Protocol Analysis in Maude-NPA Using Unification Modulo Homomorphic Encryption *

Santiago Escobar
DSIC-ELP, Universidad Politécnica de Valencia, Spain
sescobar@dsic.upv.es

Deepak Kapur
University of New Mexico, Albuquerque, NM, USA
kapur@cs.unm.edu

Christopher Lynch
Clarkson University, Potsdam, NY, USA
clynch@clarkson.edu

Catherine Meadows
Naval Research Laboratory, Washington DC, USA
meadows@itd.nrl.navy.mil

José Meseguer
University of Illinois at Urbana-Champaign, USA
meseguer@illinois.edu

Paliath Narendran
University at Albany-SUNY, Albany, NY, USA
dran@cs.albany.edu

Ralf Sasse
University of Illinois at Urbana-Champaign, USA
rsasse@illinois.edu

Abstract

A number of new cryptographic protocols are being designed to secure applications such as video-conferencing and electronic voting. Many of them rely upon cryptographic functions with complex algebraic properties that must be accounted for in order to be correctly analyzed by automated tools. Maude-NPA is a cryptographic protocol analysis tool based on narrowing and typed equational unification which takes into account these algebraic properties. It has already been used to analyze protocols involving bounded associativity, modular exponentiation, and exclusive-or. All of the above can be handled by the same general *variant*-based equational unification technique. However, there are important properties, in particular homomorphic encryption, that cannot be handled by variant-based unification in the same way. In these cases the best available approach is to implement specialized unification algorithms and combine them within a modular framework. In this paper we describe how we apply this approach within Maude-NPA, with respect to encryption homomorphic over a free operator. We also describe the use of Maude-NPA to analyze several protocols using such an encryption operation. To the best of our knowledge, this is the first implementation of homomorphic encryption of any sort in a tool for verifying the security of a protocol in the presence of active attackers.

Categories and Subject Descriptors C.2.2 [*Computer-communication Networks*]: Network Protocols; D.2.4 [*Software Engineering*]: Software/Program Verification; D.3.2 [*Programming Languages*]: Language Classifications; D.4.6 [*Operating Systems*]: Security and Protection; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs

General Terms Protocol verification, unification, homomorphism

1. Introduction

With the increased use of computer networks and online transactions, more and more complex cryptographic protocols using encryption techniques with sophisticated arithmetic properties are being designed to secure applications such as video-conferencing and electronic voting. Many of these protocols are known to be secure if the arithmetic properties are not exploited, but can be broken if they are. Examples include the recursive authentication protocol proposed by Bull [11], broken by Ryan and Schneider [40] by exploiting the properties of the exclusive-or operation, and a group key protocol based on group Diffie-Hellman [5], broken by Pereira and Quisquater [39] using associative-commutative properties of modular exponentiation. Traditional Dolev-Yao modeling of protocols in which the underlying cryptographic functions are treated as a black box is thus often untrustworthy; the box must be pried open enough to account for the algebraic properties of the cryptographic functions.

Model checking has been an effective tool in cryptographic protocol analysis, and a number of model checkers have been built [4, 9, 20, 32, 35]. Cryptographic protocol analysis model checkers discover attacks by generating and analyzing the search space of possible states arising from the execution of a given protocol, taking into consideration the functions of the principals as well

* S. Escobar has been partially supported by the EU (FEDER) and the Spanish MEC/MICINN under grant TIN 2010-21062-C02-02, and by Generalitat Valenciana PROMETEO2011/052. The following authors have been partially supported by NSF: S. Escobar, J. Meseguer and R. Sasse under grants CCF 09-05584, CNS 09-04749, and CNS 09-05584 ; D. Kapur under grant CNS 09-05222; C. Lynch and C. Meadows under grant CNS 09-05378, and P. Narendran under grant CNS 09-05286.

PDPP'11, July 20–22, 2011, Odense, Denmark.

as the capabilities of the intruder(s)[1]. In this paper, we propose a symbolic approach to infinite-state, unbounded-sessions model checking based on combining unification algorithms to handle various properties of operations involved in a cryptographic protocol. Roughly speaking, a protocol is modeled as a state transition system, with a possible execution of the protocol being a sequence of state transitions; in any given state, any of the actors/principals is allowed to perform any of the operations on data in its possession.

There has been a growing body of research in extending these model checkers to reason about different types of equational theories [7, 9, 10, 13, 18, 44]. However, there has been one class of theories that has proven to be rather difficult to incorporate into protocol analysis tools. These are theories involving homomorphic operators, that is operators (usually representing encryption functions) that distribute over some other operator, which is usually associative-commutative or an Abelian group operator. Homormorphic encryption of this sort has many potential applications, including blinded signatures that can be used in electronic cash, private database retrieval, and electronic voting.

This paper describes the first steps in our research program with respect to homomorphic encryption: the adaptation of an algorithm for unification modulo an equational theory arising from encryption homomorphic over a free operator for use in Maude-NPA, and the use of the tool for analyzing protocols that satisfy this theory. Maude-NPA [21, 22] is designed to take into account the algebraic properties of cryptographic protocols. It makes use of equational unification to meet its goals. The approach of Maude-NPA up to this point has been to make use of *folding variant narrowing* [23] to perform equational unification. Suppose that the operators used in the protocol obey an equational theory $E_{\mathcal{P}}$. To do this, we divide the equational theory $E_{\mathcal{P}}$ describing the algebraic properties obeyed by the cryptosystem into $\Delta \cup Ax$, where Δ is a set of rewrite rules and Ax is a set of axioms such that Δ has the *finite variant property* modulo Ax [17, 23]. In that case, it is possible to apply a version of narrowing modulo Ax called folding variant narrowing to compute $E_{\mathcal{P}}$ unifiers. The advantage of using folding variant narrowing is that it is possible to have a general-purpose procedure that applies to many different theories. The only place we need special-purpose unification algorithms are the algorithms for Ax. We note that, when we find efficient algorithms for specific theories or combinations of theories, these theories can be moved into Ax if they satisfy the appropriate conditions, thus allowing us to increase the efficiency of our approach as fast special-purpose algorithms become available. In the most extreme case Δ could be empty.

In the past we have restricted ourselves to the case in which Ax is the free theory or associativity-commutativity (AC). In these cases we have been able to rely on unification algorithms already built into Maude [16]. However, it is well known that finite variance does not hold for homomorphic encryption, whether Ax is the free theory or AC [17]. In this case homomorphic encryption unification must be implemented in Ax. We use the unification algorithm of Anantharaman et al. [2] for this. In [2] it is applied to a procedure known as *cap unification*, which combines unification and deducibility, but it can also be used by itself, as we do here.

