observations:

1. By disjointness of Σ_P and Σ_Q, and the definition of E_Q,
   (a) for any rewrite rule $l \to r$ in E_Q, if there is a substitution θ and position pos s.t. $t!_{E_P,B_P}|_{pos} =_B l\theta$, and $t \to t!_{E_P,B_P}[r\theta]_{pos}$, then pos is not a QKey position and does not have a QKey position as prefix,
   (b) for any rewrite rule $l \to r$ in E_Q, if term v is a subterm of l (resp. r) and $v \in [T_{\mathsf{QKey}}]_{E_P \cup B_P}$, then $v \in Var(l)$ (resp. $v \in Var(r)$).
   (c) for any rewrite rule $l' \to r'$ in E_P, if there is a substitution θ' and position pos' s.t. $t|_{pos'} =_{B_P} l'\theta'$, and $t \to t[r'\theta']_{pos'}$, then $t|_{pos'}$ is of sort QKey or sort Shared or subsort of Shared
2. By the convergence of $\to_{E_;,B_;}$, $\to_{E_P,B_P}$ and $\to_{E_Q,B_{Q^\alpha}}$,
   (a) $t =_{E_; \cup B_;} t'$ iff $t!_{E_;,B_;} =_{B_;} t'!_{E_;,B_;}$,
   (b) $t\alpha =_{E_Q \cup B_{Q^\alpha}} t'\alpha$ iff $(t\alpha)!_{E_Q,B_{Q^\alpha}} =_{B_{Q^\alpha}} (t'\alpha)!_{E_Q,B_{Q^\alpha}}$,

The intuition of the proof is illustrated in Figure 6.1. To show that if $t =_{E_; \cup B_;} t'$, then $t\alpha =_{E_Q \cup B_{Q^\alpha}} t'\alpha$ where $B_{Q^\alpha} = B_Q \cup B_\alpha$, we show that (i.) $t!_{E_;,B_;} =_{B_;} (t!_{E_P,B_P})!_{E_Q,B_Q}$

We prove by contradiction that $(t!_{E_P,B_P})!_{E_Q,B_Q}$ cannot be further rewritten. Suppose there is a rule $l \to r \in E_;$ and substitution θ such that $(t!_{E_P,B_P})!_{E_Q,B_Q}|_{pos} =_{B_;} l\theta$. By the definition of $!_{E_Q,B_Q}$, the rule $l \to r$ cannot be in E_Q. If $l \to r$ is a rule in $E_P - E_Q$, then the redex $(t!_{E_P,B_P})!_{E_Q,B_Q}|_{pos}$ is a term of sort QKey and is introduced by rules in E_Q. By the disjointness of Σ_P and Σ_Q, if new QKeys are introduced to the terms by applying rules in E_Q, then there are variables of sort $QKey$ on the r.h.s of the rule that do no show up in the l.h.s, which is impossible by the assumption on rules in E_Q. By observation 2, we therefore have $t!_{E_;,B_;} =_{B_;} (t!_{E_P,B_P})!_{E_Q,B_Q}$.

(ii.) if $(t!_{E_P,B_P})!_{E_Q,B_Q} =_{B_;} (t'!_{E_P,B_P})!_{E_Q,B_Q}$,

then $(t!_{E_P,B_P}\alpha)!_{E_Q,B_{Q^\alpha}} =_{B_{Q^\alpha}} (t'!_{E_P,B_P}\alpha)!_{E_Q,B_{Q^\alpha}}$

For any rewrite rule $l \to r$ in E_Q, if there is a substitution θ s.t. $t!_{E_P,B_P}|_{pos} =_{B_;} l\theta$, and $t!_{E_P,B_P} \to_{E_Q,B_Q} t!_{E_P,B_P}[r\theta]_{pos}$, then by the definition of the key abstraction α, $(l\theta)\alpha =_{B_{Q^\alpha}} (t!_{E_P,B_P}|_{pos})\alpha$. By observation 1, since *pos* is not a QKey position and does not have a QKey position as prefix, $t!_{E_P,B_P}|_{pos}\alpha = t!_{E_P,B_P}\alpha|_{pos}$, therefore $(l\theta)\alpha =_{B_{Q^\alpha}} t!_{E_P,B_P}\alpha|_{pos}$. Also by observation 1, since all the terms of sort QKey in l are variables, $(l\theta)\alpha = l(\theta\alpha)$. Therefore $t!_{E_P,B_P}\alpha|_{pos} =_{B_{Q^\alpha}} l(\theta\alpha)$. Therefore $t!_{E_P,B_P}\alpha \to_{E_Q,B_{Q^\alpha}} (t!_{E_P,B_P}\alpha)[r\theta\alpha]_{pos} = t!_{E_P,B_P}[r\theta]_{pos}\alpha$. By successively applying the above argument, we have that if $(t!_{E_P,B_P})!_{E_Q,B_Q} =_{B_;} (t'!_{E_P,B_P})!_{E_Q,B_Q}$, then $(t!_{E_P,B_P}\alpha)!_{E_Q,B_{Q^\alpha}} =_{B_{Q^\alpha}} (t'!_{E_P,B_P}\alpha)!_{E_Q,B_{Q^\alpha}}$.

(iii.) $(t\alpha)!_{E_Q,B_{Q^\alpha}} =_{B_{Q^\alpha}} (t!_{E_P,B_P}\alpha)!_{E_Q,B_{Q^\alpha}}$

By observation 1c and since a term of sort QKey can never be a subterm of a term of sort Shared, any rewrites on terms of sort Shared are either under a QKey position or in parallel. And by the definition of the key abstraction α, $t\alpha =_{E_{\mathsf{Shared}} \cup B_{\mathsf{Shared}} \cup B_\alpha} t!_{E_P,B_P}\alpha$. Since $E_{\mathsf{Shared}} \cup B_{\mathsf{Shared}} \cup B_\alpha \subseteq E_Q \cup B_{Q^\alpha}$, $(t\alpha)!_{E_Q,B_{Q^\alpha}} =_{B_{Q^\alpha}} (t!_{E_P,B_P}\alpha)!_{E_Q,B_{Q^\alpha}}$.

which then implies that if $t!_{E_;,B_;} =_{B_;} t'!_{E_;,B_;}$, then $(t\alpha)!_{E_Q,B_{Q^\alpha}} =_{B_{Q^\alpha}} (t'\alpha)!_{E_Q,B_{Q^\alpha}}$. Together with the convergence, we have that $t =_{E_; \cup B_;} t' \Leftrightarrow t!_{E_;,B_;} =_{B_;} t'!_{E_;,B_;} \Rightarrow (t\alpha)!_{E_Q,B_{Q^\alpha}} =_{B_{Q^\alpha}} (t'\alpha)!_{E_Q,B_{Q^\alpha}} \Leftrightarrow t\alpha =_{E_Q \cup B_{Q^\alpha}} t'\alpha$.

We now define the simulation relation $\mathcal{H}_Q$ between $P;_\rho Q$ and Q^α. The relation $\mathcal{H}_Q$ essentially projects out from a state in the protocol $P;_\rho Q$ the strands that are instances of strands in protocol Q together with the set of intruder knowledge that is in the signature of protocol Q, and then applies key abstraction on them. Again we only consider states that are reachable from initial states.

Definition 6.5 (Relation $\mathcal{H}_Q$)**.** Given an admissible composed protocol $P;_\rho Q$ and the abstracted protocol Q^α with α being an admissible QKey abstraction, let $S_; = SS_; \ \& \ IK_;$ denote a state that is reachable from the initial state in protocol $P;_\rho Q$, $S_{Q^\alpha} = SS_{Q^\alpha} \ \& \ IK_{Q^\alpha}$ denote a state that is reachable from the initial state in protocol Q^α, $S_;|_Q$ denote $SS_;|_Q \& IK_;|_{\Sigma_{Q^\rho}}$, and $SS|_Q \& IK|_Q = (S_;|_Q)\alpha$. The relation $\mathcal{H}_Q$ between reachable states in $P;_\rho Q$ and in Q^α is then defined by the following definitional equivalence: $(S_;, S_{Q^\alpha}) \in \mathcal{H}_Q$ iff:

(i) $SS|_Q \subseteq_{E_Q \cup B_{Q^\alpha}} SS_{Q^\alpha}$, $SS|_Q|_{HS} =_{E_Q \cup B_{Q^\alpha}} SS_{Q^\alpha}|_{HS}$,

(ii) $IK|_Q =_{E_Q \cup B_{Q^\alpha}} IK_{Q^\alpha}$

where $B_{Q^\alpha} = B_Q \cup B_\alpha$.

We then show that $\mathcal{H}_Q$ defines a simulation relation. The main reason for $\mathcal{H}_Q$ being a simulation is that, by the requirements on the key abstraction α, the abstraction preserves equality and deducibility of keys in protocol Q^ρ. In the following we first show in Lemma 6.3 that a transition from a state in the composed protocol $P;_\rho Q$ induces a transition from a corresponding state (i.e., related by $\mathcal{H}_Q$) in protocol Q^α while preserving the relation $\mathcal{H}_Q$. This will be used to prove Theorem 6.3, which allows us to simulate protocol $P;_\rho Q$ by protocol Q^α.

Lemma 6.3. *Given an admissible composed protocol $P;_\rho Q$, the secrecy enforcing key establishment protocol P, and the abstracted protocol Q^α with α being an admissible* QKey *abstraction, let $S_;$ be a state that is reachable in protocol $P;_\rho Q$ and S_{Q^α} a state that is reachable in protocol Q^α such that $(S_;, S_{Q^\alpha}) \in \mathcal{H}_Q$, if there exists a state $S'_;$ such that $S_; \rightarrow_{rl} S'_;$ by applying the transition rule rl, then there exists a state S'_{Q^α} of protocol Q^α such that $S_{Q^\alpha} \rightarrow^* S'_{Q^\alpha}$ and $(S'_;, S'_{Q^\alpha}) \in \mathcal{H}_Q$.*

Proof. The proof is by case analysis on the transition rule rl. Since the proof of this Lemma requires a somewhat lengthy case analysis, the details of the proof have been moved to Appendix C.

Theorem 6.3. *Given an admissible composed protocol $P;_\rho Q$, the secrecy enforcing key establishment protocol P, and the abstracted protocol Q^α with α being an admissible* QKey *abstraction, if a state $S_;$ is reachable from the initial state of protocol $P;_\rho Q$, then there exists a state S_{Q^α} such that S_{Q^α} is reachable from the initial state in protocol Q^α, and $(S_;, S_{Q^\alpha}) \in \mathcal{H}_Q$.*

Proof. Since the initial states of both systems consist of the empty set of strands and empty set of intruder knowledge, therefore $(S_{;init}, S_{Q^{\alpha}{}_{init}}) \in \mathcal{H}_Q$ is straightforward. The proof of this theorem is then a straightforward successive application Lemma 6.3.

From Theorem 6.3 we can conclude that if the intruder cannot learn the secret payload message between two honest principals in protocol Q^{α}, then the composed protocol $P;_{\rho} Q$ also satisfies this property. These results, as well as those in Section 6.4.1, form the basis for the modular verification of properties of the protocol $P;_{\rho} Q$ in Maude-NPA presented in Section 6.5.

6.5 MODULAR VERIFICATION OF $P;_{\rho} Q$ IN MAUDE-NPA

In this section we show how the simulation results in Section 6.4 can be used in Maude-NPA's protocol analysis. We first present a methodology which, given an attack pattern in protocol $P;_{\rho} Q$, generates corresponding attack patterns in protocols P and Q^{α} whose reachability can then be analyzed by running Maude-NPA on protocol P and protocol Q^{α} separately. We then show the correctness of this methodology in Theorem 6.4, which relies on lifting the simulation results in Section 6.4 to Maude-NPA's backwards semantics.

Given an attack state pattern $Att_{;}$ of protocol $P;_{\rho} Q$, we wish to obtain an attack state pattern Att_P for protocol P (according to Theorem 6.2) and an attack state pattern Att_Q for protocol Q^{α} (according to Theorem 6.3). However, an attack state pattern $Att_{;}$ may not include any strands of P and we need a completion mechanism which expands a given attack state pattern so that each child strand in the attack state pattern has a parent strand that is explicitly listed in the state. Following the semantics of sequential protocol composition in Maude-NPA [48], the composition completion of an attack pattern does not change the reachability of the attack pattern.

Definition 6.6 (Composition Completion)**.** Given an attack state pattern $Att_{;}$ of protocol $P;_{\rho} Q$, w.l.o.g assume that the strands which are instances of strands in protocol Q in $Att_{;}$ have the form:

$$\begin{aligned} Att_{;}|_Q = & (b_1)\ [\{a_1\ \to b_1\ ;;\ Mode_1\ ;; M_{b_1}\}, L_{b_1} \mid nil]\ \ \& \ldots \& \\ & (b_n)\ [\{a_n\ \to b_n\ ;;\ Mode_n\ ;; M_{b_n}\}, L_{b_n} \mid nil] \end{aligned}$$

where $\forall i \in [1, n]$, $(b_i)\ [\{a_i\ \to b_i\ ;;\ Mode_i\ ;; M_{b_i}\}, L_{b_i}]$ is an instance of a (possibly partial) honest protocol strand of protocol Q, and all the QKeys in L_{b_i} show up in M_{b_i}. The *composition completion* of $Att_{;}$ is $\widehat{Att_{;}} = \{\widehat{Att_{;1}}, \ldots \widehat{Att_{;m}}\}$ with $\widehat{Att}_{;j \in [1,m]} = (Att_{;} \cup Att_{Pj})\theta_j$, where Att_P is defined as follows:

$$\begin{aligned} Att_P =& (a_1)\ [L_{a_1}, \{a_1 \to b_1 \ ;;\ Mode_1\ ;; M_{a_1}\} \mid nil]\ \ \& \ldots\ \ \& \\ & (a_n)\ [L_{a_n}, \{a_n \to b_n \ ;;\ Mode_n\ ;; M_{a_n}\} \mid nil] \end{aligned}$$

where $\forall i \in [1, n]$, $(a_i)\ [nil|L_{a_i}, \{a_i \to b_i\ ;;\ Mode_i\ ;; M_{a_i}\},\ nil] \in \mathcal{P}_{P;_\rho Q} \cup Att_;|_P$, $\theta_j \in Unif_{E_P \cup B_P}(M_{a_1} = M_{b_1} \wedge \ldots \wedge M_{a_n} = M_{b_n})$. We require that all the QKeys in $\widehat{Att_;}$ are instances of the key patterns.

Example 6.11. Consider the following attack pattern in the protocol $BAC;_\rho PA$ querying whether the intruder can learn $data(passport, r')$ generated by the passport in the PA protocol:

$$\begin{aligned} Att_; =& [nil, \{BAC.B \to PA.B\ ;;\ 1-1\ ;;\ passport; reader; K_1; K_2\}, \\ & -(\mathcal{E}(read(reader), K_1, K_2)), \\ & +(\mathcal{E}(data(passport, r'); sign(h(data(passport, r')), sk(passport)), K_1, K_2)) \\ & \mid nil]\ \&\ data(passport, r') \in \mathcal{I} \end{aligned}$$

where K_1 and K_2 denote the key terms $f_1(key(reader, r_1), key(passport, r_2))$ and $f_2(key(reader, r_1), key(passport, r_2))$ respectively.
The composition completion $\widehat{Att_;}$ of $Att_;$ is:

$$Str_P \& Att_;[B \mapsto passport, A \mapsto reader, K_A \mapsto key(reader, r_1)]$$

where Str_P denotes the passport strand in Example 6.4. Note that the unification call between the input and output synchronization messages gave also some instantiation for P and, thus, we use L_P to denote $Str_P[B \mapsto passport, A \mapsto reader, K_A \mapsto key(reader, r_1)]$. Note that in the case where two participants of a protocol P can swap roles in protocol Q, the unification of the input and output synchronization messages will produce all different combinations.