Contributions

The main contribution of this paper is that it demonstrates how a special-purpose unification algorithm for homomorphic encryption over a free operator is implemented and integrated with Maude-NPA. To the best of our knowledge, this represents the first implementation and integration of a special-purpose unification algorithm within a cryptographic protocol analysis tool, and the first implementation of any complete algorithm for reasoning about homomorphic encryption of any sort within a tool that can reason about protocol security in the presence of an active attacker. The only other implementation of homomorphic encryption, even over a free operator, available in a tool is in the intruder deduction tool YAPA [8], and can only prove security against passive attacks when used by itself. (See Section 2.2 for more details.)

In more detail, our implementation consists of:

1. An implementation in Full Maude of the algorithm described in [2] for homomorphic encryption. Full Maude [16] is an extension of Maude written in Maude that takes advantage of its reflective capabilities.
2. The derivation of an order-sorted unification algorithm for homomorphic encryption (see Section 6).
3. Application of the Maude-NPA infrastructure for combination of unification algorithms à la Baader and Schultz [6] (see Section 3.2) to the integration of the algorithm with the Maude-NPA search engine.
4. Testing the integration of all these new features into Maude-NPA by analyzing four protocols using homomorphic encryption (see Section 7).
5. Documenting how these four protocols behave in the Maude-NPA, which is relevant to demonstrate the feasibility of the whole approach (see Section 7).

Plan of the paper

The rest of the paper is organized as follows. In Section 2, we give a survey of the various means by which algebraic properties are addressed in automated cryptographic protocol analysis. In Section 3 we describe how equational unification is implemented in Maude-NPA. In Section 4 we provide motivating examples we will use throughout the remainder of this paper. In Section 5 we describe how Maude-NPA uses equational unification for backwards search, illustrating our explanation with examples from Section 4. In Section 6 we describe the unification algorithm we use for the homomorphic theory and its integration into Maude-NPA. In Section 7 we describe the analysis of the protocols introduced in Section 4. In Section 8 we conclude the paper and discuss future work.

2. Handling Algebraic Properties of Crypto-Algorithms

One challenge in computing the various state transitions is dealing with the algebraic properties of the cryptographic algorithms themselves. These range from the fact that decryption with a key cancels out encryption with the same key, expressible by the equation $dec(enc(m, k), k) = m$, through the Abelian group properties of algorithms based on exponentiation and/or elliptic curves, all the way to the property of homomorphism over an Abelian group possessed by many of the algorithms used for privacy-preserving computation. There are three classes of procedures that have been developed for dealing with algebraic properties: (i) augmented intruder inference rules, (ii) deducibility algorithms, and (iii) equational unification.

2.1 Augmented Inference rules

When specifying a cryptographic protocol, one normally specifies a set of inference rules that describe the operations that an intruder can perform. Thus, one would specify an inference rule that says that, if an intruder knows a message and a key, then he can construct the encryption of the message with the key. These inference rules can also be augmented to describe the consequences of equational

[1] We note that the terms "intruder" and "attacker" are used interchangeably in the literature

properties. For example, the encryption-decryption equation given above could be represented by the following inference rule:

$$\frac{enc(m,k) \in \mathcal{I}, k \in \mathcal{I}}{m \in \mathcal{I}}$$

where $\mathcal{I}$ stands for the set of terms known to the intruder.

The problem is that this method is often incomplete. Consider the following protocol:

$$1.\ A \rightarrow B : M \qquad 2.\ B \rightarrow A : dec(M, key(B))$$

The inference rule fails to predict what happens when $M = enc(X, key(B))$. In this case, A would wind up sending a cleartext message X, but this is because of the action of A's decryption operation, not because of the application of an inference rule by the intruder. Thus, the inference rules are not complete.

Although there are subclasses of protocols for which given sets of inference rules are sound (see for example [33, 37]), this must be carefully worked out for each case. However, if successful, augmented inference rules have the advantage that they can be used with tools that do not support the equational theory that the inference rules represent. Indeed, Küsters and Truderung show in [30, 31] how their inference systems for Diffie-Hellman exponentiation and exclusive-or can be used for a restricted class of protocols using Blanchet's ProVerif tool, which by itself only supports associative-commutative properties in a limited way.

We note that there is at least one example of applying augmented inference rules to homomorphic encryption. In [29], Kremer and Ryan define a set of intruder inference rules which they use, together with the ProVerif protocol analysis tool, to analyze protocols using cipher block chaining, which has a prefix homomorphism property. This approach is subject to the same incompleteness that we discussed above. However, it may be possible that completeness can be proved for the specific protocols analyzed in that paper.

2.2 Deducibility algorithms

These are algorithms for determining whether an intruder can deduce a term from a set of terms already in its possession, given that terms obey a given equational theory. In this case, one starts with a set T of terms known to the intruder, a term t the intruder is trying to learn, a set of inference rules describing operations the intruder can perform, and an equational theory E. The deducibility algorithm is a procedure for determining whether or not the intruder can derive a term from s from T such that $s \equiv_E t$. A number of algorithms have been developed for different classes of equational theories, including associative-commutative and homomorphic operators [1]. A survey of deducibility with respect to equational theories may be found in [12].

In particular, algorithms for a class of theories known as subterm convergent (convergent theories for which the right-hand side is a either an irreducible ground term or subterm of the left-hand side) have been developed in [15] and for a larger class of convergent theories that include encryption homomorphic over a free operator [8] and have been developed and implemented in the tools KISS and YAPA, respectively. These tools when used by themselves can only prove security against a passive intruder, who only spies upon message traffic but does not further interact with the protocol. However they can also be interfaced with other tools that use deducibility to reason about security against an active attacker who reads, alters, redirects, and deletes traffic as well as creating its own messages. The *intruder with caps* approach of [3] also uses deducibility algorithms for classes of equational theories that extend the subterm convergent class. The tools OFMC [7], and CL-Atse [44] all make use of deducibility and provide some support for equational theories, in the case of OFMC and CL-Atse those governing Diffie-Hellman and exclusive-or.

A limitation of using deducibility is that it requires a complete description of the terms an intruder knows at a given state. This is fine for tools that generate states in a forward fashion, but it does not work as well for tools such as Maude-NPA, which generate states on the fly in a backwards manner and thus only are aware of some of the terms the intruder knows at any point in time.

2.3 Equational Unification

Unification of two terms s and t modulo an equational theory E is the process of finding substitutions σ to the variables in s and t making them equal modulo E. We call a substitution σ to the variables in s and t an *E-unifier* of s and t, or a *unifier modulo E*, if and only if $\sigma(t) =_E \sigma(s)$, with $=_E$ the provable E-relation. We say that a set Θ of E-unifiers is a *complete set of E-unifiers* of s and t if for any unifier τ, there is a $\sigma \in \Theta$ such that $\tau =_E \rho\sigma$ for some substitution ρ away from the variables of s and t. Thus, a complete set of E-unifiers characterizes all the unifiers of two terms.

An advantage of E-unification over deducibility is that it can be applied even on incomplete information, since this incomplete information can be represented by variables. Thus, it applies not only to both forward and backwards search, but to constraint-based searches that can proceed from any direction (e.g. Comon-Lundh and Shmatikov's application of unification to constraint-based analysis in protocols that use exclusive-or [18], and Chevalier et al.'s use of constraints to verify presence of subterms when equational theories are present [13]). In particular, Chevalier and Rusinowitch [14] have developed a decision procedure for constraint-based protocol analysis over unions of disjoint intruder theories that is based on Baader and Schultz's [6] and Schmidt-Schauss' [42] algorithms for combining unification algorithms over disjoint theories and has similar complexity.

Early work on cryptographic protocol analysis modulo equational theories relied on existing techniques such as *narrowing*, which requires that the equational theory be expressed as a set of convergent rewrite rules. Narrowing a term consists of identifying a non-variable subterm of it that can be syntactically unified with the left-hand side of a rewrite rule and replacing it with the corresponding substitution instance of the right-hand side. This process proceeds until no further narrowing steps can be applied. Narrowing-based unification of two terms s and t is the narrowing of $eq(s,t)$ together with the rewrite rules expressing the given equations plus the additional rewrite rule $eq(x,x) \rightarrow true$. If one backtracks upon failure and upon each successful unification, one obtains a finite complete set of unifiers if the narrowing procedure terminates.

The original NRL Protocol Analyzer [35], upon which Maude-NPA is based, relied on a type of narrowing known as *basic narrowing* [26]. However, although theories such as the cancellation of encryption and decryption can be handled by basic narrowing alone, it does not apply to theories that involve associative-commutative properties. Thus, although Maude-NPA can and does make use of narrowing, it implements it in a very different way.

3. Unification in Maude-NPA

Maude-NPA employs a number of unification strategies, described below.

3.1 Folding Variant Narrowing

One way of dealing with theories containing associativity-commutativity (AC) is to split the theory E into two disjoint pieces, $E = \Delta \cup AC$, so that Δ is confluent, terminating and coherent modulo AC. However, narrowing modulo AC doesn't terminate for many theories of interest to cryptographic protocol analysis, including

exclusive-or and Abelian groups. Comon and Delaune [17] have identified a property known as the *finite variant property* which is checkable under appropriate conditions [23]. Although for many theories $E = \Delta \cup AC$ narrowing modulo AC does not terminate, The *folding variant narrowing strategy* [23] computes a finite complete set of E-unifiers whenever E has the finite variant property. Folding variant narrowing (currently implemented for strongly right-irreducible theories [41]) has proved to be the backbone of Maude-NPA. Although it is not as efficient as algorithms designed specifically for a given theory, it is more widely applicable, and it is easy to combine different equational theories [41].

3.2 Typed Modular Unification in Maude-NPA

Although the ease of implementation folding variant narrowing makes it very useful for exploration and experimentation, and interesting cryptographic theories satisfy the finite variant property, ultimately we also want to be able to make use of more efficient special-purpose algorithms. Moreover, there is a class of equational theories that appears prominently in cryptographic protocols applied to privacy-preserving computation: operators that are homomorphic with respect to another, e.g., $q(X * Y) = q(X) * q(Y)$. Theories like these can be shown to lack the finite variant property whether or not $*$ is a free operator or obeys the axioms for an Abelian group.[2] In these cases folding variant narrowing does not provide a finitary E-unification algorithm, and we must seek a different method.

As a result, special-purpose algorithms are also being developed, especially for homomorphic theories. But in order to do this it is necessary to do more than just develop and implement algorithms. They must also be integrated with analysis tools like the Maude-NPA, and we must satisfy ourselves that it is possible to use the tool to specify and analyze protocols that rely upon these properties.

Integrating equational unification into protocol analysis is challenging for several reasons. First of all, in principle we need to have a different $E_{\mathcal{P}}$-unification algorithm for each protocol $\mathcal{P}$; second, experience with the Maude-NPA tool has shown the great advantages (typically leading to a much smaller search space) of *typed unification*, where variables have types (or *sorts*) and types can be arranged in *subtype hierarchies*; for example, to properly specify a protocol we may wish to distinguish different subtypes —e.g., for nonces, keys, or principal names— of a general type for messages; third, we often need to *combine* several such unification algorithms, for example when composing together various subprotocols or taking into account the associative-commutative-identity (ACU) axioms of the state constructors (see Section 5). This is made even more challenging by the fact that, in order to allow the option of verifying different kinds of implementations (e.g. the case in which a key is indistinguishable from a nonce), typing is mostly left to the discretion of the user.

Given the wide range of protocols and protocol combinations that need to be analyzed, a *modular* approach to the development of $E_{\mathcal{P}}$-unification algorithms is very much needed. Such a modular approach and its necessary infrastructure are now under development. Besides using the known techniques for combining unification algorithms for disjoint theories à la Baader and Schultz [6], Maude-NPA employs a more general methodology and associated tool infrastructure (in the Maude-NPA) in which unification algorithms can be combined and developed at three different levels and in a not necessarily disjoint way: (i) a basic library of commonly occurring theories and their combinations —currently including any combination of typed commutative, associative commutative, associative commutative and identity, or free function symbols— is efficiently supported by the Maude tool at the C++ level; (ii) unification algorithms for special-purpose cryptographic theories can be developed in a declarative way in Maude itself using its metalevel facilities as done here for the homomorphic encryption theory E_h; and (iii) it is often possible to decompose an equational theory $E_{\mathcal{P}}$ as a disjoint union $E_{\mathcal{P}} = \Delta \cup Ax$, (where Δ and Ax may share some function symbols), and where a dedicated Ax-unification algorithm exists. If Δ is viewed as a set of rewrite rules that is convergent, coherent and has the finite variant property modulo Ax, *folding variant narrowing modulo* Ax with the rules Δ provides a finitary $E_{\mathcal{P}}$-unification algorithm [23]. Finally, in the modular approach proposed in this paper it is also possible to *automatically derive* a typed (called order-sorted) $E_{\mathcal{P}}$-unification algorithm from an untyped one for which an implementation is already available. This derivation follows the methodology proposed in [25] and applies a general method by which, under mild conditions on the order-sorted theory E, an order-sorted E-unification algorithm can be automatically obtained by: (i) associating to E its unsorted version $\bar{E}$; (ii) computing a complete set of (unsorted) $\bar{E}$-unifiers for the given E-unification problem; and (iii) typing and filtering out the unsorted $\bar{E}$-unifiers to obtain a complete set of order-sorted E-unifiers using the generic *sort propagation algorithm* described in [25]. This algorithm is part of the Maude-NPA infrastructure that is applied to the homomorphic encryption algorithm described in this paper.