Given an attack state pattern $Att_;$ in the protocol $P;_\rho Q$, we follow the following steps to check whether the attack state pattern $Att_;$ can backwards reach an initial state in protocol $P;_\rho Q$:

1. *Composition Completion*: we first generate the composition completion $\widehat{Att_;} = \{\widehat{Att_;}_1, \ldots \widehat{Att_;}_m\}$ of $Att_;$.

2. For each composition completed state $\widehat{Att}_{;j} = SS_{;} \mathbin{\&} IK_{;}$ in $\widehat{Att}_{;}$, we analyze the reachability of $\widehat{Att}_{;j}$ as follows:

   (a) *Reachability in P*: check the attack pattern $S_P = SS_{;}|_P \& IK_{;}|_{\Sigma_P(X)}$ in protocol P. If S_P cannot backwards reach an initial state, skip the step (b), and mark the reachability of the state $\widehat{Att}_{;j}$ as false.

   (b) *Reachability in Q^α*: check the attack pattern $S_Q = (SS_{;}|_Q \& IK_{;}|_{\Sigma_{Q^\rho}(X)})\alpha$ in protocol Q^α. If S_Q cannot backwards reach an initial state, mark the reachability of the state $\widehat{Att}_{;j}$ as false.

If the reachability of all the states in $\widehat{Att}_{;}$ are marked as false, then the attack state $Att_{;}$ cannot reach an initial state, i.e., the protocol $P;_\rho Q$ is secure with respect to the attack state $Att_{;}$. Note that extra analysis may be executed in P in order to generate some specific key abstraction mappings.

The correctness of this approach is proved in the following theorem. The proof is based on lifting Theorems 6.1 and 6.3 to symbolic states according to the soundness and completeness of Maude-NPA's backwards operational semantics w.r.t. the forwards semantics [60].

Theorem 6.4 (Symbolic Modular Verification Theorem)**.** *Given an admissible composed protocol $P;_\rho Q$, the secrecy enforcing key establishment protocol P, the abstracted protocol Q^α with α being an admissible* QKey *abstraction, a symbolic attack state $Att_{;}$ of protocol $P;_\rho Q$, if $Att_{;}$ can backwards reach an initial state in protocol $P;_\rho Q$, then there exists a state $\widehat{Att}_{;j} = SS_{;} \& IK_{;}$ in the set of composition completion of $Att_{;}$, such that $S_P = SS_{;}|_P \& IK_{;}|_{\Sigma_P(X)}$ can backwards reach an initial state in protocol P, and $S_{Q^\alpha} = (SS_{;}|_Q \& IK_{;}|_{\Sigma_{Q^\rho}(X)})\alpha$ can backwards reach an initial state in protocol Q^α.*

To prove Theorem 6.4, in the following we prove auxiliary Lemmas 6.4, 6.5 and 6.6. We first show in Lemma 6.4 that if the attack pattern $Att_{;}$ can backwards reach an initial state in protocol $P;_\rho Q$, then there exists $\widehat{Att}_{;j}$ in the set of composition completion of $Att_{;}$ that can backwards reach an initial state. By lifting the results that we have proved in Section 6.4, we then show in Lemmas 6.5 and 6.6, respectively, that if $\widehat{Att}_{;j}$ can backwards reach an initial state, then the state S_P can backwards reach an initial state in protocol P and the state S_{Q^α} can backwards reach an initial state in protocol Q^α. The proof of Theorem 6.4 then follows straightforwardly.

We show in the following Lemma that the composition completion of an attack pattern does not change the reachability of the attack pattern.

Lemma 6.4. *Given an admissible composed protocol $P;_\rho Q$ and an attack pattern $Att_;$ of protocol $P;_\rho Q$, if $Att_;$ can backwards reach an initial state in protocol $P;_\rho Q$, then there exists a state $\widehat{Att}_{;j}$ in the composition completion $\widehat{Att}_;$ of $Att_;$ which can backwards reach an initial state in $P;_\rho Q$.*

Proof. According to Maude-NPA's semantics of protocol composition, if the attack pattern $Att_;$ can backwards reach an initial state, then each child strand in $Att_;$ can backwards synchronize with a parent strand such that the composed strand can reach an initial state. The proof of this Lemma is then straightforward, since the composition completion explores all possible parent strand that can synchronize with the child strands in $Att_;$.

In the following, we lift the protocol composition modularity results that we have proved in Section 6.4 to symbolic backwards analysis. We first recall the lifting relation that is defined in [60].

Definition 6.7 (Lifting Relation [60])**.** Given a protocol $\mathcal{P}$, a symbolic state S and a ground state s, we say that s *lifts* to S, or that S *instantiates* to s with a *ground* substitution $\theta : (Var(S) - \{SS, IK\}) \to \mathcal{T}_\Sigma$ written $S >^\theta s$ iff

- for each strand $:: r_1, \ldots, r_m :: [L]$ in S, there exists a strand $[l]$ in s such that $l =_{E_\mathcal{P}} L\theta$.
- for each positive intruder fact $w{\in}\mathcal{I}$ in S, there exists a positive intruder fact w' in s such that $w' =_{E_\mathcal{P}} w\theta$, and
- for each negative intruder fact $w{\notin}\mathcal{I}$ in S, there is no positive intruder fact w' in s such that $w' =_{E_\mathcal{P}} w\theta$.

where SS and IK are variables denoting unspecified set of strands and knowledge.

We then lift Theorem 6.1 in Section 6.4 to symbolic backwards reachability semantics in the following Lemma 6.5.

Lemma 6.5. *Given an admissible composed protocol $P;_\rho Q$, the secrecy enforcing key establishment protocol P, and an attack pattern $Att_;$ of protocol $P;_\rho Q$, let $\widehat{Att}_{;j} = SS_;\&IK_;$ be a state in the composition completion of $Att_;$, where $SS_;$ and $IK_;$ denote the set of strands and intruder knowledge of $\widehat{Att}_{;j}$ respectively. If $\widehat{Att}_{;j}$ can backwards reach an initial state in protocol $P;_\rho Q$, then the state $S_P = SS_;|_P\&IK_;|_{\Sigma_P(X)}$ can backwards reach an initial state in protocol P.*

Proof. Since the state $\widehat{Att_{;}}_{j}$ can backwards reach an initial state S_0 in protocol $P;_{\rho} Q$, by soundness of Maude-NPA's backwards operational semantics w.r.t. the forwards semantics, there is a grounding substitution θ and a ground state $s_{;} = ss_{;}\&ik_{;}$ where $ss_{;}$ and $ik_{;}$ denote the set of ground strands and intruder knowledge of $s_{;}$ respectively, such that $s_{;}$ is reachable from the initial state s_0 in the forward semantics in $P;_{\rho} Q$ and $\widehat{Att_{;}}_{j} >^{\theta} s_{;}$. By Theorem 6.1, there exists a state s_P that is reachable from the initial state in protocol P such that $(s_{;}, s_P) \in \mathcal{H}_P$. By the definition of $>^{\theta}$, for any strand L in $SS_{;}$, there is a strand $l =_{E_{;} \cup B_{;}} L\theta$ in $ss_{;}$. By the disjointness of the signatures of protocol P and protocol Q, if $L \in SS_{;}|_P$, then $l \in ss_{;}|_P$, and $l =_{E_P \cup B_P} L\theta$. Similarly, if a message M is in $IK_{;}|_{\Sigma_P(X)}$, there is a message m in the intruder knowledge of s_P such that $m =_{E_P \cup B_P} M\theta$. Since $Att_{;}$ is an attack pattern, there is no negative intruder fact $w \notin \mathcal{I}$ in $Att_{;}$. Therefore $S_P >^{\theta} s_P$. Therefore, by completeness of Maude-NPA's backwards operational semantics w.r.t. the forwards semantics, S_P can reach an initial state in protocol P.

We then lift Theorem 6.3 in Section 6.4 to symbolic backwards reachability semantics in Lemma 6.6.

Lemma 6.6. *Given an admissible composed protocol $P;_{\rho} Q$, the secrecy enforcing key establishment protocol P, the abstracted protocol Q^{α} with α being an admissible* QKey *abstraction, and an attack pattern $Att_{;}$ of protocol $P;_{\rho} Q$, let $\widehat{Att_{;}}_{j} = SS_{;}\&IK_{;}$ be a state in the composition completion of $Att_{;}$, where $SS_{;}$ and $IK_{;}$ denote the set of strands and intruder knowledge of $\widehat{Att_{;}}_{j}$ respectively. If $\widehat{Att_{;}}_{j}$ can backwards reach an initial state in protocol $P;_{\rho} Q$, then the state $S_{Q^{\alpha}} = (SS_{;}|_Q \& IK_{;}|_{\Sigma_{Q^{\rho}}(X)})\alpha$ can backwards reach an initial state in protocol Q^{α}.*

Proof. Since the state $\widehat{Att_{;}}_{j}$ can reach an initial state in protocol $P;_{\rho} Q$, by soundness of Maude-NPA's backwards operational semantics w.r.t. the forwards semantics, there is a ground state $s_{;} = ss_{;}\&ik_{;}$ and a substitution θ such that $s_{;}$ is reachable from the initial state in the forward semantics in $P;_{\rho} Q$ and $\widehat{Att_{;}}_{j} >^{\theta} s_{;}$. By Theorem 6.3, there exists a ground state $s_{Q^{\alpha}}$ that is reachable from the initial state in the forward semantics in protocol Q^{α}, and $(s_{;}, s_{Q^{\alpha}}) \in \mathcal{H}_Q$. By the definition of $>^{\theta}$, for any strand L in $SS_{;}$, there is a strand $l =_{E_{;} \cup B_{;}} L\theta$ in $ss_{;}$. By the disjointness of the signatures of protocol P and protocol Q, if $L \in SS_{;}|_Q$, then $l \in ss_{;}|_Q$. By the definition of $\mathcal{H}_Q$, since $(s_{;}, s_{Q^{\alpha}}) \in \mathcal{H}_Q$ and $l \in ss_{;}|_Q$, $l\alpha$ is a strand in $s_{Q^{\alpha}}$. By the requirements on the attack patterns and the key abstraction α, $L\theta\alpha = (L\alpha)\theta|_{Var(L\alpha)}$. Therefore $l\alpha =_{E_Q \cup B_{Q^{\alpha}}} (L\alpha)\theta|_{Var(L\alpha)}$. Similarly, if a message $M\alpha$ is in the intruder knowledge of $S_{Q^{\alpha}}$, there exists a message m in the intruder knowledge of $s_{Q^{\alpha}}$ such that $m\alpha =_{E_Q \cup B_{Q^{\alpha}}} (M\alpha)\theta|_{Var(M\alpha)}$. Since $Att_{;}$ is an attack pattern, there is no negative intruder fact $w \notin \mathcal{I}$ in $Att_{;}$. Therefore $S_{Q^{\alpha}} >^{\theta'} s_{Q^{\alpha}}$, where $\theta' \subseteq \theta$. Then by completeness of

Maude-NPA's backwards operational semantics w.r.t.the forwards semantics, S_{Q^α} can reach an initial state in protocol Q^α.

Theorem 6.4 is now a straightforward corollary. It shows that the symbolic reachability analysis of the composed protocol $P;_\rho Q$ can be reduced to the symbolic reachability analysis of the parent protocol and the abstracted child protocol separately. The proof of Theorem 6.4 is a straightforward application of Lemma 6.4, Lemma 6.5 and Lemma 6.6.

Example 6.12. As an example of modular protocol analysis, we continue with the protocol $BAC;_\rho PA$ and Example 6.11. We first check that the protocol BAC is secrecy-enforcing; therefore the intruder cannot learn K_1 or K_2. We then check in protocol BAC the attack state pattern in which the strand L_P finished execution. Maude-NPA found an initial state. To generate the key abstraction α, we also check in protocol BAC the attack pattern in which the passport's (resp. reader's) strand finished execution without the corresponding reader's (resp. passport's) strand (an authentication attack). Maude-NPA terminated without reaching any initial states. We therefore obtain protocol PA^α from PA by removing the input synchronization message and replacing the two keys by $k_1(A, B, R_A)$ and $k_2(A, B, R_B)$ respectively with R_A, R_B of sort $\mathsf{Fresh}?$, a super sort of Fresh. The attack pattern of Example 6.11 is now written according to the key abstraction α and the specification of protocol PA^α as:

$$\begin{aligned}
&[nil, -(\mathcal{E}(read(reader), k_1(reader, passport, r_1), k_2(reader, passport, r_2)),\\
&+ (\mathcal{E}(data(passport, r'); sign(h(data(passport, r')), sk(passport)),\\
&\quad k_1(reader, passport, r_1), k_2(reader, passport, r_2))) \mid nil]\\
&\&\ data(passport, r') \in \mathcal{I}
\end{aligned}$$

Maude-NPA terminated without finding any initial states, i.e., this attack cannot happen. We therefore conclude that the intruder cannot learn the secret payload.

6.6 EXPERIMENTS

In this section we describe some experiments that we have performed. using the Maude-NPA cryptographic protocol analysis tool.

6.6.1 BAC-PA.

The basic access control protocol (BAC) and passive authentication protocl (PA) [102] are used in E-passport to protect the content of the RFID chip. The BAC protocol is used for preventing skimming, the session keys established by this protocol are used to secure further communication. The BAC protocol starts with a challenge, which is then followed by a mutually authenticated communication between the passport and the reader. One of the protocols that can use the secure communication established by the BAC protocol is the PA protocol. The PA protocol protects the integrity of the content stored in the passport by first hashing the data and then having it signed by the Document Signers. Protocols BAC and PA proceed as explained in Examples 6.1 and 6.2. Our model is similar to [101].

We used Maude-NPA to search for the attack state in which the intruder can learn the stored data from the communication of an honest passport and an honest reader. We first analyze in the protocol BAC alone whether the intruder can learn the generated session keys from an honest passport and reader. Maude-NPA terminated without any attack being found, therefore the protocol BAC is secrecy-enforcing.

```
eq ATTACK-STATE(0)  =
    :: r, r2 :: [ nil, -(challenge), +(n(passport, r)),
    -(senc(ke(passport, reader), Nr; n(passport, r); Kr);
      mac(km(passport, reader), senc(ke(passport, reader),
              Nr; n(passport, r); Kr))),
    +(senc(ke(passport, reader), n(passport, r); Nr;
         key(passport, r2)); mac(km(passport, reader),
         senc(ke(passport, reader), n(passport, r);
          Nr; key(passport, r2))))
     | nil ]
    || f1(Kr, key(passport, r2)) inI

eq ATTACK-STATE(2)  =
    :: r, r2 :: [ nil, -(challenge), +(n(passport, r)),
    -(senc(ke(passport, reader), Nr; n(passport, r); Kr);
      mac(km(passport, reader), senc(ke(passport, reader),
              Nr; n(passport, r); Kr))),
    +(senc(ke(passport, reader), n(passport, r); Nr;
         key(passport, r2)); mac(km(passport, reader),
         senc(ke(passport, reader), n(passport, r);
          Nr; key(passport, r2))))
```

```
   | nil ]
  || f2(Kr, key(passport, r2)) inI
```

There are two other attack patterns that use reader's strand and are similar to ATTACK-STATE(0) and ATTACK-STATE(2). We omit those two attack patterns for the sake of brevity.

Note that the search space can be reduced by searching whether the intruder can learn $key(passport, r_2)$ and Kr, respectively, instead of $f_1(Kr, key(passport, r_2))$ and $f_2(Kr, key(passport, r_2))$, since the QKeys constructed using f_1 and f_2 only show up in the output parameters.