Finally, we combine $E_{\mathcal{P}}$-unification,with a typed version of ACU-unification. The latter is needed because Maude-NPA states are multisets of terms, which are associative-commutative and have the empty multiset as tthe identity. This combination is supported by Maude-NPA by means of an order-sorted variant of the standard combination method for disjoint theories à la Baader and Schultz [6], so that in the end typed $E_h \cup ACU$-unification is achieved. A more complete description of how this is done is given in [41].

4. Protocol Examples

In this section we include several protocols that we will use as motivating examples that are subject to attacks which we demonstrate in Maude-NPA. Because homomorphic encryption is usually used together with an operator with additional algebraic properties, there are not very many examples of protocols that rely on homomorphic encryption over an operator with no further algebraic properties that are relevant to the protocol. Thus in some cases we have devised our own. In doing this, we have attempted to cover various situations that could arise when dealing with homomorphic encryption. The first protocol is a secure two-party computation protocol that illustrates the application of homomorphic encryption to multi-party computation. The second is a version of the Needham-Schroeder-Lowe protocol using encryption in Electronic Code Book (ECB) mode, due to Cortier et al. [19], which allows us to test our implementation on an independently developed protocol. The third is an ECB version of shared key Needham-Schoeder, which allows us to check how homomorphic encryption would behave under nested encryption. The final protocol uses a homomorphic "hash function", and was designed as an example of a protocol that could not be analyzed by the application of standard intruder inference rules. The last three test protocols are not intended to be realistic, since the unsafeness of using ECB mode when message integrity is required is well known, but they serve to test the limits of our implementation.

4.1 Multi-Party Computation with Semi-Trusted Third Party

This is a protocol in which two principals, A (for Alice) and B (for Bob)want to compute a function f of their private data X and Y

[2] Comon and Delaune only prove the result for the exclusive-or case in [17], but their proof can easily be extended to the other cases.

without revealing anything about X and Y other than $f(X, Y)$. They use a trusted server to compute $f(X, Y)$, but they don't want to reveal X and Y to the server either. They make use of a public key encryption algorithm $hpke$ which is homomorphic with respect to f, i.e., it satisfies the equation $hpke(f(X, Y), Z) = f(hpke(X, Z), hpke(Y, Z))$. We assume that A and B share the same public (and corresponding private) key $pkey(A, B)$ for the homomorphic public key encryption algorithm $hpke$, so that both can decrypt data encrypted by $pkey(A, B)$. The server s also possesses a public (and private) key for a conventional public key encryption algorithm; the encryption of message M by server's key is denoted by $pke(M, S)$. All principals have digital signature keys; the digital signature of message M by principal P is denoted by $sign(M, P)$. Finally, concatenation is denoted by ;.

1. $A \to B : sign(B; N_A;$
   $pke(hpke(D_A, pkey(A, B)), S), A)$

   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. $B \to A : sign(N_A; N_B;$
   $pke(hpke(D_B, pkey(A, B)), S), B)$

   B sends a similar message to A, including both his and A's nonce.

3. $A \to S : sign(A; B; N_A; N_B;$
   $pke(hpke(D_A, pkey(A, B)), S);$
   $pke(hpke(D_B, pkey(A, B)), S), A)$

   A sends a signed message containing both nonces and both encrypted data sets to S.

4. $S \to A, B : sign(A; B; N_A; N_B;$
   $f(hpke(D_A, pkey(A, B)),$
   $hpke(D_B, pkey(A, B))), S)$

   The server applies f to both encrypted data sets and sends the result to A and B.

This protocol is potentially vulnerable to an attack in which A can be led to believe that f has been applied to B's data when actually it has not. However, we can use the homomorphic property of the encryption to implement a check that prevents the attack. This attack is as follows.

1. $A \to I(B) : sign(B; N_A;$
   $pke(hpke(D_A, pkey(A, B)), S), A)$

   A initiates the protocol with B, but A's message is intercepted by I. We denote I impersonating B by $I(B)$.

2. $I \to B : sign(B; N_A; E, I)$

   I uses A's message 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. $B \to A : sign(N_A; N_B;$
   $pke(hpke(D_B, pkey(I, B)), S), B)$

   B believes that he is talking to I and sends the corresponding reply message. I forwards it to A.

4. $A \to S : sign(A; B; N_A; N_B;$
   $pke(hpke(D_A, pkey(A, B)), S);$
   $pke(hpke(D_B, pkey(I, B)), S), A)$

   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. $S \to A, B : sign(A; B; N_A; NB;$
   $f(hpke(D_A, pkey(A, B)),$
   $hpke(D_B, pkey(I, B)), S)$

   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 $hpke(f(X, Y), pkey(A, B))$ for some X and Y, and one in which she does not. We do this by specifying the format of the final message that A receives. If no check is made, A will accept anything the server sends her. In this version the final message she receives in her strand is written as $sign(A; B; n(A, r); N; X1, s)$ where $X1$ is a free variable. In the version in which she checks the format, we write $hpke(f(X, Y), pkey(A, B))$ instead of *X1*. Note that the homomorphic property of the encryption is specified in both versions, but is only used in the second version, to unify the message the server sent (in which f is applied to the encrypted data) with the message Alice received (in which the encryption function is applied to the result of computing f). Note also that this check, or the lack of it, is not easy to specify in the informal, journal level style, but is straightforward to specify in Maude-NPA, in which the message the server sends and the message Alice accepts are specified separately.

4.2 Homomorphic Needham-Schroeder-Lowe

In [19] Cortier et al. give the following example of the Needham-Schroeder-Lowe protocol using public key encryption implemented in Electronic Code Book Mode, so that data is

1. $A \to B : pke(N_A; A, B)$
2. $B \to A : pke(N_A; N_B; B, A)$
3. $A \to B : pke(N_B, B)$

There are a number of ways in which either A or B can be tricked into believing that they have successfully completed a run of the protocol with another, when in fact this has not happened. Here is one of the simplest:

1. $I_A \to B : pke(N_I; A, B)$
2. $B \to I_A : pke(N_I; N_B; B, A)$

   This message is intercepted by the intruder, who, thanks to the homomorphic property, is able to extract $pke(N_B, A)$. He uses this to initiate the protocol with A, posing as B:

3. $I_B \to A : pke(N_B; B, A)$
4. $A \to I_B : pke(N_A; N_B; A, B)$

   The intruder is now able to extract $pke(N_B, B)$ and use it to complete its impersonation of A to B.