We then analyze in protocol BAC the attack patterns denoting regular executions with the reader, passport and their keys being honest. Maude-NPA terminated with initial states being found. To avoid repetition, here we only list the passport's strand.

```
eq ATTACK-STATE(5) =
 :: r, r2 :: [ nil,
      -(challenge),
      +(n(passport, r)),
      -(senc(ke(passport, reader), Nr; n(passport, r); key(reader, r1));
           mac(km(passport, reader), senc(ke(passport, reader),
           Nr; n(passport, r); key(reader, r1)))),
     +(senc(ke(passport, reader), n(passport, r); Nr; key(passport, r2));
      mac(km(passport, reader), senc(ke(passport, reader),
            n(passport, r); Nr; key(passport, r2)))
     | nil ]
    || empty
```

We therefore analyze in the protocol PA with an admissible key abstraction whether the intruder can learn the honest passport's stored data from the communication of a reader and passport assuming the honest session keys being secure. The attack pattern is as follows:

```
eq ATTACK-STATE(1) =
   :: r1, r2 :: [ nil ,
    +(E(read(reader), k1(reader, passport, r1), k2(reader, passport, r2))),
    -(E(Dt; sign(sk(passport), h(Dt)),
          k1(reader, passport, r1), k2(reader, passport, r2)))
     | nil]
    || Dt inI
```

For this property, Maude-NPA terminated without any attack being found. We then can conclude that the intruder cannot learn an honest passport's data from its communications with a reader using the composition of the protocols BAC and PA.

We also checked whether the intruder can trick the reader into accepting a passport's data whose certificate is forged by the intruder. That is, the reader and a passport agreed on a session key in the BAC protocol, but then in protocol PA, the passport data that is sent to the reader is certificated by the intruder. The attack pattern is as follows:

```
eq ATTACK-STATE(3) =
:: r1, r2 :: [ nil ,
+(E(read(reader), k1(reader, passport, r1), k2(reader, passport, r2))),
-(E(data(passport, r'); sign(sk(i), h(data(passport, r'))),
     k1(reader, passport,  r1), k2(reader, passport, r2))) | nil]
  || empty
```

For this property, Maude-NPA terminated without any attack being found. We then can also conclude that the intruder cannot forge the certificate.

Note that although the BAC protocol can be cracked by brute-force key search, that is an attack that is out of the scope of this analysis tool.

6.6.2 BAC-AA.

The session keys established in the BAC protocol can also be used to secure communications in the Active Authentication (AA) protocol [102], which prevents passport cloning. The protocol AA starts when the reader sends a challenge to the passport. After receiving the challenge, the passport generates its own random string, concatenates it with the received challenge, signs the concatenated message with its signing key and sends it back to the reader. More specifically, the protocol AA proceeds as follows:

1. $A \rightarrow B : senc(init; Nr, K1); mac(senc(init; Nr, K1), K2)$
2. $B \rightarrow A : senc(sign(Np; Nr, sk), K1); mac(senc(sign(Np; Nr, sk), K1), K2)$

where A and B denote the reader and the passport respectively. K_1 and K_2 denote the session keys generated in protocol BAC. Nr and Np denote random nonces generated by the reader and passport respectively. $senc$ denotes shared key encryption, mac denotes a MAC, $sign(M, K)$ denotes message M signed with key K, and sk denotes the signing key.

We used Maude-NPA to search for the attack state in which the intruder can learn the stored data or the signing key of a passport. Since we already verified that the BAC protocol is secrecy enforcing, we therefore analyze in the protocol AA with an admissible key abstraction that the intruder cannot learn the nonce that is generated by the passport or the signing key of the passport assuming the honest session keys being secure. For this property, Maude-NPA terminated without any attack being found. The attack patterns are as below. Here we only list the attack patterns for the reader's strand. There are two other similar attack patterns for the passport's strand. We omitted those here.

```
eq ATTACK-STATE(1) =
:: r, r1, r2 :: [ nil ,
 +(E(init; q(reader, r),
      k1(reader, passport, r1), k2(reader, passport, r2))),
 - E(sign(sk(passport), q(passport, r'); q(reader, r)),
      k1(reader, passport , r1), k2(reader, passport , r2)))
  | nil]
 || q(passport, r') inI


eq ATTACK-STATE(3) =
:: r, r1, r2 :: [nil ,
  +(E(init ; q(reader, r),
       k1(reader, passport, r1), k2(reader, passport, r2))),
  -(E(sign(sk(passport), q(passport, r'); q(reader, r)),
       k1(reader, passport, r1), k2(reader, passport, r2)))
   | nil]
  || sk(passport) inI
```

We also verified that the reader cannot finish the protocol with a passport whose signature cannot be verified. The attack pattern is listed below. Maude-NPA terminated without any attacks being found for this attack pattern.

```
 eq ATTACK-STATE(4) =
 :: r, r1, r2 ::
 [nil, +(E(init; q(reader, r),
              k1(reader, passport, r1), k2(reader, passport, r2))),
       -(E(sign(sk(i), q(passport, r'); q(reader, r)),
             k1(reader, passport, r1), k2(reader, passport, r2)))
   | nil]
  || empty
```

6.6.3 IKEv1.

The Internet Key Exchange (IKE) [103], is a protocol suite in IPSec that establishes session keys for protecting the remainder of the sessions. There are two phases in IKE, where Phase 1 generates a key *SKEYID* that is used to generate three keys $SKEYID_a$, $SKEYID_e$ and $SKEYID_d$ that are passed on to the Phase 2 protocol. Phase 1 is in turn divided into two modes, Main Mode, in which identities are always encrypted, and Aggressive Mode, in which they are not. The Phase 2 mode, called Quick Mode, is used to generate session keys. There are many ways to mix and match the protocols, which greatly increases the complexity of formal analysis. So the ability to analyze the Phase 1 and Phase 2 protocols separately would be preferable.

As an example, we experimented with a slightly simplified version of the Aggressive Mode with digital signatures (AM_{id}) and the Quick Mode without perfect forward secrecy (QM) in IKE version 1. The Aggressive Mode proceeds as follows:

1. $A \rightarrow B : SA; g^{x_A}; N_A; ID(A)$
2. $B \rightarrow A : SA; g^{x_B}; N_B; ID(B); sig(SA, sk(B),$
   $prf(SA, prf(SA, g^{x_A * x_B}, N_A; N_B), g^{x_B}; g^{x_A}; SA; ID(B)))$
3. $A \rightarrow B : sig(SA, sk(A), prf(SA, prf(SA, g^{x_A * x_B}, N_A; N_B), g^{x_A}; g^{x_B}; SA; ID(A)))$

This protocol has a Diffie-Hellman exponentiation as a core part. The Quick Mode without perfect forward secrecy proceeds as follows:

1. $A \rightarrow B : MID; e(SA, SKEYID_e, prf(SA, SKEYID_a, MID; SA'; M_A); SA'; M_A)$
2. $B \rightarrow A : MID; e(SA, SKEYID_e, prf(SA, SKEYID_a, MID; M_A; SA'; M_B); SA'; M_B)$
3. $A \rightarrow B : MID; e(SA, SKEYID_e, prf(SA, SKEYID_a, MID; M_A; M_B))$

where $SKEYID_a$ and $SKEYID_e$ denote the keys generated in protocol AM_{id} and passed to QM, and N_A, N_B, M_A and M_B denote nonces. The fact that security associations may specify algorithms is modeled by making the security association SA one of the arguments in the cryptographic functions prf, e, and sig.

To check that the intruder cannot learn the generated session key, we first check in the AM$_{id}$ protocol that the key *SKEYID* exchanged between honest participants cannot be learned by the intruder. Note that the keys $SKEYID_a$, $SKEYID_e$ and $SKEYID_d$ are generated based on the key *SKEYID*. Moreover, the keys $SKEYID_a$, $SKEYID_e$ and $SKEYID_d$ never show up in the protocol strands of the Aggressive Mode. Therefore, it is enough to just check the secrecy of the key *SKEYID*. For this property, Maude-NPA terminated without any attack being found. Therefore we can conclude that AM$_{id}$ is secrecy-enforcing.

```
eq ATTACK-STATE(4) =
  :: r3, r4 :: [ nil ,
-(sa1; ExpA; NA; id(a)),
+(sa1; exp(g, c(b, r3)); n(b, r4); id(b); sig(sa1, sk(b),
      prf(sa1, prf(sa1, exp(ExpA, c(b, r3)), NA; n(b, r4)),
          exp(g, c(b, r3)); ExpA; sa1; id(b)))),
-(sig(sa1, sk(a), prf(sa1,  prf( sa1,
          exp(ExpA, c(b, r3)), NA; n(b, r4)),
          ExpA; exp(g, c(b, r3)); sa1; id(a))))
   | nil]
  || exp(ExpA, c(b, r3)) inI

eq ATTACK-STATE(6) =
:: r1, r2 :: [ nil,
  +(sa1; exp(g, c(a, r1)); n(a, r2); id(a)),
  -(sa1; ExpB; NB; id(b); sig(sa1, sk(b), prf(sa1,
             prf(sa1, exp(ExpB, c(a, r1)), n(a, r2); NB),
             ExpB; exp(g, c(a, r1)); sa1; id(b)))),
  +(sig(sa1, sk(a), prf (sa1, prf(sa1,
             exp(ExpB, c(a, r1)), n(a, r2); NB),
             exp(g, c(a, r1)); ExpB; sa1; id(a))))
  | nil]
 || exp(ExpB, c(a, r1)) inI
```

We were able to find the authentication attack that is mentioned by Cremers in [6], in which Alice may think that she shared a key with Bob, but Bob actually did not accept that key. The attack pattern is as follows:

```
  eq ATTACK-STATE(2) =
  :: r1, r2 :: [ nil,
     +( sa1; exp(g, c(a, r1)); n(a, r2); id(a)),
     -(sa1; ExpB; NB; id(b); sig(sa1, sk(b) ,
         prf(sa1, prf(sa1, exp(ExpB, c(a, r1)), n(a, r2); NB),
         ExpB; exp(g, c(a, r1)); sa1; id(b)))),
    +(sig(sa1, sk(a), prf(sa1,
         prf(sa1, exp(ExpB, c(a, r1)), n(a, r2); NB),
         exp(g, c(a, r1)); ExpB; sa1; id(a)))) | nil]
   || empty || nil || nil
   || never (
    :: r3, r4 :: [ nil |
   -(sa1; exp(g, c(a, r1)); n(a, r2); id(a)),
   +(sa1; ExpB; NB; id(b); sig(sa1, sk(b),
         prf(sa1, prf(sa1, exp(ExpB, c(a, r1)), n(a, r2); NB),
         xpB; exp(g, c(a, r1)); sa1; id(b)))), nil]
   & S:StrandSet || K:IntruderKnowledge)
```

Although this authentication property failed, the intruder still cannot learn or forge that session key. Thus our composition result can be applied. Actually, we can verify that the AM$_{id}$ protocol satisfies a weaker authentication property: if both honest Alice and Bob finished their strand and passed the keys to protocol Q, then they must agreed on the same key. This property is described in the following attack pattern, in which both Alice and Bob finished their strand in the same section, but the key that Alice received is different from the key that is generated by Bob. For this property, Maude-NPA terminated without any attack being found.

```
eq ATTACK-STATE(10) =
 :: r1, r2 :: [ nil,
  +(sa1; exp(g, c(a, r1)); n(a, r2); id(a)),
  -(sa1; ExpB; NB; id(b); sig(sa1, sk(b) ,
      prf(sa1, prf(sa1, exp(ExpB, c(a, r1)), n(a, r2); NB),
      ExpB; exp(g, c(a, r1)); sa1; id(b)))),
  +(sig(sa1, sk(a), prf(sa1,
      prf(sa1, exp(ExpB, c(a, r1)), n(a, r2); NB),
      exp(g, c(a, r1)); ExpB; sa1; id(a)))) | nil]
   &
 :: r3, r4 :: [ nil ,
  -(sa1; ExpA; n(a, r2); id(a)),
 +(sa1; exp(g, c(b, r3)); n(b, r4); id(b);  sig(sa1, sk(b) ,
      prf(sa1, prf(sa1, exp(ExpA, c(b, r3)), n(a, r2); n(b, r4)),
      exp(g, c(b, r3)); ExpA; sa1; id(b)))),
 -(sig(sa1, sk(a), prf(sa1, prf(sa1, exp(ExpA, c(b, r3)),
      n(a, r2); n(b, r4)), ExpA; exp(g, c(b, r3)); sa1; id(a)))
 | nil] || ExpA != exp(g, c(a, r1))
```

We then check in QM with an admissible key abstraction that if the keys $SKEYID_a$, $SKEYID_e$ and $SKEYID_d$ generated by honest principals cannot be learned by the intruder, then the secrets that are used for constructing the final session keys cannot be learned by the intruder either. Here we only present the attack patterns for the initiator's strand. There are two other similar attack patterns for the responder's strand, that we omit.

```
 eq ATTACK-STATE(2)  =
    :: r, r', r1, r2 ::  [nil ,
   +(m(a, r); e(sa1, k1((a, r1)*(b, r2)), prf(sa1,
     k2((a, r1)*(b, r2)),m(a, r); qsa1; q(a, r')); qsa1; q(a, r'))),
   -(m(a, r); e(sa1, k1((a, r1)*(b, r2)), prf(sa1,
      k2((a, r1)*(b, r2)),m(a, r); q(a, r'); qsa1; MB); qsa1; MB)),
   +(m(a, r); e(sa1, k1((a, r1)*(b, r2)), prf(sa1,
      k2((a, r1)*(b, r2)),m(a, r); q(a, r'); MB)))
  | nil]
 || q(a, r') inI

eq ATTACK-STATE(3)  =
    :: r, r', r1, r2 :: [nil ,
   +(m(a, r); e(sa1, k1((a, r1)*(b, r2)), prf(sa1,
      k2((a, r1)*(b, r2)),m(a, r); qsa1; q(a, r')); qsa1; q(a, r'))),
   -(m(a, r); e(sa1, k1((a, r1)*(b, r2)), prf(sa1,
      k2((a, r1)*(b, r2)),m(a, r); q(a, r'); qsa1; MB); qsa1; MB)),
   +(m(a, r); e(sa1, k1((a, r1)*(b, r2)), prf(sa1,
      k2((a, r1)*(b, r2)), m(a, r); q(a, r'); MB)))  | nil]
  || MB inI
```

Another variant of Aggressive Mode uses public key encryption (AM$_{pk}$) in which both participants' identities and nonces are protected by public key encryption. The protocol proceeds as follows:

1. $A \rightarrow B : SA; g^{x_A}; \{N_A\}_{pk(B)}; \{ID(A)\}_{pk(B)}$
2. $B \rightarrow A : SA; g^{x_B}; \{N_B\}_{pk(A)}; \{ID(B)\}_{pk(A)};$
   $prf(SA, prf(SA, h(SA, N_A; N_B)), g^{x_B}; g^{x_A}; SA; ID(B))$
3. $A \rightarrow B : prf(SA, prf(SA, h(SA, N_A; N_B)), g^{x_A}; g^{x_B}; SA; ID(A))$

where N_A and N_B denote Nonces. The fact that security associations may specify algorithms is modeled by taking the security association SA as one of the arguments in the cryptographic functions prf and h.

Similarly, to check the secrecy of the QKeys, it is enough to check the secrecy of the keys *SKEYID* and $g^{x_A * x_B}$. We therefore check in the AM$_{pk}$ protocol that the keys *SKEYID* and $g^{x_A * x_B}$ exchanged between honest participants cannot be learned by the intruder. For this property, Maude-NPA terminated without any attack being found. Therefore we can conclude that AM$_{pk}$ is secrecy-enforcing. Here we only present the attack patterns for the responder's strand. There are two other similar attack patterns for the initiator's strand that we omit.

```
eq ATTACK-STATE(4)   =
    :: r', r3, r4 :: [ nil ,
   -(sa1; ExpA; pk(id(a), b); pk(NA, b)),
   +(sa1; exp(g, c(b, r3)); pk(id(b), a); pk(n(b, r4), a);
                prf(sa1, prf1(sa1, h(sa1, NA; n(b, r4))),
                    exp(g, c(b, r3)); ExpA; sa1; id(b))),
    -(prf(sa1,  prf1(sa1, h(sa1, NA; n(b, r4))),
                     ExpA; exp(g, c(b, r3)); sa1; id(a)))
          | nil ]   || exp(ExpA, c(b, r3)) inI

  eq ATTACK-STATE(5)   =
    :: r', r3, r4 :: [ nil ,
   -(sa1; ExpA; pk(id(a), b); pk(NA, b)),
   +(sa1; exp(g, c(b, r3)); pk(id(b), a);  pk(n(b, r4), a);
                    prf(sa1, prf1(sa1, h(sa1, NA; n(b, r4))),
                    exp(g, c(b, r3)); ExpA; sa1; id(b))),
   -(prf(sa1, prf1(sa1, h(sa1, NA; n(b, r4))),
                     ExpA; exp(g, c(b, r3)); sa1; id(a)))
     | nil ]  || h(sa1, NA; n(b, r4)) inI
```

We can also check that the authentication attack in protocol AM$_{id}$ does not happen in protocol AM$_{pk}$. The following attack pattern denotes that the initiator finished her strand with the responder without the corresponding responder's strand. Maude-NPA terminated without finding any attack for this property.

```
eq ATTACK-STATE(2) =
  :: r,  r1, r2 :: [ nil ,
    +(sa1; exp(g, c(a, r1)); pk(id(a), b); pk(n(a, r2), b)),
    -(sa1; ExpB; pk(id(b), a); pk(NB, a);
             prf(sa1, prf1(sa1,  h(sa1, n(a, r2); NB)),
             ExpB; exp(g, c(a, r1)); sa1; id(b))),
    +( prf(sa1, prf1(sa1, h(sa1, n(a, r2); NB)),
                exp(g, c(a, r1)); ExpB; sa1; id(a)))
    | nil]
    || empty || nil  || nil
    || never
    (:: r', r3, r4 :: [ nil |
    -(sa1; exp(g, c(a, r1)); pk(id(a), b); pk(n(a, r2), b)),
    +(sa1; ExpB; pk(id(b), a); pk(NB, a);
             prf(sa1, prf1(sa1, h(sa1, n(a, r2); NB)),
             ExpB; exp(g, c(a, r1)); sa1; id(b))), nil]
    & S:StrandSet || K:IntruderKnowledge)
```

Since we already checked in QM that if the keys $SKEYID_a$, $SKEYID_e$ and $SKEYID_d$ generated by honest principals cannot be learned by the intruder, then the secrets that are used for constructing the final session keys cannot be learned by the intruder either. Therefore, we can conclude that the intruder cannot learn the session key of honest principals generated by running protocol AM$_{pk}$ followed by protocol QM.

6.6.4 Simplified EAP-IKEv2 and IEEE802.11 Handshake

The protocol uses a version of IKEv2 [104] in the EAP framework as a parent protocol to agree on a master key. The master key is then used in a 4-way handshake protocol in IEEE 802.11i [105] to generate a pairwise transient keys. Due to performance issues, we had to simplify the protocol while preserving the security property that we intend to analyze. We refer to this simplified version the IKEv2-like protocol and 4-way-handshake-like protocol. The protocols proceed as follows:

1. $A \to B : g^{x_i}; N_A; SA$
2. $B \to A : g^{x_r}; N_B; SA$
3. $A \to B : e(SA, prf1(SA, N_A; N_B; g^{x_i * x_r}, ei), ID(A); sign(SA, SA; g^{x_i};$
   $N_A; N_B; mac(SA, prf1(SA, N_A; N_B; g^{x_i * x_r}, pi), ID(A)), A))$
4. $B \to A : e(SA, prf1(SA, N_A; N_B; g^{x_i * x_r}, er), ID(B); sign(SA, SA; g^{x_r};$
   $N_B; N_A; mac(SA, prf1(SA, N_A; N_B; g^{x_i * x_r}, pr), ID(B)), B))$

1. $A \to B : M_A$
2. $B \to A : M_B; SA; mac(SA, M_B, prf(SA, ID(B); ID(A); M_B; M_A, MSK))$
3. $A \to B : M_A; SA; mac(SA, M_A; SA, prf(SA, ID(B); ID(A); M_B; M_A, MSK))$
4. $B \to A : Finished; mac(SA, Finished, prf(SA, ID(B); ID(A); M_B; M_A, MSK))$

where N_A, N_B denote nonces in the IKEv2-like protocol, M_A, M_B denote nonces in the 4-way-handshake-like protocol, and $ID(A)$, $ID(B)$ denote identities. The fact that security associations may specify algorithms is modeled by taking the security association SA as one of the arguments in the cryptographic functions se, prf and mac.

To check that the intruder cannot learn the generated session key, we first check in the IKEv2-like protocol that the master key exchanged between honest participants cannot be learned by the intruder. Similarly, the master key is generated based on the seed key, and it never shows up in the protocol strands of the parent protocol. Therefore, it is enough to check the secrecy of the seed key. For this property, Maude-NPA terminated without any attack being found. Therefore we can conclude that the IKEv2-like protocol is secrecy-enforcing. The attack patterns analyzed by Maude-NPA are as follows:

```
eq ATTACK-STATE(2) =
:: r1, r2 :: [nil ,
   +(exp(g, c(a, r1)); n(a, r2); sa),
   -(exp(g, NSB) ; NB; sa),
   +(e(sa, id(a); sign(sa, exp(g, c(a, r1)), n(a, r2), NB, mac(sa,
    id(a), prf1(sa, exp(g, NSB*c(a, r1)), n(a, r2), NB, pi)), a),
               prf1(sa, exp(g, NSB*c(a, r1)), n(a, r2), NB, ei))),
   -(e(sa, id(b); sign(sa, exp(g, NSB), NB, n(a, r2), mac(sa, id(b),
        prf1(sa, exp(g, NSB*c(a, r1)), NB, n(a, r2), pr)), b),
              prf1(sa, exp(g, NSB*c(a, r1)), NB, n(a, r2), er)))
   | nil]
  || exp(g, NSB*c(a, r1)) inI

eq ATTACK-STATE(3) =
:: r3, r4 :: [nil,
  -(exp(g, NSA); NA; sa),
  +(exp(g, c(b, r3)); n(b, r4); sa),
  -(e(sa, id(a); sign(sa, exp(g, NSA), NA, n(b, r4), mac(sa,
     id(a), prf1(sa, exp(g, NSA*c(b, r3)), NA, n(b, r4), pi)), a),
          prf1(sa, exp(g, NSA*c(b, r3)), NA, n(b, r4), ei))),
  +(e(sa, id(b); sign(sa, exp(g, c(b, r3)), n(b, r4), NA, mac(sa,
     id(b), prf1(sa, exp(g, NSA*c(b, r3)), n(b, r4), NA, pr)), b),
           prf1(sa, exp(g, NSA*c(b, r3)), n(b, r4), NA, er)))
 | nil]
 || exp(g, NSA*c(b, r3)) inI
```

Even with this simplified IKEv2-like protocol, Maude-NPA still suffered from state explosion. To reduce the search space, we used never patterns [58]. One of the never patterns stops the intruder from trying to use the secret to construct other keys. The other one stops the intruder from trying to find the secret from another session of the same strand, since we know from the protocol that the keys are freshly generated in each session.

We also checked that the following authentication attack does not happen in the IKEv2-like protocol. The attack pattern describes the scenario where the initiator finished her strand without the corresponding responder finishing the responder's strand. This is specified by having an instance of the responder's strand in the never pattern. Maude-NPA terminated without finding any attack for this property. There is another similar attack pattern specifying that the responder finished her strand without the corresponding initiator's strand. Maude-NPA also terminated without finding any attack. We omit the details here.

```
 eq ATTACK-STATE(4)
  = :: r1, r2 ::
   [ nil,
    +(exp(g, c(a, r1)); n(a, r2); sa),
    -(exp(g, NSB); NB; sa),
    +(e(sa, id(a); sign(exp(g, c(a, r1)), n(a, r2), NB, mac(sa,
       id(a), prf1(sa, exp(g, NSB*c(a, r1)), n(a, r2), NB, pi)),a),
       prf1(sa, exp(g, NSB*c(a, r1)), n(a, r2), NB, ei))),
    -(e(sa, id(b); sign(sa, exp(g, NSB), NB, n(a, r2),mac(sa,
        id(b), prf1(sa, exp(g, NSB*c(a, r1)), NB, n(a, r2), pr)),b),
        prf1(sa, exp(g, NSB*c(a, r1)), NB, n(a, r2), er)))
    | nil]
   || empty || nil || nil
   || never(
    (:: r3, r4 ::
   [ nil |
    -(exp(g, c(a, r1)); n(a, r2); sa),
    +(exp(g, NSB); NB; sa),
    -(e(sa, id(a); sign(exp(g, c(a, r1)), n(a, r2), NB, mac(sa,
        id(a),prf1(sa, exp(g, NSB*c(a, r1)), n(a, r2), NB, pi)),a),
           prf1(sa, exp(g, NSB*c(a, r1)), n(a, r2), NB, ei))),
    +(e(sa, id(b); sign(sa, exp(g, NSB), NB, n(a, r2), mac(sa,
        id(b),prf1(sa, exp(g, NSB*c(a, r1)), NB, n(a, r2), pr)),b),
           prf1(sa, exp(g, NSB*c(a, r1)), NB, n(a, r2), er))),
    nil] & S:StrandSet || K:IntruderKnowledge)
```

For these authentication attack patterns, Maude-NPA also suffered from state explosion. To reduce the state space, we again used never patterns to reduce the search space. Since we already checked that the intruder cannot learn the secret keys of the communication of two honest principals, never patterns are added to prevent the intruder from trying to find the secret keys, or trying to use the secret key to construct other messages.

We then check in the 4-way-handshake-like protocol with an admissible key abstraction that if the master keys generated by honest principals cannot be learned by the intruder, then the pairwise transient key of honest principals is secure. Maude-NPA terminated without finding any attack for this property. Therefore, we can conclude that the intruder cannot learn the pairwise transient key of honest principals in the composed protocol. Here we only present the attack patterns for the responder's strand. There is another similar attack pattern for the initiator's strand that is omitted here.

```
eq ATTACK-STATE(3)
   =  :: r, r2, r3 :: [ nil,
        -(NA),
        +(q(b, r); qsa; mac(qsa, q(b, r); qsa,
     prf(qsa, id(b); id(a); q(b, r); NA, k((a, r2)*(b, r3))))),
   -(NA; qsa; mac(qsa, NA; qsa,
      prf(qsa, id(b); id(a); q(b, r); NA, k((a, r2)*(b, r3))))),
   +(fin; mac(qsa, fin,
     prf(qsa, id(b); id(a); q(b, r); NA, k((a, r2)*(b, r3)))))
  | nil]
 || prf(qsa, id(b); id(a); q(b, r); NA, k((a, r2)*(b, r3))) inI
```

6.7 RELATED WORK

The earliest work on modular verification of composed security protocols concentrated on *parallel* composition of protocols, where two or more protocols run in parallel but are not (or at least not intentionally) sharing any data. Conditions that make modular parallel composition possible, and that can be verified on the protocols running in isolation, were set forth by Guttman and Thayer in [42]. Later work by Cortier et al. in [43] develops syntactically checkable conditions that guarantee modular parallel composition.

One of the most important applications of modular verification, however, is the case of *sequential composition*, in which one protocol (the parent) provides information such as keys, that are used by another protocol (the child). Generally, results in this area describe a set of (mostly syntactic) conditions on parent and child to ensure that the two protocols do not interfere with each other, and a set of security properties, so that, if the conditions are satisfied, and parent and child each satisfy a security property separately, then so does the sequential composition. Modeling of this type of composition is generally done in one of two ways. One, also referred to as *vertical composition* in [106], is to think of the parent protocol as providing *secure channels* through which the child protocol communicates. The other is to have the parent protocol provide keys and other information directly to the child protocol.

Work on composition via channels includes that of Mödersheim et al. [106, 107] and that of Cheval et al. [101]. Mödersheim et al. do not impose explicit syntactic conditions on the composed protocols; instead they require that the protocols be secure under parallel composition, which can be verified either syntactically or non-syntactically. This allows them to reason about such constructs as self-composition. Cheval et al. do put syntactic conditions on the composed protocols, and also make use of a construct called an *encapsulation*, that provides a term constructed via cryptographic operations on inherited keys, that achieves a concrete representation of a secure channel. Encapsulations have multiple uses: they protect keys, guarantee that child and parent cannot be confused with each other, and, by providing authentication and confidentiality functionality, provided concrete representations of secure channels. In our work we show that they can be useful for modular verification for composition via inheritance as well.

Work on modular verification for composition via inheritance, to which our work belongs, includes [108, 109, 110, 111, 112, 113]. Like ours, these works concentrate on showing that, assuming certain syntactic conditions are satisfied, then, if the individual protocols satisfy certain security properties, the composed protocol satisfies a certain security property. These properties range from very specific (e.g. the security properties satisfied by a PKI in [112]) to very broad (e.g. the secrecy and authentication properties covered in [113]). In our work, we use simulation to prove results about reachability properties in general. In addition we note that, although [108] offers tool support via Scyther, and [111, 109, 110, 112] extend to equational theories, to the best of our knowledge ours is the first work to both support equational theories and offer tool support.

6.8 CONCLUDING REMARKS

We have presented a method for modular verification of sequentially composed protocols that enables verification of reachability properties that can be decomposed into reachability properties for the component protocols. In addition, our work both supports equational theories and offers tool support.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

This thesis extends the earlier publications [65, 91, 100]. It enhances Maude-NPA's capabilities for modeling and analyzing cryptographic protocols in terms of: (i) handling combinations of equational theories that could not be handled before; (ii) modeling and analyzing protocols with choices, which could not be model and analyzed naturally in Maude-NPA before; and (iii) analyzing in a modular way composed protocols whose search space is too large to be handled in Maude-NPA in practice.

7.1 FVP APPROXIMATIONS OF HOMOMORPHIC ENCRYPTION

We have developed a hierarchy of theories for approximating the algebraic property of homomorphic encryption and homomorphic encryption over abelian groups. All the theories we developed have the finite variant property, and therefore variant unification can be applied for analyzing protocols with these theories. The existence of finitary unification algorithms for many of these theories was not known before our work. The notion of variant complexity is introduced for the first time in the literature as a metric of performance. The experiments revealed the tradeoffs between the expressiveness of the theory and the performance of the analysis.

The work also points out a number of avenues for future work. In particular, it demonstrates that state space reduction techniques applied after a state is generated are likely not to be adequate by themselves for addressing performance issues when dealing with theories of high variant complexity. This points out the need for techniques that can be applied earlier in the state generation process. Another way to deal with theories not having the finite variant property would be to develop and implement new ways to integrate variant unification and specially implemented dedicated unification algorithm. Although there are general algorithms for combining unification algorithms for different theories, e.g., [89], they can be highly nondeterministic. Therefore careful design and implementation of such a combination algorithm would be needed to avoid a combination algorithm that is too slow to be practical.

7.2 PROTOCOL PROCESS ALGEBRA AND STRANDS WITH CHOICE

We have defined a protocol process algebra that supports both deterministic and nondeterministic choice. We also developed a choice extension of strand spaces, together with an operational semantics for choice in strand spaces. We proved the properties establishing a semantic connection between the process algebra and the extended strand space, and therefore made it possible to integrate this process algebra into Maude-NPA's existing implementation. This process algebra therefore provides a new specification language for Maude-NPA that is more natural for specifying choices in protocols.