5. $I_A \to B : pke(N_B, B)$.

4.3 Homomorphic Needham-Schroeder Shared Key

This is a version of the Needham-Schroeder shared key protocol, in which a principal A requests a session key for communicating with B from a server S. The server sends A the key, encrypted under a master key shared between A and S. The message containing that key also contains the same key encrypted under a master key shared between B and S. A then forwards the encrypted key to B, after which A and B perform a handshake.

Normally the Needham-Schroeder shared key protocol is specified using only a single encryption algorithm, but here we specify three: me used for the outer encryption, e used for the inner

encryption, and se used for the handshake. Only e is homomorphic over concatenation, i.e., it satisfies the equation $e(X; Y, Z) = e(X, Z); e(Y, Z)$:

1. $A \to S : A; B; N_A$
   A sends server S a request for a key to share with B.
2. $S \to A : me(N_A; B; K_{AB}; e(K_{AB}; A, K_{BS}), K_{AS})$
   S encrypts a session key K_{AB} and A's name with K_{BS} using the homomorphic e operator. It then encrypts that, along with K_{AB}, B, and N_A, with the key K_{AS} it shares with A using the me operator, and sends the result to A.
3. $A \to B : e(K_{AB}; A, K_{BS})$
   A removes the outer layer of encryption and sends the inner encrypted message to B.
4. $B \to A : se(N_B, K_{AB})$
5. $A \to B : se(s(N_B), K_{AB})$
   A and B agree that they share a key.

This protocol is vulnerable to an attack using two regular sessions in parallel, where the intruder gets $e(K_{CB}, K_{BS})$ from one regular execution between C and B and $e(A, K_{BS})$ from one regular execution between A and B.

1. $C \to S : C; B; N_C$
   C sends server S a request for a key to share with B.
2. $S \to C : me(N_C; B; K_{CB}; e(K_{CB}; C, K_{BS}), K_{CS})$
   S encrypts a session key K_{CB} and C's name with K_{BS} using the homomorphic operator e. It then encrypts that, along with K_{CB}, B, and N_C, with the key K_{CS} that it shares with C using the me operator, and sends the result to C.
3. $C \to I(B) : e(K_{CB}; C, K_{BS})$
   C removes the outer layer of encryption and sends the inner encrypted message to B. However, this is intercepted by the intruder I.
4. $I(A) \to B : e(K_{CB}; A, K_{BS})$
   Now the intruder I exploits the fact that $e(K_{CB}; C, K_{BS}) = e(K_{CB}, K_{BS}); e(C, K_{BS},)$ to obtain $e(K_{CB}, K_{BS})$. If A previously requested a key to talk to B, then the intruder could also have obtained $e(A, K_{BS})$ in the same way. He uses this to construct $e(K_{CB}, K_{BS}); e(A, K_{BS}) = e(K_{CB}; A, K_{BS})$, which he then sends to B.
5. $B \to I(A) : se(N_B, K_{CB})$
6. $I(B) \to C : se(N_B, K_{CB})$
   B responds according to the protocol, and I forwards his message to C.
7. $C \to B : se(s(N_B), K_{CB})$
   C responds according to the protocol. Now B will attribute any message from C to A.

4.4 Homomorphic Hash Protocol

In this protocol, A and B use a shared key to agree on a secret nonce N'_B. They use keyed hash functions to guarantee integrity of their messages, but let us suppose that they use a hash function h with a fatal flaw. Function h is homomorphic over concatenation, i.e., it satisfies the equation $h(X; Y, Z) = h(X, Z); h(Y, Z)$.

1. $A \to B : A; N_A$
   A starts by sending B her name and a nonce.
2. $B \to A : N_B; e(h(N_B; N'_B; N_A, K_{AB}), K_{AB});$
   $e(N'_B; N_A, K_{AB})$
   B responds with a nonce and two encrypted messages. Note that N'_B is only sent encrypted; it is intended to be a shared secret between A and B.
3. $A \to B : e(h(N_B, K_{AB}); h(N'_A, K_{AB}), K_{AB});$
   $e(N'_A, K_{AB})$
   A verifies to B that she received his message.

Now it turns out that an intruder can trick B into believing that he has completed a successful run of the protocol with A even though A is not present. This is because B can be fooled into accepting the message he sent in the second step of the protocol as the message he receives in the third step. We note that since h is homomorphic over concatenation, we have $h(N_B; N'_B; N_A, K_{AB}) = h(N_B, K_{AB}); h(N'_B; N_A, K_{AB})$. Thus we have the following attack.

1. $I(A) \to B : A; N_I$
   I starts by sending B A's name and a nonce.
2. $B \to I(A) : N_B; e(h(N_B; N'_B; N_I, K_{AB}), K_{AB});$
   $e(N'_B; N_I, K_{AB})$
   B responds to A according to the protocol. This is intercepted by I.
3. $I(A) \to B : e(h(N_B; N'_B; N_I, K_{AB}), K_{AB});$
   $e(N'_B; N_I, K_{AB})$
   I repeats B's message back to him, only leaving out the N_B at the beginning. If B has no way of telling the concatenation of two nonces from a nonce, he can mistake $N_B; N_I$ for a nonce N'_I. Thus he will mistake $e(N'_B; N_I, K_{AB})$ for $e(N'_A, K_{AB})$, and, because of the homomorphic properties of the hash function, he will mistake $e(h(N_B; N'_B; N_I, K_{AB}), K_{AB})$ for $e(h(N_B, K_{AB}); h(N'_A, K_{AB}), K_{AB})$.

5. Search in Maude-NPA

In this section we give a high-level summary of the general approach advocated in this paper for formally analyzing protocols *modulo* their algebraic properties, with particular attention to the way this approach is implemented in Maude-NPA. For further information, please see [21, 22].

Given a protocol $\mathcal{P}$, states are modeled as elements of an initial algebra $T_{\Sigma_\mathcal{P}/E_\mathcal{P}}$, where $\Sigma_\mathcal{P}$ is the signature defining the sorts and function symbols (for the cryptographic functions and for all the state constructor symbols) and $E_\mathcal{P}$ is a set of equations specifying the *algebraic properties* of the cryptographic functions and the state constructors. Therefore, a state is an $E_\mathcal{P}$-equivalence class $[t] \in T_{\Sigma_\mathcal{P}/E_\mathcal{P}}$ with t a ground $\Sigma_\mathcal{P}$-term. However, since the number of states $T_{\Sigma_\mathcal{P}/E_\mathcal{P}}$ is in general infinite, rather than exploring concrete protocol states $[t] \in T_{\Sigma_\mathcal{P}/E_\mathcal{P}}$ we explore *symbolic state patterns* $[t(x_1, \ldots, x_n)] \in T_{\Sigma_\mathcal{P}/E_\mathcal{P}}(X)$ on the free $(\Sigma_\mathcal{P}, E_\mathcal{P})$-algebra over a set of typed variables X. In this way, a state pattern $[t(x_1, \ldots, x_n)]$ represents not a single concrete state but a possibly infinite set of such states, namely all the instances of the pattern $[t(x_1, \ldots, x_n)]$ where the variables $x_1, \ldots, x_n$ have been instantiated by concrete ground terms.