The process algebra specification language for Maude-NPA also potentially helps us to relate the strand space model to other formal notations for protocol analysis based on process calculi, e.g. systems based on the applied pi calculus. This gives us a better basis for comparison with these systems.

By extending the strand space model, we have also provided a means for evaluating both equality and disequality predicates in the strand space model in Maude-NPA. This allows us to implement features such as type checking in Maude-NPA. This proved to be very helpful, for example, in our specification of the Rock-Scissors-Paper protocol as we described earlier. The expressiveness of Maude-NPA can be further enhanced by extending the types of predicates that can be evaluated, e.g., by including predicates for subsumption and their negations.

Another direction for future work is to include more features in the protocol process algebra. For example, extending the syntax and semantics of the process algebra to support protocol composition, or adding support for specifying indistinguishability properties. This can possibly lead to more natural ways to specify some protocols and properties in Maude-NPA.

Since our protocol process algebra supports a rich taxonomy of choice behaviors and can model systems that are highly nondeterministic, the application of our approach is not limited to cryptographic protocol analysis. For example, the technique of using choice variables modeling implicit nondeterministic choices could be used in modeling the behaviors of players in highly nondeterministic games.

7.3 MODULAR VERIFICATION OF SEQUENTIAL COMPOSITION FOR PRIVATE CHANNELS

We have presented a method for modular verification of sequentially composed protocols for private channels. This enables decomposing the verification of reachability properties of the composed protocol into reachability properties for the component protocols. We also support a large class of equational theories. We have performed experiments on a suite of non-trival protocols in Maude-NPA to illustrate and validate our approach.

One future direction is to further explore the properties of encapsulation to explicitly define the properties of the channels used by the child protocol. This should simplify the analysis of the child protocol even further and increase modularity.

Actually, most of the child protocols that we have seen are simpler than the associated key exchange protocol. According to the experiments that we have performed, after the key abstraction, Maude-NPA was able to analyze the child protocol without too much performance trouble. From the experiments that we have performed, we have observed performance difficulties in analyzing key exchange protocols even just by themselves. This is because the key exchange protocols usually involve many steps of communication together with cryptographic primitives having non-trival algebraic properties. Therefore, another future work direction would be investigating techniques specifically focused on improving the performance of analyzing key exchange protocols.

APPENDIX A: MAUDE SPECIFICATIONS FOR FVP THEORIES OF HOMOMORPHIC ENCRYPTION

A.1 THEORY OF BOUNDED HOMOMORPHISM.

In this section we present the Maude specifications for the main theories that we mentioned in Section 4.3.1.

```
fmod 3H is
sorts Key SingleMsg Msg .
subsort Key < SingleMsg < Msg .
op e : Msg Key -> Msg .
op e : SingleMsg Key  -> SingleMsg .
op _*_ : Msg Msg -> Msg [gather (e E)] .
op _*_ : SingleMsg SingleMsg -> Msg [gather (e E)] .

var X : Msg .
var K : Key .
var S1 S2 S3 : SingleMsg .
eq e(S1 * S2, K) = e(S1, K) * e(S2, K) .
eq e(S1 * S2 * S3, K) = e(S1, K) * e(S2, K) * e(S3, K)  .
endfm

fmod 3HD is including 3H .
op d : Msg Key -> Msg .
op d : SingleMsg Key  -> SingleMsg .

var X : Msg .
var K : Key .
var S1 S2 S3 : SingleMsg .
eq e(d(X, K), K) = X .
eq d(e(X, K), K) = X  .
eq d(S1 * S2, K) = d(S1, K) * d(S2, K) .
eq d(S1 * S2 * S3, K) = d(S1, K) * d(S2, K) * d(S3, K) .
endfm
```

```
fmod PK is
sort  Name Msg .
subsort  Name < Msg .

op pk : Msg Name -> Msg .
op sk : Msg Name -> Msg .

vars X Y Z : Msg .     vars A : Name .
eq sk(pk(X, A), A) = X .
eq pk(sk(X, A), A) = X .
endfm

fmod 3HPK is
including 3H + PK .
endfm

fmod 3HDPK is
including 3HD + PK .
endfm
```

A.2 THEORY OF HOMOMORPHIC ENCRYPTION OVER A FREE OPERATOR.

In this section we present the Maude specifications for the main theories that we mentioned in Section 4.3.2.1.

```
fmod H& is
sorts Key Keys Msg .
subsort Key < Keys < Msg .
op _*_ : Msg  Msg -> Msg  .
op _&_ : Keys  Keys -> Keys [assoc comm] .
op e : Msg Keys ->  Msg .

vars X Y Z : Msg  .
vars U V W : Keys .
eq e(e(X, V), U) =  e(X, U & V) .
eq e(X, U) * e(Y, U) =  e(X * Y, U)  .
eq e(X, U & V) * e(Y, U) = e(e(X, V) * Y, U) .
eq e(X, U) * e(Y, U & V) = e(X * e(Y, V), U) .
eq e(X, U & V) * e(Y, U & W) = e(e(X, V) * e(Y, W), U) .
endfm

fmod HD& is including H& .
sort Name Fresh .
subsort Name < Msg .
op d : Msg Keys ->  Msg .

vars X : Msg  .
vars U V W : Keys .

eq d(e(X, U), U) = X .
eq d(e(X, U & V), U) = e(X, V)   .
eq d(e(X, U), U & W) = d(X, W) .
eq d(e(X, U & V), U & W) = d(e(X, V), W)    .
endfm

fmod H&PK is
including H& + PK .
endfm

fmod HD&PK is
including HD& + PK .
endfm
```

A.3 THEORY OF HOMOMORPHIC ENCRYPTION OVER A PRE-GROUP.

In this section we present the Maude specifications for the main theories that we mentioned in Section 4.3.2.2.

```
fmod PG is
sorts Msg .
op _*_ : Msg Msg -> Msg .
op inv : Msg -> Msg .
op 1 : -> Msg .

vars X : Msg .

eq X * 1 = X .
eq 1 * X = X .
eq X * inv(X) = 1 .
eq inv(X) * X = 1 .
eq inv(inv(X)) = X .
eq inv(1) = 1 .
endfm

fmod PGAA is including PG .
sorts Nonce .
subsort Nonce < Msg .

vars X : Msg .
var N : Nonce .
eq (inv(N) * X) * N = X  .
eq N * (X * inv(N)) = X  .
eq (N * X) * inv(N) = X  .
eq inv(N) * (X * N) = X  .
endfm

fmod PGH& is including PG .
sorts Key Keys .
subsort Key < Keys < Msg .

op _&_ : Keys  Keys -> Keys  [assoc comm] .
op e : Msg Keys ->  Msg .

vars X Y Z : Msg  .
vars U V W : Keys .
eq e(e(X, V), U) =  e(X, U & V)         .
eq e(X, U) * e(Y, U) =  e(X * Y, U)     .
eq e(X, U & V) * e(Y, U) = e(e(X, V) * Y, U)      .
eq e(X, U) * e(Y, U & V) = e(X * e(Y, V), U)      .
eq e(X, U & V) * e(Y, U & W) = e(e(X, V) * e(Y, W), U)   .
eq e(1, U) = 1   .
eq inv(e(X, U)) = e(inv(X), U)   .
endfm

fmod PGHD& is including PGH& .
op d : Msg Keys ->  Msg .

var X : Msg .
vars U V W : Keys .
eq d(e(X, U), U) = X .
eq d(e(X, U & V), U) = e(X, V)   .
eq d(e(X, U), U & W) = d(X, W) .
eq d(e(X, U & V), U & W) = d(e(X, V), W)    .
eq d(1, U) = 1 .
endfm

fmod PGAAH& is including PGAA + PGH& .
endfm

fmod PGAAHD& is including PGAA + PGHD& .
endfm
```

A.4 THEORY OF HOMOMORPHIC ENCRYPTION OVER AN ABELIAN PRE-GROUP.

In this section we present the Maude specifications for the main theories that we mentioned in Section 4.3.2.3.

```
fmod APG is
sorts Msg .

op _*_ : Msg Msg -> Msg    [comm] .
op inv : Msg -> Msg .
op 1 : -> Msg .

vars X Y Z : Msg .

eq X * 1 = 1 .
eq X * inv(X) = 1 .
eq inv(inv(X)) = X .
eq inv(1) = 1 .
endfm

fmod APGAA is including APG .
sorts Nonce .
subsort Nonce < Msg .

vars X Y Z : Msg .
vars N : Nonce .
eq (inv(N) * X) * N = X .
eq (N * X) * inv(N) = X .
endfm

fmod APGH& is including APG .
sorts Key Keys .
subsort Key < Keys < Msg .

op _&_ : Keys Keys -> Keys [assoc comm] .
op e : Msg Keys -> Msg .

vars X Y Z : Msg .
vars U V W : Keys .
eq e(1, U) = 1 .
eq inv(e(X, U)) = e(inv(X), U) .
eq e(e(X, V), U) = e(X, U & V) .
eq e(X, U) * e(Y, U) = e(X * Y, U) .
eq e(X, U & V) * e(Y, U) = e(e(X, V) *  Y, U) .
eq e(X, U) * e(Y, U & V) = e(X * e(Y, V), U) .
eq e(X, U & V) * e(Y, U & W) = e(e(X, V) * e(Y, W), U) .
```

```
endfm

fmod APGHD& is including APGH& .
op d : Msg Keys -> Msg .

vars X : Msg .
vars U V W : Keys .
eq d(1, U) = 1 .
eq d(e(X, U), U) = X .
eq d(e(X, U & V), U) = e(X, V)  .
eq d(e(X, U), U & W) = d(X, W) .
eq d(e(X, U & V), U & W) = d(e(X, V), W) .
endfm

fmod APGAAH& is including APGH& + APGAA .
endfm

fmod APGAAHD& is including APGHD& + APGAA .
endfm
```

A.5 THEORY OF HOMOMORPHIC ENCRYPTION OVER PRE-XOR.

In this section we present the Maude specifications for the main theories that we mentioned in Section 4.3.2.4.

```
fmod PXOR is
sorts  Msg  .

op _*_ : Msg Msg -> Msg    [comm] .
op 1 : -> Msg .

vars X Y Z : Msg .
eq X * X = 1 .
eq X * 1 = X .
endfm
```

```
fmod PXORAA is including PXOR .
vars X Y Z : Msg .
eq X * (X * Y) = Y .
endfm

fmod PXORH is including PXOR .
sorts Key Keys .
subsort Key < Keys < Msg  .

op _&_ : Keys Keys -> Keys [comm assoc] .
op e : Msg Keys -> Msg .

vars X Y Z : Msg .
vars U V W : Keys .
eq e(1, U) = 1 .
eq e(e(X, V), U) = e(X, U & V)  .
eq e(X, U) * e(Y, U) = e(X * Y, U) .
eq e(X, U & V) * e(Y, U) = e(e(X, V) *  Y, U)  .
eq e(X, U) * e(Y, U & V) = e(X * e(Y, V), U) .
eq e(X, U & V) * e(Y, U & W) = e(e(X, V) * e(Y, W), U) .
endfm

fmod PXORHD is including PXORH .
op d : Msg Keys -> Msg .

vars X Y Z : Msg  .
vars U V W : Keys .
eq d(1, U) = 1 .
eq d(e(X, U), U) = X .
eq d(e(X, U & V), U) = e(X, V) .
eq d(e(X, U), U & W) = d(X, W) .
eq d(e(X, U & V), U & W) = d(e(X, V), W)  .
endfm

fmod PXORAAH is including PXORH + PXORAA .
endfm
```

```
fmod PXORAAHD is including PXORHD + PXORAA .
endfm
```

A.6 THEORY OF HOMOMORPHIC ENCRYPTION OVER TWO ABELIAN GROUPS.

In this section we present the Maude specifications for the main theories that we mentioned in Section 4.3.3.1.

```
fmod 2AGH is
sorts AG1 AG2 Msg .
subsort AG1 AG2 < Msg .

op _+_ : AG1 AG1 -> AG1 [comm assoc] .
op - : AG1 -> AG1        .
op 0 : ->  AG1 .
op _*_ : AG2 AG2 -> AG2  [comm assoc] .
op inv : AG2 -> AG2    .
op 1 : ->  AG2 .
op e  : AG1 -> AG2    .

vars X Y Z : AG1 .
vars P Q R : AG2 .
eq X + 0 = X .
eq X + (-(X)) = 0 .
eq X + (-(X) + Y) = Y .
eq -(-(X)) = X .
eq -(0) = 0 .
eq (-(X)) + (-(Y)) = -(X + Y) .
eq -(X + Y) + Y = -(X) .
eq -(-(X) + Y) = X + (-(Y)) .
eq -(X) + (-(Y) + Z ) = -(X + Y) + Z  .
eq -(X + Y) + (Y + Z) = -(X) + Z .

eq P * 1 = P .
eq P * inv(P) = 1 .
eq P * (inv(P) * Q) = Q .
eq inv(inv(P)) = P .
eq inv(1) = 1 .
eq inv(P) * inv(Q) = inv(P * Q) .
eq inv(P * Q) * Q = inv(P) .
eq inv(inv(P) * Q) = P * inv(Q) .
eq inv(P) * (inv(Q) * R) = inv(P * Q) * R  .
eq inv(P * Q) * (Q * R) = inv(P) * R .

eq e(0) = 1 .
eq inv(e( X)) = e(-(X)) .
eq inv(e( X) * P) = e(-(X)) * inv(P) .
eq e(X) * e(Y) = e(X + Y) .
eq e(X) * e(Y) * P = e(X + Y)  * P .
endfm

fmod 2AGHD is including 2AGH .
op d  : AG2 -> AG1   .

var X : AG1 .
eq d(1) = 0 .
eq d(e(X)) = X .
endfm
```

A.7 THEORY OF HOMOMORPHIC ENCRYPTION OVER TWO XOR OPERATORS.

In this section we present the Maude specifications for the main theories that we mentioned in Section 4.3.3.2.

```
fmod 2XORH is
sorts Xor1 Xor2 .

op _+_ : Xor1 Xor1 -> Xor1 [comm assoc] .
op 0 : -> Xor1 .
op _*_ : Xor2 Xor2 -> Xor2 [comm assoc] .
op 1 : -> Xor2 .
op e : Xor1 -> Xor2   .

vars X Y Z : Xor1 .
vars P Q R : Xor2 .
eq X + X = 0 .
eq X + X + Y = Y .
eq X + 0 = X .

eq P * P = 1 .
eq P * P * Q = Q  .
eq P * 1 = P .

eq e(0) = 1 .
eq e(X) * e(Y) = e(X + Y)  .
eq e(X) * e(Y) * P = e(X + Y)  * P   .
endfm

fmod 2XORHD is including 2XORH .
op d : Xor2 -> Xor1   .

var X : Xor1 .
eq d(1) = 0  .
eq d(e(X)) = X .
endfm
```

APPENDIX B: PROOF OF THEOREM 5.1

We present the proof of Theorem 5.1 in this section.

Proof. Since $P_{init} \rightarrow_{nil} P_{init}$ and $F_{init} \rightarrow_{nil} F_{init}$, therefore, $(P_{init}, F_{init}) \in \mathcal{H}$. We then prove that: for all PA-State Pst_n, and FW-State Fst_n, if $(Pst_n, Fst_n) \in \mathcal{H}$, and there exists a PA-State Pst_{n+1} such that $Pst_n \rightarrow_a Pst_{n+1}$, then there exists a FW-State Fst_{n+1} such that $Fst_n \rightarrow_a Fst_{n+1}$ and $(Pst_{n+1}, Fst_{n+1}) \in \mathcal{H}$. If $(Pst_n, Fst_n) \in \mathcal{H}$, by definition of the relation $\mathcal{H}$, there exists a label sequence α s.t. $P_{init} \rightarrow_\alpha Pst_n$ and $F_{init} \rightarrow_\alpha Fst_n$. Suppose that there exists state Pst_{n+1} such that $Pst_n \rightarrow_a Pst_{n+1}$. We prove by case analysis on label a that there exists Fst_{n+1} such that $Fst_n \rightarrow_a Fst_{n+1}$. The fact that $(Pst_{n+1}, Fst_{n+1}) \in \mathcal{H}$ then follows this by the definition of relation $\mathcal{H}$.

In the rest of this proof, L, L_1 and L_2 denote lists of messages, M, M' and m denote messages, P, Q and R denote processes, PS denotes a process configuration, SS denotes a set of constrained protocol strands, IK and IK' denote the set of messages in the intruder's knowledge.

1) $a = (ro, i, j, +m, 0)$: if $j > 1$, according to the semantics, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA++), the state Pst_n is of the form $\{(ro, i, j)\ (+M \cdot P)\ \&\ PS \mid \{IK\}\}$ s.t. there exists a ground substitution σ binding the choice variables in M and $m = M\sigma$, the state $Pst_{n+1} = \{(ro, i, j+1)\ P\sigma\ \&\ PS \mid \{m \in \mathcal{I}, IK\}\}$ and $m{\in}\mathcal{I} \notin IK$. Since $Pst_n\ \mathcal{H}\ Fst_n$, by Lemmas 5.1 and 5.2, Fst_n is of the form $\{(ro, i)\ [L]\ \&\ SS\ \&\ \{IK\}\}$ s.t. $(ro, i, j)\ (+M{\cdot}P)\ \mathcal{H}_{LP\_Str}\ (ro, i)\ [L]$. Let $(ro)\ [L_1, L_2]$ be a constrained strand in P_{CstrSS} s.t. there exists a ground substituion θ s.t. $L_1\rho_{ro,i}\theta = L$. By the definition of relation $\mathcal{H}_{LP\_Str}$ and mapping *toCstrSS*, the first message of L_2 is $+M'$, s.t. $M'\rho_{ro,i}\theta = M$. Then since $M\sigma = m$ and $m{\in}\mathcal{I} \notin IK$, the rule (F++) can be applied for the rewrite $Fst_n \rightarrow_a Fst_{n+1}$, where $Fst_{n+1} = \{(ro, i)\ [L, +m]\ \&\ SS\ \&\ \{m{\in}\mathcal{I}, IK\}\}$.

   If $j = 1$, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA&), there exists a process $(ro)\ (+M \cdot P)$ in P_{PA} and a ground substitution σ s.t. $M\rho_{ro,i}\sigma = m$. Since *toCstrSS*$(P_{PA}) = P_{CstrSS}$, by the definition of *toCstrSS*, for all strands of role ro in P_{CstrSS}, the first message is $+M$. Without loss of generality, let Pst_n be $\{PS \mid \{IK\}\}$, and Fst_n be $\{SS\ \&\ \{IK'\}\}$. Since the rule (PA&) can be applied, $m{\in}\mathcal{I} \notin IK$. By Lemma 5.2, $IK = IK'$. Moreover, by Lemma 5.1, *MaxStrId*(SS, ro) = *MaxProcId*(PS, ro), and since *MaxProcId*$(PS, ro) + 1 = i$, by applying the rule (F++&) we get $Fst_n \rightarrow_a Fst_{n+1}$.

2) $a = (ro, i, j, M\sigma, 0)$: similar to case 1.

3) $a = (ro, i, j, -m, 0)$: if $j > 1$, according to the semantics, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA-), Pst_n is of the form $\{(ro, i, j)\ (-M \cdot P)\ \&\ PS \mid \{m{\in}\mathcal{I}, IK\}\}$ s.t. $m =_{E_\mathcal{P}} M\sigma$ for some ground substitution σ and $Pst_{n+1} = \{(ro, i, j+1)\ P\sigma\ \&\ PS \mid \{m{\in}\mathcal{I}, IK\}\}$. Since $Pst_n\ \mathcal{H}\ Fst_n$, by Lemmas 5.1 and 5.2, $Fst_n = \{(ro, i)\ [L]\ \&\ SS\ \&\ \{m{\in}\mathcal{I}, IK\}\}$ s.t. $(ro, i, j)\ (-M \cdot P)\ \mathcal{H}_{LP\_Str}\ (ro)\ [L]$. Let $(ro)\ [L_1, L_2] \in P_{CstrSS}$ s.t. there exists a ground substitution θ s.t. $L_1\rho_{ro,i}\theta = L$, then by definition of $\mathcal{H}_{LP\_Str}$ and *toCstrSS*, the first message of L_2 is $-M'$ s.t. $M'\rho_{ro,i}\theta = M$. Since $m =_{E_\mathcal{P}} M\sigma$, rule (F-) can be applied to get the transition $Fst_n \rightarrow_a Fst_{n+1}$, where $Fst_{n+1} = \{(ro, i)\ [L, -m]\ \&\ SS\ \&\ \{m{\in}\mathcal{I}, IK\}\}$.

   If $j = 1$, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA&), there exists a process $(ro)\ (-M \cdot P)$ in P_{PA} and a ground substitution σ s.t. $M\rho_{ro,i}\sigma = m$. Without loss of generality, let Pst_n be $\{PS \mid \{IK\}\}$. Then $m{\in}\mathcal{I} \in IK$. Since *toCstrSS*$(P_{PA}) = P_{CstrSS}$, by the definition of *toCstrSS*, for all strands of role ro in P_{CstrSS}, the first message is $-M$. By Lemma 5.2, $m{\in}\mathcal{I}$ is in the intruder knowledge of Fst_n. Moreover, by Lemma 5.1, *MaxStrId*(SS, ro) = *MaxProcId*(PS, ro), and since *MaxProcId*$(PS, ro) + 1 = i$, by applying the rule (F-&) we get $Fst_n \rightarrow_a Fst_{n+1}$.

4) $a = (ro, i, j, T, 1)$: according to the transition rules, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PAif1). Therefore Pst_n is of the form $\{(ro, i, j)\ ((\mathit{if}\ c\ \mathit{then}\ P\ \mathit{else}\ Q) \cdot R)\ \&\ PS \mid \{IK\}\}$, $Pst_{n+1} = \{(ro, i, j+1)\ (P \cdot R)\ \&\ PS \mid \{IK\}\}$ and $c =_{E_\mathcal{P}} \mathit{true}$. Since $Fst_n\ \mathcal{H}\ Pst_n$, by Lemma 5.1, $Fst_n = \{(ro)\ [L]\ \&\ SS\ \&\ \{IK'\}\}$ s.t. $(ro, i, j)\ ((\mathit{if}\ c\ \mathit{then}\ P\ \mathit{else}\ Q) \cdot R)\ \mathcal{H}_{LP\_Str}\ (ro, i)\ [L]$. By the definition of the relation $\mathcal{H}_{LP\_Str}$ and the mapping *toCstrSS*, there exists $(ro)\ [L_1, \{C, 1\}, L_2] \in P_{CstrSS}$ and a ground substitution θ s.t. $L = L_1\rho_{ro,i}\theta$, and $C\rho_{ro,i}\theta = c$. Since $c =_{E_\mathcal{P}} \mathit{true}$, the rule (Fif) can be applied for the rewrite $Fst_n \rightarrow_a Fst_{n+1}$, where $Fst_{n+1} = \{\{(ro)\ [L, \{t, 1\}]\ \&\ SS\ \&\ \{IK'\}\}$

5) $a = (ro, i, j, T, 2)$: similar to case 4.

6) $a = (ro, i, j, ?, 1)$: if $j > 1$, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA?1). Therefore Pst_n is of the form $\{(ro, i, j)\ ((P\ ?\ Q) \cdot R)\ \&\ PS \mid \{IK\}\}$ and $Pst_{n+1} = \{(ro, i, j+1)\ (P \cdot R)\ \&\ PS \mid \{IK\}\}$. Since $Fst_n\ \mathcal{H}\ Pst_n$, by Lemma 5.1, $Fst_n = \{(ro, i)\ [L]\ \&\ SS\ \&\ \{IK'\}\}$ s.t. $(ro, i, j)\ ((P\ ?\ Q) \cdot R)\ \mathcal{H}_{LP\_Str}(ro, i)\ [L]$. By the definition of $\mathcal{H}_{LP\_Str}$ and *toCstrSS*, there is a strand $(ro, i)\ [L_1, \{?, 1\}, L_2] \in P_{CstrSS}$ s.t. $L = L_1\theta$. Therefore, rule (F?) can be applied for the rewrite $Fst_n \rightarrow_a Fst_{n+1}$, and $Fst_{n+1} = \{(ro, i)\ [L, \{?, 1\}]\ \&\ SS\ \&\ \{IK'\}\}$.

   If $j = 1$, $Pst_n \rightarrow_a Pst_{n+1}$ by applying rule (PA&). Therefore, there exists a process $(ro)\ ((P\ ?\ Q) \cdot R)$ in P_{PA}. Since *toCstrSS*$(P_{PA}) = P_{CstrSS}$, by the definition of *toCstrSS*, there is a strand of role ro whose first message is $(?, 1)$ in P_{CstrSS}. Moreover, by Lemma 5.1, $\mathit{MaxStrId}(SS, ro) = \mathit{MaxProcId}(PS, ro)$, and since $\mathit{MaxProcId}(PS, ro) + 1 = i$, by applying the rule (F?&) we get $Fst_n \rightarrow_a Fst_{n+1}$.

7) $a = (ro, i, j, ?, 2)$ similar to case 6.

Similarly, we can prove that for all PA-State Pst_n, and FW-State Fst_n, if $(\mathit{Pst}_n, \mathit{Fst}_n) \in \mathcal{H}$, and there exists a FW-State Fst_{n+1} such that $\mathit{Fst}_n \rightarrow_a \mathit{Fst}_{n+1}$, then there exists a PA-State Pst_{n+1} such that $\mathit{Pst}_n \rightarrow_a \mathit{Pst}_{n+1}$ and $(\mathit{Pst}_{n+1}, \mathit{Fst}_{n+1}) \in \mathcal{H}$

APPENDIX C: PROOF OF LEMMA 6.3

We present the proof of Lemma 6.3 in this section.

Proof. The proof is by case analysis on the transition rule rl. To simplify the presentation, we say that a transition rule is labeled with ℓ_P (resp. ℓ_Q) if the transition rule is associated with a strand in protocol P (resp. Q) in the forward semantics.

- If rl is of the form: $[L]\&SS\&\{IK\} \to [L, +M]\&SS\&\{IK, M\}$ if $M \notin IK$ or $SS\&\{IK\}$ $\to [+M]\&SS\&\{IK, M\}$ if $M \notin IK$, and is labeled with ℓ_P, then there exists a message m in the intruder knowledge of state $S'_{;}$ but not in the intruder knowledge of state $S_{;}$ such that $m =_{E_{;} \cup B_{;}} M\theta$ for some ground substitution θ.
  - If $m =_{E_{;} \cup B_{;}} m'$ such that m' is of sort Shared or any subsort of Shared, then, according to the assumption that sort Shared is a subsort of sort Public, m' can be generated by the intruder itself (i.e., Dolev-Yao strands) in protocol Q^α. Therefore there exists a state S'_{Q^α} such that $S_{Q^\alpha} \to_{DY} S'_{Q^\alpha}$ by applying Dolev-Yao strands in protocol Q^α such that m' is in the intruder knowledge of the state S'_{Q^α}. Thus $(S'_{;}, S'_{Q^\alpha}) \in \mathcal{H}_Q$.
  - if $m \in [T_{\mathsf{QKey}}]_{E_P \cup B_P}$, and $\alpha(m)$ is not in the intruder knowledge of state S_{Q^α}, according to the requirement of key abstraction α, $\alpha(m)$ can be generated by Dolev-Yao strands in protocol Q^α. Therefore there exists a state S'_{Q^α} such that $S_{Q^\alpha} \to S'_{Q^\alpha}$ by applying only Dolev-Yao strands and $\alpha(m)$ is in the intruder knowledge of S'_{Q^α}. Therefore $(S'_{;}, S'_{Q^\alpha}) \in \mathcal{H}_Q$.
  - Otherwise, $(S'_{;}, S_{Q^\alpha}) \in \mathcal{H}_Q$.
- if rl is of the form: $[\{I\}, L]\&SS\&\{IK\} \to [\{I\}, L, +M]\&SS\&\{IK, M\}$ if $M \notin IK$, rl is labeled with ℓ_Q and $[\{I\}, L, +M]$ is an honest protocol strand, then there exists a strand $[l]$ in state $S_{;}$, a strand $[\{u\}, l, +m]$ and message m in state $S'_{;}$, a strand $[\{I\}, L, +M, L'] \in SSpec_Q$, and disjoint ground substitutions σ and θ, such that $[l] =_{E_{;} \cup B_{;}} [L](\sigma \uplus \theta)$ and $m =_{E_{;} \cup B_{;}} M(\sigma \uplus \theta)$, where σ is the ground substitution representing the concrete synchronization of the input/output parameters. According to the construction of protocol Q and the semantics of protocol composition, the substitution θ does not include any substitution on QKeys, i.e., all the terms of sort QKey are grounded by substitution σ. By the definition of the relation $\mathcal{H}_Q$, there exists a strand $[l'] \in S_{Q^\alpha}$ such that $[l'] =_{E_Q \cup B_{Q^\alpha}} [l]\alpha$. By Lemma 6.2, since

$[l] =_{E_{;} \cup B_{;}} [L](\sigma \uplus \theta)$, $[l]\alpha =_{E_Q \cup B_{Q^\alpha}} [L](\sigma \uplus \theta)\alpha$. By the construction of protocol Q, all terms of sort QKey in L are variables, and since all the QKeys are grounded by substitution σ, therefore, $[l'] =_{E_Q \cup B_{Q^\alpha}} [l]\alpha =_{E_Q \cup B_{Q^\alpha}} ([L]\sigma\alpha)\theta$. By the construction of the protocol Q, m is of sort $\mathcal{E}$Msg or subsort of $\mathcal{E}$Msg. Together with the fact that $m =_{E_{;} \cup B_{;}} M(\sigma \uplus \theta)$, by Lemma 6.2, $m\alpha =_{E_Q \cup B_{Q^\alpha}} M(\sigma \uplus \theta)\alpha$. Similarly, since all QKeys are grounded by substitution σ, $m\alpha =_{E_Q \cup B_{Q^\alpha}} (M\sigma\alpha)\theta$. Since $[\{I\}, L, +M, L']$ is an honest protocol strand in protocol Q, $[L, +M, L'](\sigma'\alpha)$ is an honest protocol strand of protocol Q^α where σ' is a substitution such that $\sigma'\delta =_{E_P \cup B_P} \sigma$ with δ a ground substitution. By the definition of the key abstraction α, $\sigma\alpha =_{B_\alpha} (\sigma'\alpha)\delta'$ with $\delta' = \delta|_{Ran(\sigma'\alpha)}$. By the construction of protocol Q^α, $Dom(\sigma') \cap Dom(\theta) = \varnothing$ and $Ran(\sigma') \cap Dom(\theta) = \varnothing$. Therefore, the substitutions $\sigma'\alpha$ and δ' are both disjoint with the substitution θ. Therefore, $[l]\alpha =_{E_Q \cup B_{Q^\alpha}} ([L]\sigma'\alpha)(\delta' \uplus \theta)$, and $m\alpha =_{E_Q \cup B_{Q^\alpha}} (M\sigma'\alpha)(\delta' \uplus \theta)$. Therefore, there exists a state S'_{Q^α} such that $S_{Q^\alpha} \to S'_{Q^\alpha}$ by taking the transition $[l']\&SS_{Q^\alpha}\&\{IK_{Q^\alpha}\} \to [l', +m\alpha]\&SS_{Q^\alpha}\&\{IK_{Q^\alpha}, m\alpha\}$, or the transition $[l']\&SS_{Q^\alpha}\&\{IK_{Q^\alpha}\} \to [l', +m\alpha]\&SS_{Q^\alpha}\&\{IK_{Q^\alpha}\}$ if $m\alpha \in IK_{Q^\alpha}$, and $(S'_{;}, S'_{Q^\alpha}) \in \mathcal{H}_Q$.