In Maude-NPA [21, 22], a *state* in the protocol execution is a term t of sort *state*, $t \in T_{\Sigma_\mathcal{P}/E_\mathcal{P}}(X)_{state}$, which is a multiset. Each element in the multiset can be a strand or the intruder knowledge at that state. A *strand* [24] represents the sequence of messages sent and received by a principal executing the protocol and is indicated by a sequence of messages $[msg_1^-, msg_2^+, msg_3^-, \ldots, msg_{k-1}^-, msg_k^+]$ such that $msg_i \in T_{\Sigma_\mathcal{P}/E_\mathcal{P}}(X)_{\mathsf{Msg}}$, msg^- represents an input message, and msg^+

represents an output message. 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 (with a strand specified for each intruder operation). In Maude-NPA, strands evolve over time; the symbol | is used to divide past and future: $[msg_1^{\pm}, \ldots, msg_{j-1}^{\pm} \mid msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}]$ where $msg_1^{\pm}, \ldots, msg_{j-1}^{\pm}$ are the past messages, and $msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}$ are the future messages ($msg_j^{\pm}$ is the immediate future message). The *intruder knowledge* is represented as a multiset of facts. 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.

We illustrate the approach using the homomorphic Needham-Schroeder shared key protocol from Section 4.3. The strands associated to the five protocol steps above are given next. There are three strands, one for each principal in the protocol. Constants and function symbols are represented by small letters, and variables by capital letters, with the exception of variables of sort *Fresh* which are special variables that can't be unified with each other once they appear in a state. and are used for nonce generation. These are denoted by small letters and declared within the delimiter :: :: at the beginning of a strand. Sent messages are prefixed with a $+$, and received messages with a $-$. Note that sent messages from one strand do not always exactly with their corresponding receive messages. For example, the first message $A \to S : A; B; N_A$ is represented by a message in Alice's strand sending $(A; B; n(A, r))^+$ and another message in the Server's strand receiving $(A; B; N)^-$. When a principal cannot observe the contents of a concrete part of a received message (e.g., because a key is necessary to look inside), a generic variable is used for such part of the message in the strand (as with variable N of sort *Nonce* above). We encourage the reader to compare the protocol in strand notation to the presentation of the protocol in Section 4.3. Note that we first name the principal, then show the special variable(s) used for nonce generation (framed within ::), see below, and then the actual strand.

```
(A) :: r ::
    [ +(A ; B ; n(A,r)),
      -(me(n(A,r) ; B ; SK ; X , mkey(S, A))),
      +(X), -(se(M, SK)), +(se(succ(M), SK)) ]
(B) :: r' ::
    [ -(e(SK ; A, mkey(S,B))), +(se(n(B,r'), SK)),
      -(se(succ(n(B,r')), SK)) ]
(S) :: r'' ::
    [ -(A ; B ; N),
      +(me(N ; B ; skey(S,r'') ;
               e(skey(S,r'') ; A, mkey(S,B)),
          mkey(S,A))) ]
```

Intruder strands are also included for each function. For example, concatenation by the intruder is described by the strand $[(X)^-, (Y)^-, (X; Y)^+]$.

The protocol analysis methodology of Maude-NPA is then based on the idea of *backward reachability analysis*, where we begin with one or more state patterns corresponding to *attack states*, and want to prove or disprove that they are *unreachable* from the set of initial protocol states. In order to perform such a reachability analysis we must describe how states change as a consequence of principals performing protocol steps and of the intruder actions. This can be done by describing such state changes by means of a set $R_{\mathcal{P}}$ of *rewrite rules*, so that the rewrite theory $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{\mathcal{P}})$ characterizes the behavior of protocol $\mathcal{P}$ modulo the equations $E_{\mathcal{P}}$; see [21, 22] for the concrete rewrite rules.

The way to analyze *backwards* reachability is then relatively easy, namely to run the protocol "in reverse." This can be achieved by using the set of rules $R_{\mathcal{P}}^{-1}$, where $v \longrightarrow u$ is in $R_{\mathcal{P}}^{-1}$ iff $u \longrightarrow v$ is in $R_{\mathcal{P}}$. Reachability analysis can be performed *symbolically*, not on concrete states but on symbolic state patterns $[t(x_1, \ldots, x_n)]$ by means of *narrowing* [26, 36], where at each step of rewriting instead of *matching* a subterm t' of a concrete state t with a left-hand side v we *unify* v and the state pattern $t(x_1, \ldots, x_n)$. However, since our state patterns are not just syntactic terms $t(x_1, \ldots, x_n)$ but rather $E_{\mathcal{P}}$-equivalence classes $[t(x_1, \ldots, x_n)]$ we cannot just perform syntactic unification but instead should perform *semantic unification* modulo $E_{\mathcal{P}}$. In other words, we should perform not just syntactic narrowing for our backwards reachability analysis with $R_{\mathcal{P}}^{-1}$, but *narrowing modulo* $E_{\mathcal{P}}$ [27, 36]. Note that this is a different application of narrowing than the previous use for unification in Section 3.

$E_{\mathcal{P}}$-unification precisely models all the different ways in which an intruder could exploit the algebraic properties $E_{\mathcal{P}}$ of $\mathcal{P}$ to break the protocol; therefore, if an initial state can be shown unreachable by backwards reachability analysis modulo $E_{\mathcal{P}}$ from an attack state pattern, this ensures that, even if the intruder uses the algebraic properties $E_{\mathcal{P}}$, the attack cannot be mounted. This means that efficient support for $E_{\mathcal{P}}$-unification is a crucial feature of symbolic reachability analysis of protocols modulo their algebraic properties $E_{\mathcal{P}}$.

6. Unification modulo E_h

In this section we outline an algorithm for unification modulo the homomorphic encryption theory E_h defined by the single oriented equation $e(X; Y, Z) \to e(X, Z); e(Y, Z)$ in a signature containing symbols e, $\_;\_$, and uninterpreted function symbols. Here we only give a high-level description of the algorithm—some of the details are omitted and can be found in [2]. Since E_h can be viewed as a one-sided distributivity rule, the inference system given here can be compared to the one in [43]. We believe the algorithm we use is simpler and easier to implement, as it does not involve any cycle checking.