- The case where rl is of the form $[\{I\}, L]\&SS\&\{IK\} \to [\{I\}, L, +M]\&SS\&\{IK\}$, and is associated to an honest protocol strand in protocol Q can be proved similarly.

- If rl is of the form: $[L]\&SS\&\{IK\} \to [L, +M]\&SS\&\{IK, M\}$, rl is labeled with ℓ_Q, and $[L, +M]$ is a Dolev-Yao strand, then there exists a strand $[l]$ in the state $S_{;}$, a strand $[l, +m]$ in state $S'_{;}$, the Dolev-Yao strand $[L, +M] \in SSpec_Q$ and a ground substitution θ, such that $[l] =_{E_{;} \cup B_{;}} [L]\theta$, and $m =_{E_{;} \cup B_{;}} M\theta$. By the definition of the relation $\mathcal{H}_Q$, there exists a strand $[l']$ in the state S_{Q^α} such that $[l'] =_{E_Q \cup B_{Q^\alpha}} [l]\alpha$. By Lemma 6.2, since $[l] =_{E_{;} \cup B_{;}} [L]\theta$, $[l'] =_{E_Q \cup B_{Q^\alpha}} ([L]\theta)\alpha$. By the construction of the protocol Q, the message m is of sort $\mathcal{E}$Msg or subsort of $\mathcal{E}$Msg. Therefore, by Lemma 6.2, since $m =_{E_{;} \cup B_{;}} M\theta$, $m\alpha =_{E_Q \cup B_{Q^\alpha}} (M\theta)\alpha$. Since all QKeys in $[L, +M]$ are variables, therefore, $[l'] =_{E_Q \cup B_{Q^\alpha}} [L](\theta\alpha)$, and $m\alpha =_{E_Q \cup B_{Q^\alpha}} M(\theta\alpha)$. Since $[L, +M]$ is a Dolev-Yao strand in protocol Q, the strand $[L, +M]$ is also a Dolev-Yao strand in protocol Q^α. Therefore, there exists a state S'_{Q^α} such that $S_{Q^\alpha} \to S'_{Q^\alpha}$ by taking the transition $[l']\&SS_{Q^\alpha}\&\{IK_{Q^\alpha}\} \to [l', +m\alpha]\&SS_{Q^\alpha}\&\{IK_{Q^\alpha}, m\alpha\}$, or the transition $[l']\&SS_{Q^\alpha}\&\{IK_{Q^\alpha}\} \to [l', +m\alpha]\&SS_{Q^\alpha}\&\{IK_{Q^\alpha}\}$ if $m\alpha \in \{IK_{Q^\alpha}\}$, and $(S'_{;}, S'_{Q^\alpha}) \in \mathcal{H}_Q$.

- The case where rl is of the form $[L]\&SS\&\{IK\} \to [L, +M]\&SS\&\{IK\}$ and is associated to a Dolev-Yao strand in protocol Q can be proved similarly.

- If rl is of the form: $[\{I\}, L]\&SS\&\{IK, M\} \to [\{I\}, L, -M]\&SS\&\{IK, M\}$, rl is labeled with ℓ_Q, and $[\{I\}, L, -M]$ is an honest protocol strand, then there exists a strand

$[\{u\}, l]$ in state $S_{;}$, a strand $[\{u\}, l, -m]$ in state $S'_{;}$, a strand $[\{I\}, L, -M, L'] \in SSpec_Q$ and ground substitutions σ and θ such that $[\{u\}, l, -m] =_{E_{;} \cup B_{;}} [\{I\}, L, -M](\sigma \uplus \theta)$, where σ is the substitution representing the synchronization of the input/output parameters. According to the construction of protocol Q and the semantics of the protocol composition, the substitution θ does not include any substitution on QKeys, i.e., all the terms of sort QKey are grounded by substitution σ. According to the definition of the relation $\mathcal{H}_Q$, there exists a strand $[l']$ in the state S_{Q^α} such that $[l'] =_{E_Q \cup B_{Q^\alpha}} [l]\alpha$. By Lemma 6.2, since $[l] =_{E_{;} \cup B_{;}} [L](\sigma \uplus \theta)$, $[l'] =_{E_Q \cup B_{Q^\alpha}} [L](\sigma \uplus \theta)\alpha$. Since all the QKeys in strand $[L]$ are variables and are grounded by the substitution σ, $[l'] =_{E_Q \cup B_{Q^\alpha}} ([L]\sigma\alpha)\theta$. Since $m \in \mathcal{T}_{\Sigma_{Q^\rho}}$, according to the definition $\mathcal{H}_Q$, there exists message m' in the intruder knowledge of S_{Q^α} such that $m' =_{E_Q \cup B_{Q^\alpha}} m\alpha$. Since $m =_{E_{;} \cup B_{;}} M(\sigma \uplus \theta)$, by Lemma 6.2, $m\alpha =_{E_Q \cup B_{Q^\alpha}} M(\sigma \uplus \theta)\alpha$. Similarly, since all QKeys in M are variables and are grounded by the substitution σ, $m' =_{E_Q \cup B_{Q^\alpha}} (M\sigma\alpha)\theta$. Since $[\{I\}, L, -M, L']$ is an honest protocol strand in protocol Q, the strand $[L, -M, L'](\sigma'\alpha)$ is in protocol Q^α, where σ' is a substitution such that $\sigma'\delta =_{E_P \cup B_P} \sigma$ with δ a ground substitution. According to the definition of the key abstraction α, $\sigma\alpha =_{B_\alpha} (\sigma'\alpha)\delta'$ with $\delta' = \delta|_{Ran(\sigma'\alpha)}$. By the construction of protocol Q^α, $Dom(\sigma') \cap Dom(\theta) = \varnothing$ and $Ran(\sigma') \cap Dom(\theta) = \varnothing$. Therefore, the substitutions $\sigma'\alpha$ and δ' are disjoint with the substitution θ. Therefore, $[l'] =_{E_Q \cup B_{Q^\alpha}} [l]\alpha =_{E_Q \cup B_{Q^\alpha}} ([L]\sigma'\alpha)(\delta' \uplus \theta)$, and $m' =_{E_Q \cup B_{Q^\alpha}} m\alpha =_{E_Q \cup B_{Q^\alpha}} (M\sigma'\alpha)(\delta' \uplus \theta)$. Therefore, there exists a state S'_{Q^α} such that $S_{Q^\alpha} \to S'_{Q^\alpha}$ by taking the transition $[l'] \& SS_{Q^\alpha} \& \{IK_{Q^\alpha}, m'\} \to [l', -m'] \& SS_{Q^\alpha} \& \{IK_{Q^\alpha}, m'\}$, and $(S'_{;}, S'_{Q^\alpha}) \in \mathcal{H}_Q$.

- If rl is of the form $[L] \& SS \& \{IK, M\} \to [L, -M] \& SS \& \{IK, M\}$, rl is labeled with ℓ_Q, and $[L]$ is a Dolev-Yao strand, then there exists a strand $[l] \in S_{;}$, a Dolev-Yao strand $[L, -M, L'] \in SSpec_Q$ and a substitution θ such that $[l, -m] =_{E_{;} \cup B_{;}} [L, -M]\theta$. According to the definition of the relation $\mathcal{H}_Q$, there exists a strand $[l']$ in S_{Q^α} such that $[l'] =_{E_Q \cup B_{Q^\alpha}} [l]\alpha$. By Lemma 6.2, since $[l] =_{E_{;} \cup B_{;}} [L]\theta$, $[l'] =_{E_Q \cup B_{Q^\alpha}} ([L]\theta)\alpha$. Since $m \in IK|_{\Sigma_{Q^\rho}}$, according to the definition $\mathcal{H}_Q$, there exists m' in the intruder knowledge of S_{Q^α} such that $m' =_{E_Q \cup B_{Q^\alpha}} m\alpha$. Since $m =_{E_{;} \cup B_{;}} M\theta$, by Lemma 6.2, $m' =_{E_Q \cup B_{Q^\alpha}} (M\sigma)\alpha$. Since all QKeys in $[L, -M]$ are variables, $l' =_{E_Q \cup B_{Q^\alpha}} l\alpha =_{E_Q \cup B_{Q^\alpha}} [L](\theta\alpha)$, and $m' =_{E_Q \cup B_{Q^\alpha}} m\alpha =_{E_Q \cup B_{Q^\alpha}} M(\theta\alpha)$. Therefore, there exists a state S'_{Q^α} such that $S_{Q^\alpha} \to S'_{Q^\alpha}$ by the transition $[l'] \& SS_{Q^\alpha} \& \{IK_{Q^\alpha}, m'\} \to [l', -m'] \& SS_{Q^\alpha} \& \{IK_{Q^\alpha}, m'\}$, and $(S'_{;}, S'_{Q^\alpha}) \in \mathcal{H}_Q$.

- The case where rl is of the form $SS \& \{IK, M\} \to [-M] \& SS \& \{IK, M\}$ and is a Dolev-Yao strand in protocol Q can be proved similarly.

- For other cases, $(S'_{;}, S_{Q^\alpha}) \in \mathcal{H}_Q$. The synchronization steps are covered by this case.

REFERENCES

[1] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using FDR," in *Tools and Algorithms for the Construction and Analysis of Systems.* Springer Berlin Heidelberg, 1996, pp. 147–166.

[2] C. Meadows, "Analysis of the internet key exchange protocol using the nrl protocol analyzer," in *IEEE Symposium on Security and Privacy*, 1999, pp. 216–231.

[3] C. Meadows, P. Syverson, and I. Cervesato, "Formal specification and analysis of the group domain of interpretation protocol using npatrl and the nrl protocol analyzer," *J. Comput. Secur.*, vol. 12, no. 6, pp. 893–931, Dec. 2004.

[4] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu, "Cryptographically verified implementations for TLS," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. Alexandria, Virginia, USA: ACM, 2008, pp. 459–468.

[5] D. Basin and C. Cremers, "Modeling and analyzing security in the presence of compromising adversaries," in *Proceedings of the 15th European Conference on Research in Computer Security*, ser. ESORICS'10. Athens, Greece: Springer-Verlag, 2010, pp. 340–356.

[6] C. Cremers, "Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2," in *Proceedings of the 16th European Conference on Research in Computer Security*, ser. ESORICS'11, Leuven, Belgium, 2011, pp. 315–334.

[7] C. Hoare, *Communicating Sequential Processes.* Prentice Hall, 1985.

[8] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Commun. ACM*, vol. 21, no. 12, pp. 993–999, Dec. 1978.

[9] J. Mitchell, M. Mitchell, and U. Stern, "Automated analysis of cryptographic protocols using Murphi," in *IEEE Symposium on Security and Privacy.* IEEE Computer Society, 1997.

[10] D. L. Dill, "The Murϕ verification system," in *Computer-Aided Verification, CAV 1996.* Springer-Verlag, 1996, pp. 390–393.

[11] L. C. Paulson, "Inductive analysis of the internet protocol TLS," in *Security Protocols, 6th International Workshop*, Cambridge, UK, April, 15-17 1998, pp. 1–12.

[12] B. Blanchet, "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules," in *14th IEEE Computer Security Foundations Workshop (CSFW-14).* Cape Breton, Nova Scotia, Canada: IEEE Computer Society, June 2001, pp. 82–96.

[13] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin, "The Tamarin prover for the symbolic analysis of security protocols," in *Computer-Aided Verification, CAV 2013*. Saint Petersburg, Russia: Springer, July 13-19 2013, pp. 696–701.

[14] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P. Heam, O. Kouchnarenko, J. Mantovani, S. Moedersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Vigano, and L. Vigneron, "The Avispa tool for the automated validation of internet security protocols and applications," in *Proceedings of Computer-Aided Verification, CAV 2005*. Springer-Verlag, 2005.

[15] C. J. F. Cremers, "The Scyther tool: Verification, falsification, and analysis of security protocols," in *Computer-Aided Verification, CAV 2008*, Princeton, NJ, USA, July 7-14 2008, pp. 414–418.

[16] S. Escobar, C. Meadows, and J. Meseguer, "Maude-NPA: Cryptographic protocol analysis modulo equational properties," in *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, ser. LNCS vol. 5705. Springer, 2009, pp. 1–50.

[17] M. Turuani, "The CL-Atse protocol analyser," in *Term Rewriting and Applications, 17th International Conference, RTA 2006*. Seattle, WA, USA: Springer, August 12-14 2006, pp. 277–286.

[18] A. Armando, L. Compagna, and Y. Lierler, "SATMC: a SAT-based model checker for security protocols," in *Proceedings of the 9th European Conference on Logic in Artificial Intelligence(JELIA'04)*, ser. Lecture Notes in Computer Science. Lisbon, Portugal: Springer, September 2004.

[19] D. Basin, S. Mödersheim, and L. Viganò, "OFMC: A symbolic model checker for security protocols," *International Journal of Information Security*, vol. 4, no. 3, pp. 181–208, 2005.

[20] D. Dolev and A. C.-C. Yao, "On the security of public key protocols (extended abstract)," in *FOCS*, 1981, pp. 350–357.

[21] C. Cremers, "Scyther - semantics and verification of security protocols," Ph.D. dissertation, Eindhoven University of Technology, 2006.

[22] D. L. Mitchell, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov, "Undecidability of bounded security protocols," in *Proc. Workshop on Formal Methods and Security Protocols*, 1999.

[23] M. Rusinowitch and M. Turuani, "Protocol insecurity with a finite number of sessions, composed keys is NP-complete," vol. 299. Essex, UK: Elsevier Science Publishers Ltd., April 2003, pp. 451–475.

[24] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani, "An NP decision procedure for protocol insecurity with XOR," in *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, ser. LICS '03. Washington, DC, USA: IEEE Computer Society, 2003.

[25] F. J. T. Fabrega, J. Herzog, and J. Guttman, "Strand spaces: What makes a security protocol correct?" *Journal of Computer Security*, vol. 7, pp. 191–230, 1999.

[26] S. Schneider, "Security properties and CSP," in *1996 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 6-8 1996, pp. 174–187.

[27] M. Abadi and A. D. Gordon, "A calculus for cryptographic protocols: The spi calculus," in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, ser. CCS '97. Zurich, Switzerland: ACM, 1997, pp. 36–47.

[28] P. Y. A. Ryan and S. A. Schneider, "An attack on a recursive authentication protocol. a cautionary tale," *Inf. Process. Lett.*, vol. 65, no. 1, pp. 7–10, 1998.

[29] O. Pereira and J.-J. Quisquater, "On the impossibility of building secure cliques-type authenticated group key agreement protocols," *Journal of Computer Security*, vol. 14, no. 2, pp. 197–246, 2006.

[30] S. Stubblebine and C. Meadows, "Formal characterization and automated analysis of known-pair and chosen-text attacks," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, pp. 571–581, 2000.

[31] D. Basin, S. Mödersheim, and L. Viganò, "An on-the-fly model-checker for security protocol analysis," in *Computer Security – ESORICS 2003, LNCS 2808.* Springer, 2003, pp. 253–270.

[32] R. Küsters and T. Truderung, "Using ProVerif to analyze protocols with Diffie-Hellman exponentiation," in *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009.* Port Jefferson, New York, USA: IEEE Computer Society, July 8-10 2009, pp. 157–171.

[33] R. Küsters and T. Truderung, "Reducing protocol analysis with xor to the xor-free case in the horn theory based approach," *J. Autom. Reasoning*, vol. 46, no. 3-4, pp. 325–352, 2011.

[34] M. Arapinis, S. Bursuc, and M. Ryan, "Reduction of equational theories for verification of trace equivalence: Re-encryption, associativity and commutativity," in *Principles of Security and Trust.* Springer, 2012, pp. 169–188.

[35] B. Schmidt, S. Meier, C. J. F. Cremers, and D. A. Basin, "Automated analysis of diffie-hellman protocols and advanced security properties," in *Proceedings of the 2012 IEEE 25th Computer Security Foundations Symposium*, ser. CSF '12, Washington, DC, USA, 2012, pp. 78–94.

[36] J. Dreier, L. Hirschi, S. Radomirovic, and R. Sasse, "Automated unbounded verification of stateful cryptographic protocols with exclusive OR," in *31st IEEE Computer Security Foundations Symposium, CSF 2018*, Oxford, United Kingdom, July 9-12 2018, pp. 359–373.

[37] S. Escobar, J. Meseguer, and R. Sasse, "Variant narrowing and equational unification," *Electr. Notes Theor. Comput. Sci.*, vol. 238, no. 3, pp. 103–119, 2009.

[38] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, ser. CSFW '01, Washington, DC, USA, 2001, pp. 82–96.

[39] S. Escobar, C. Meadows, and J. Meseguer, "A rewriting-based inference system for the nrl protocol analyzer and its meta-logical properties," *Theor. Comput. Sci.*, vol. 367, no. 1-2, pp. 162–202, 2006.

[40] A. Datta, A. Derek, J. C. Mitchell, and A. Roy, "Protocol composition logic (PCL)," *Electron. Notes Theor. Comput. Sci.*, vol. 172, pp. 311–358, 2007.

[41] L. Gong and P. Syverson, "Fail-stop protocols: An approach to designing secure protocols," in *Proc. of the 5th IFIP International Working Conference on Dependable Computing for Critical Applications (Urbana-Champaign, IL, Sept. 1995)*, R. K. Iyer, M. Morganti, W. K. Fuchs, and V. Gligor, Eds. IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 79–99.

[42] J. D. Guttman and F. J. Thayer, "Protocol independence through disjoint encryption," in *Proceedings of the 13th IEEE Workshop on Computer Security Foundations*, ser. CSFW '00, Washington, DC, USA, 2000, pp. 24–34.

[43] V. Cortier and S. Delaune, "Safely composing security protocols," *Formal Methods in System Design*, vol. 34, no. 1, pp. 1–36, 2009.

[44] T. Gro and S. Modersheim, "Vertical protocol composition," in *2011 IEEE 24th Computer Security Foundations Symposium*, June 2011, pp. 235–250.

[45] J. D. Guttman, "Establishing and preserving protocol security goals," *J. Comput. Secur.*, vol. 22, no. 2, pp. 203–267, March 2014.

[46] S. Ciobaca and V. Cortier, "Protocol composition for arbitrary primitives," in *23rd IEEE Computer Security Foundations Symposium*, July 2010, pp. 322–336.

[47] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago, "Sequential protocol composition in Maude-NPA," in *Proceedings of the 15th European Conference on Research in Computer Security*, ser. ESORICS'10, Athens, Greece, 2010, pp. 303–318.

[48] S. Santiago, S. Escobar, C. A. Meadows, and J. Meseguer, "Effective sequential protocol composition in Maude-NPA," *CoRR*, vol. abs/1603.00087, 2016.

[49] S. Escobar, J. Meseguer, and R. Sasse, "Folding variant narrowing and optimal variant termination," in *Rewriting Logic and its Applications, 8th International Workshop, WRLA 2010*, ser. Lecture Notes in Computer Science, P. Ölvezcsky, Ed. Springer, 2010.

[50] H. Comon-Lundh and S. Delaune, "The finite variant property: How to get rid of some algebraic properties," in *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21*, ser. Lecture Notes in Computer Science, J. Giesl, Ed., vol. 3467. Springer, 2005, pp. 294–307.

[51] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[52] F. Crazzolara and G. Winskel, "Composing strand spaces," in *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science*, 2002, pp. 97–108.

[53] S. B. Fröschle, "Adding branching to the strand space model," *Electr. Notes Theor. Comput. Sci.*, vol. 242, no. 1, pp. 139–159, 2009.

[54] I. Cervesato, N. A. Durgin, J. C. Mitchell, P. Lincoln, and A. Scedrov, "Relating strands and multiset rewriting for security protocol analysis," in *Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW '00*, 2000, pp. 35–51.

[55] J. Meseguer, "Conditional rewriting logic as a unified model of concurrency," *Theoretical Computer Science*, vol. 96, no. 1, pp. 73–155, 1992.

[56] TeReSe, Ed., *Term Rewriting Systems.* Cambridge University Press, Cambridge, 2003.

[57] S. Escobar, R. Sasse, and J. Meseguer, "Folding variant narrowing and optimal variant termination," *J. Log. Algebr. Program.*, vol. 81, no. 7-8, pp. 898–928, 2012.

[58] S. Escobar, C. Meadows, and J. Meseguer, *Maude-NPA Version 3.0*, 2017. [Online]. Available: http://maude.cs.illinois.edu/w/index.php?title=Maude_Tools:_Maude-NPA

[59] S. Escobar, C. Meadows, and J. Meseguer, "A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties," *Theoretical Computer Science*, vol. 367, no. 1-2, pp. 162–202, 2006.

[60] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago, "A rewriting-based forwards semantics for Maude-NPA," in *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security, HotSoS 2014.* ACM, 2014.

[61] "Maude Formal Environment," `http://maude.lcc.uma.es/MFE/`.

[62] H. Comon-Lundh and S. Delaune, "The finite variant property: How to get rid of some algebraic properties," in *Term Rewriting and Applications, 16th International Conference, RTA 2005*, ser. Lecture Notes in Computer Science, J. Giesl, Ed., vol. 3467, Nara, Japan, April 19-21, 2005, pp. 294–307.

[63] C. Bouchard, K. A. Gero, C. Lynch, and P. Narendran, "On forward closure and the finite variant property," in *FroCos*, 2013, pp. 327–342.

[64] A. Cholewa, J. Meseguer, and S. Escobar, "Variants of variants and the finite variant property," University of Illinois at Urbana-Champaign, http://hdl.handle.net/2142/47117, Tech. Rep., 2014.

[65] F. Yang, S. Escobar, C. A. Meadows, J. Meseguer, and P. Narendran, "Theories of homomorphic encryption, unification, and the finite variant property," in *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming*, ser. PPDP '14, Kent, Canterbury, United Kingdom, September 2014, pp. 123–133.

[66] R. Küsters and T. Truderung, "Reducing protocol analysis with xor to the xor-free case in the horn theory based approach," *Journal of Automated Reasoning*, vol. 46, no. 3-4, pp. 325–352, 2011.

[67] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO*, 1982, pp. 199–203.

[68] R. Rivest, L. Adleman, and M. Dertouzos, "On data banks and privacy homomorphism," in *Foundations of Security Computation*, R. DeMillo, R. Lipton, D. Dobkin, and A. Jones, Eds. ACM, 1978.

[69] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–88, 1981.

[70] S. Escobar, D. Kapur, C. Lynch, C. Meadows, J. Meseguer, P. Narendran, and R. Sasse, "Protocol analysis in Maude-NPA using unification modulo homomorphic encryption," in *Proceedings of the 13th International ACM SIGPLAN Symposium on Principles and Practices of Declarative Programming*, ser. PPDP '11, Denmark, 2011, pp. 65–76.

[71] H. O. Pflugfelder, *Quasigroups and Loops: Introduction.* Heideman, 1990.

[72] M. Schmidt-Schauß, "Unification in a combination of arbitrary disjoint equational theories," *J. Symb. Comput.*, vol. 8, no. 1/2, pp. 51–99, 1989.

[73] F. Baader and K. U. Schulz, "Unification in the union of disjoint equational theories: Combining decision procedures," in *Proceedings of the 11th International Conference on Automated Deduction: Automated Deduction*, ser. CADE-11. Springer, 1992, pp. 50–65.

[74] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *J. Log. Algebr. Program.*, vol. 75, no. 1, pp. 3–51, 2008.

[75] S. Mödersheim, "Models and methods for the automated analysis of security protocols," Ph.D. dissertation, ETH Zurich, 2007.

[76] S. Erbatur, S. Escobar, D. Kapur, Z. Liu, C. Lynch, C. A. Meadows, J. Meseguer, P. Narendran, S. Santiago, and R. Sasse, "Asymmetric unification: A new unification paradigm for cryptographic protocol analysis," in *24th International Conference on Automated Deduction*, ser. CADE-24, Lake Placid, NY, USA, June 9-14n 2013, pp. 231–248.

[77] E. Tidén and S. Arnborg, "Unification problems with one-sided distributivity," *J. Symb. Comput.*, vol. 3, no. 1/2, pp. 183–202, 1987.

[78] A. M. Marshall, "Equational unification: Algorithms and complexity with applications to cryptographic protocol analysis," Ph.D. dissertation, Albany, NY, USA, 2012.

[79] A. M. Marshall and P. Narendran, "New algorithms for unification modulo one-sided distributivity and its variants," in *Proceedings of the 6th International Joint Conference on Automated Reasoning*, ser. IJCAR'12. Springer, 2012, pp. 408–422.

[80] H. Lin, "Algorithms for cryptographic protocol verification in presence of algebraic properties," Ph.D. dissertation, Clarkson University, 2009.

[81] S. Anantharaman, H. Lin, C. Lynch, P. Narendran, and M. Rusinowitch, "Cap unification: application to protocol security modulo homomorphic encryption," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '10, Beijing, China, 2010, pp. 192–203.

[82] W. Nutt, "Unification in monoidal theories," in *Proceedings of the Tenth International Conference on Automated Deduction*, ser. CADE-10. Kaiserslautern, Germany: Springer, 1990, pp. 618–632.

[83] F. Baader and W. Nutt, "Adding homomorphisms to commutative/monoidal theories or how algebra can help in equational unification," in *Rewriting Techniques and Applications, RTA' 91*, 1991, pp. 124–135.

[84] F. Baader, "Unification in commutative theories, hilbert's basis theorem, and gröbner bases," *J. ACM*, vol. 40, no. 3, pp. 477–503, 1993.

[85] P. Narendran, "Solving linear equations over polynomial semirings," in *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science, LICS 96*, New Brunswick, New Jersey, USA, July 1996, pp. 466–472.

[86] D. Kapur, P. Narendran, and L. Wang, "An E-unification algorithm for analyzing protocols that use modular exponentiation," in *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, ser. RTA'03, Valencia, Spain, 2003, pp. 165–179.

[87] D. Kapur, P. Narendran, and L. Wang, “A unification algorithm for analysis of protocols with blinded signatures,” in *Mechanizing Mathematical Reasoning*, 2005, pp. 433–451.

[88] Z. Liu, “Dealing efficiently with exclusive or, abelian groups and homomorphism in cryptographic protocol analysis,” Ph.D. dissertation, 2012.

[89] F. Brandt, “Efficient cryptographic protocol design based on distributed el gamal encryption,” in *Proceedings of the 8th International Conference on Information Security and Cryptology*, ser. ICISC’05, Seoul, Korea, 2006, pp. 32–47.

[90] J.-M. Hullot, “A catalogue of canonical term rewriting systems,” SRI International, Tech. Rep. CSL-113, 1980.

[91] F. Yang, S. Escobar, C. A. Meadows, J. Meseguer, and S. Santiago, “Strand spaces with choice via a process algebra semantics,” in *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming*, ser. PPDP ’16, Edinburgh, United Kingdom, September 2016, pp. 76–89.

[92] E. Rescorla, “The transport layer security (TLS) protocol version 1.3,” IETF, Tech. Rep., 2016.

[93] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago, “Symbolic protocol analysis with disequality constraints modulo equational theories,” in *Programming Languages with Applications to Biology and Security*, 2015, pp. 238–261.

[94] M. Abadi and C. Fournet, “Mobile values, new names, and secure communication,” in *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2001, pp. 104–115.

[95] R. Milner, *Communicating and mobile systems - the Pi-calculus.* Cambridge University Press, 1999.

[96] J. Goubault-Larrecq, C. Palamidessi, and A. Troina, “A probabilistic applied pi-calculus,” in *Programming Languages and Systems, 5th Asian Symposium, APLAS 2007*, 2007, pp. 175–190.

[97] C. Olarte and F. D. Valencia, “The expressivity of universal timed ccp: Undecidability of monadic fltl and closure operators for security,” in *Proceedings of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, ser. PPDP ’08. Valencia, Spain: ACM, 2008, pp. 8–19.

[98] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin, “The TAMARIN prover for the symbolic analysis of security protocols,” in *Computer Aided Verification, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings.*

[99] C. Cremers, M. Horval, S. Scott, and T. van der Merwe, “Automated analysis and verification of TLS 1.3:0-RTT, resumption and delayed authentication,” in *2016 IEEE Symposium on Security and Privacy*, May 2016, pp. 470–485.

[100] F. Yang, S. Escobar, C. A. Meadows, and J. Meseguer, "Modular verification of sequential composition for private channels in Maude-NPA," in *Security and Trust Management - 14th International Workshop, STM 2018*, Barcelona, Spain, September 2018, pp. 20–36.

[101] V. Cheval, V. Cortier, and E. le Morvan, "Secure refinements of communication channels," in *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015*, December 16-18, 2015, pp. 575–589.

[102] I. C. A. Organization, "Machine readable travel documents (ICAO Doc 9303)."

[103] D. Harkins and D. Carrel, "The internet key exchange (IKE)," ProposedStandard, RFC Editor, RFC 2409, Nov 1998.

[104] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet key exchange protocol version 2 (IKEv2)," Internet Standard, RFC Editor, RFC 7296, Oct 2014.

[105] "IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," *IEEE Std 802.11-2016*, pp. 1–3534, Dec 2016.

[106] T. Groß and S. Mödersheim, "Vertical protocol composition," in *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011*, Cernay-la-Ville, France, June 27-29, 2011, pp. 235–250.

[107] S. Mödersheim and L. Viganò, "Sufficient conditions for vertical composition of security protocols," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '14, 2014, pp. 435–446.

[108] S. Andova, C. J. F. Cremers, K. Gjøsteen, S. Mauw, S. F. Mjølsnes, and S. Radomirovic, "A framework for compositional verification of security protocols," *Inf. Comput.*, vol. 206, no. 2-4, pp. 425–459, 2008.

[109] Ştefan Ciobâcă and V. Cortier, "Protocol composition for arbitrary primitives," in *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010*, Edinburgh, United Kingdom, July 17-19, 2010, pp. 322–336.

[110] C. Chevalier, S. Delaune, S. Kremer, and M. D. Ryan, "Composition of password-based protocols," *Formal Methods in System Design*, vol. 43, no. 3, pp. 369–413, 2013.

[111] M. Arapinis and S. Delaune, "Composing security protocols: From confidentiality to privacy," in *Principles of Security and Trust - 4th International Conference, POST 2015*, London, UK, April 11-18, 2015, pp. 324–343.

[112] V. Cheval, V. Cortier, and B. Warinschi, "Secure composition of PKIs with public key protocols," in *30th IEEE Computer Security Foundations Symposium, CSF 2017*, Santa Barbara, CA, USA, August 21-25, 2017, pp. 144–158.

[113] A. V. Hess, S. A. Mödersheim, and A. D. Brucker, "Stateful protocol composition," in *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018*, Barcelona, Spain, September 3-7, 2018, pp. 427–446.