Over the empty theory, two terms with different top-level function symbols do not unify; but, modulo E_h, a concatenation may unify with an encryption. We extend the standard algorithm for syntactic unification by introducing additional inference rules called **Shaping**, **Parsing** and **Failure**.

Given the following E_h-unification problem: $X; Y = e(Z, k)$, the only way this can be solved is if X and Y are both encryptions with key k, so we instantiate X with $e(X', k)$ and instantiate Y with $e(Y', k)$. This idea is generalized into an inference rule called **Shaping**. Once the **Shaping** rule has been applied, everything is encrypted with key k, and then we need to remove key k and deduce that $Z = X'; Y'$. This can be generalized into an inference rule called **Parsing**.

For example, consider the E_h unification problem $X; e(Y, k_2) = e(e(Z, k_1), k_2)$. Here Z has been encrypted by k_1 followed by k_2. The pieces of the concatenation must also be encrypted by the same sequence of keys. So two applications of the **Shaping** rule will instantiate X by $e(e(X', k_1), k_2)$, and Y by $e(Y', k_1)$. The result is $e(e(X', k_1), k_2); e(e(Y', k_1), k_2) = e(e(Z, k_1), k_2)$. One application of **Parsing** removes key k_2 everywhere, resulting in $e(X', k_1); e(Y', k_1) = e(Z, k_1)$. A second application removes k_1, resulting in $Z = X'; Y'$.

To summarize, when a concatenation s is equal to an encryption t, each element of s must be a term encrypted with the same sequence of keys as t is encrypted with. The purpose of the **Shaping** rule is to guarantee that everything is encrypted by the same sequence of keys. Once everything has been encrypted by the same sequence of keys, the **Parsing** rule can be applied to remove the outermost key from each key sequence. Several applications of **Parsing** will remove all the keys.

At certain times, it can be detected that **Shaping** cannot make those key sequences to be the same. There are two rules to handle these cases and detect failure. For example, suppose we have $X; c = e(Z, k)$ or $X; e(c, k_2) = e(e(Z, k_1), k_2)$, where c is a constant. In both of these cases, the constant c cannot be instantiated, so it is impossible to make the sequence of keys the same everywhere, so we fail. This is generalized as the first **Failure** rule. A second example of failure is $X; Y = e(X, k)$ or $e(X, k_2); Y = e(e(X, k_1), k_2)$. Again here we cannot make the key sequence to be the same, because any instantiation of X on the left-hand side of the equation will also require the instantiation of X on the right hand side of the equation, so we fail. This is generalized as the second **Failure** rule.

The E_h-*Unification* procedure is defined by a don't-care nondeterministic application of the inference rules. The terms in the equations are rewritten so they are kept in E_h-normal form, thus putting concatenations over encryptions.

In a set of equality constraints, a variable x is said to be *solved* iff x appears only once and as one side of an equation. A *solved form* for E_h-Unification is a set of E_h-equalities $\{x_1 = t_1, \cdots, x_n = t_n\}$, where each x_i is a *solved variable*. The unification algorithm produces a solved form if the unification problem is solvable modulo E_h; else it returns $Fail$.

The inference rules in this section have been proved sound, complete, and terminating in [2] meaning that all solved forms created by the algorithm are correct solutions of the unification problem and that for every solution of the unification problem there is a more They also have been implemented in Full Maude and integrated into Maude-NPA, using the methods described in Section 3.2.

7. Finding attacks modulo $E_h \cup ACU$ using Maude-NPA

7.1 Multiparty Computation Protocol

We define an attack state for Alice in which the Alice strand completes, but there is no corresponding Bob strand using the same data. We do this by including Alices's strand in the final goal and putting in Bob's strand as a "never pattern" [22]. A never pattern is a pattern denoting a state with partial strand information or positive intruder knowledge that can never happen within the path from an initial state to the given attack pattern. These never patterns are checked by matching modulo $E_h \cup ACU$, i.e., if any state in a backwards reachability path matches modulo $E_h \cup ACU$ with a never pattern, then the path is discarded. Note that we do not include Bob's final received message, since it can always be blocked by the intruder. The attack pattern is as follows, written in Maude-NPA syntax [22]:

```
:: r, r' ::
[ nil, +(sign( b ; n(a,r) ;
               pke(hpke(data(a,r'),pkey(a,b)),s),
              a)),
       -(sign( n(a,r) ; N ; E ,b )),
       +(sign( a ; b ; n(a,r) ; N ;
               pke(hpke (data(a,r'),pkey(a,b)),s) ; X,
              a)),
       -(sign (a ; b ; n(a,r) ; N ; Z , s )) | nil ]
|| empty || nil || nil
|| never(*** Never Pattern for authentication
     :: r1, r2  ::
     [ nil | -(sign( b ; n(a,r) ;
                     pke(hpke(data(a,r'), pkey(a,b)),s),
                    a)),
             +(sign( n(a,r) ; N ; E ,b )),
             nil ]
     & SS:StrandSet || IK:IntruderKnowledge )
```

This pattern produces the attack we describe in Section 4.1 in ten steps, most of which involve the intruder removing the nonce from A's original message and inserting it into his own message. In our original specification of the protocol, in which encrypted messages were not typed, the tool failed to terminate, generating states in which the encrypted message field was replaced by an ever larger number of concatenated messages. Although Maude-NPA has inductive methods for avoiding such infinite sequences, they do not always work, which, given the undecidability of unbounded session protocol analysis, is not surprising. Instead, when we specified sorts for the different types of encrypted messages, the tool terminated in twelve steps (i.e., the search space was finite after twelve backwards narrowing steps, while the attack was found at step number ten). When we specified Alice's format check, Maude-NPA terminated at the fourth step without finding an attack, for the sorted version of the protocol, verifying it secure.

We also specified a corresponding attack pattern for B, in which a B strand completes without a corresponding A strand. In this case, Maude-NPA terminated in four steps without finding an attack, with and without the final check.

We note that the use of the homomorphic equational theory makes it very straightforward to specify the final check. To do something similar in the free theory we would have to either add extra deductions by A and B, or simply assume that A and B could recognize data of the form $f(hpke(X, Y))$ without specifying the reason why. Either way adds to the complexity of the analysis and detracts from the intuitive understanding of the protocol.

7.2 Homomorphic Needham-Schroeder-Lowe

For the Needham-Schroeder-Lowe protocol, we defined four attack states. In the two authentication patterns, Bob (resp. Alice) completes, apparently with the other party, but Alice (resp. Bob) does not complete with the same data. These are similar to the state defined in Section 7.1, so are omitted. In the other two , Bob (resp. Alice) completes, apparently with the other party, but the intruder learns Bob's (resp. Alice's) nonce. The secrecy pattern for Bob is as follows:

```
:: r ::
   [ nil, -(pk(a ; NA, b)), +(pk(NA ; n(b,r) ; b,a)),
          -(pk(n(b,r), b)) | nil ]
   || n(b,r) inI, empty || nil || nil || nil
```

and the pattern for Alice is similar. For the Bob authentication pattern, Maude-NPA terminated after ten steps and found three attacks (including the one described in Section 4.2). For the Bob secrecy pattern specified above, Maude-NPA terminated after thirteen steps and found three attacks. For the Alice secrecy and authentication patterns, respectively, Maude-NPA terminated after ten steps and found two attacks, and terminated after eight steps and found four attacks.

7.3 Homomorphic Needham-Schroeder

One of the purposes of this protocol was to give a "stress test" to Maude-NPA and thus at the beginning we used only one encryption algorithm for the entire protocol, which was homomorphic with respect to concatenation. We used an attack state in which Bob completes an instance of the protocol as a responder, apparently with Alice as initiator but Alice does not complete the corresponding instance of the protocol with B. This caused an enormous state explosion, and the tool did not complete or find an attack. We tried again using three cryptosystems, only one homormorphic. We again suffered from a state space explosion, but we noticed that the tool was spending much of its effort in searching for the term $\mathtt{e}(\mathtt{A}, \mathtt{mkey}(\mathtt{b}, \mathtt{s}))$. In order to eliminate this, we changed the speci-

fication to say that the intruder knows this term initially, thus eliminating the search for it. Since the production of $\mathtt{e}(\mathtt{A},\mathtt{mkey}(\mathtt{b},\mathtt{s}))$ was the only place in which Alice's initiation of an instance of the protocol with B was needed, we could now specify a much weaker security property, in which Alice never initiates any instance of the protocol with Bob at all. This requires us to rule out not only any complete Alice strand initiated with Bob, but any aborted strands as well. In the Maude-NPA model any strand that is enabled for a send will send, so strands can only abort when they are enabled for a receive. There are two places where Alice's strand is enabled for a receive: after she has sent the initiation message, and after she has forwarded the session key to Bob. This gives us three never patterns: one for Alice's full strand, and two for the two partial strands. Our attack pattern is thus as follows:

```
:: r ::
[ nil, -(e(SKEY ; a , mkey(b,s))),+(se(n(b,r), SKEY )),
       -(se(succ(n(b,r)), SKEY)) | nil ]
|| empty || nil  || nil
|| never( *** Never Pattern for authentication
  (:: r' :: [ nil | +(a ; b ; n(a',r)),
                    -(Z), +(X), -(Y), +(W), nil ]
          & S:StrandSet || K:IntruderKnowledge )
          *** Never Pattern for authentication
  (:: r ':: [ nil | +(a ; b ; n(a,r')), -(Z), +(X), nil ]
          & S:StrandSet || K:IntruderKnowledge )
          *** Never Pattern for authentication
  (:: r' :: [ nil | +(a ; b ; n(a,r')), nil ]
          & S:StrandSet || K:IntruderKnowledge ) )
```

Maude-NPA found the attack in seven steps. However, Maude-NPA does not terminate immediately upon finding an attack, but continues to search until it has exhausted the search space. In this case, although it did find the attack, the complete search still did not terminate. Interestingly, we were not able to find evidence that Maude-NPA was generating any infinite paths in its search. Rather, the use of homomorphic encryption gave the intruder so many opportunities for generating different types of states that it overwhelmed the tool. Thus, this protocol did turn out to provide an excellent opportunity for stress testing, and likely will be useful for benchmarking in future work.

7.4 Homomorphic Hash Protocol

This protocol uses a keyed hash algorithm that is homomorphic over concatenation. This protocol was designed in order to demonstrate how Maude-NPA can reason about situations in which standard augmented intruder inference rules would not be applicable. In this protocol the homomorphic hash is hidden under a non-homomorphic encryption, so applying augmented inference rules —as explained in Section 2— that say, for example, that the intruder learns $h(X,K);h(Y;K)$ if he knows $h(X;Y,K)$ would not apply. However, the intruder can still use this property to fool honest principals, even if he can't apply it to learn anything new himself.

For this protocol we showed that a very weak security property fails to hold: Bob can execute an instance of the protocol as responder without any initiator strand executing, even a partial one. Only initiator strands are of the form `:: r1 , r2 :: [ nil | +(X) , ... ]` , so we specify the attack state as follows:

```
:: r,  r' ::
[ nil, -(a ; NA),
       +(n(b,r) ; e(h( n(b,r) ; n(b,r') ; NA, mkey(a,b)),
                    mkey(a,b))
                ; e(n(b,r') ; NA, mkey(a,b))),
       -( e( h(n(b,r), mkey(a,b)) ;
             h(NA', mkey(a,b)), mkey(A,B))
        ; e(NA', mkey(a,b)))  | nil ]
|| empty || nil || nil
|| never( *** Never Pattern for authentication
   (:: r'', r''' :: [ nil | +(X), -( Y), +(W), nil ]
   & S:StrandSet || K:IntruderKnowledge )
          *** Never Pattern for authentication
   (:: r'', r''' ::  [ nil | +(X), nil]
   & S:StrandSet || K:IntruderKnowledge))
```

This produces the attack described in Section 4.4. In this case the tool found the attack in four steps without our needing to add never patterns. However, after finding the attack, the tool demonstrated the same type of state explosion as the homomorphic Needham-Schroeder protocol. This happened as the result of several paths in which the intruder engaged in the protocol as a legitimate user of the system. The intruder thus was able to decrypt the hash and use its homomorphic property to break the hashed message into its components, resulting in a large number of states in which these parts were mixed and matched in different ways.

7.5 Discussion

As we can see, there was considerable variation in Maude-NPA's performance. It behaved well on the multi-party computation protocol, and on Cortier et al.'s ECB Needham-Schroeder-Lowe protocol. However, it suffered from combinatorial state explosion problems on the two other ECB protocols, even though their complexity was about the same as the multi-party computation protocol.

This, however, should not be a surprise. In general, extremely insecure protocols should be expected to give rise to a larger state space than more carefully designed ones, no matter what search strategy is used. There will be few possible paths through a sound protocol, while an unsound one will generate multiple paths. Thus even in the simplest ECB case, ECB Needham-Schroeder-Lowe, the tool found multiple attacks for each attack pattern.

However, even though the examples we encountered here were somewhat pathological, it can still be useful to have methods for dealing with them. Similar problems can arise when reasoning about protocols that employ long strings of concatenated messages, since the intruder can try different ways of combining different components. One possible approach to dealing with this is to employ some of the techniques that the NRL Protocol analyzer (NPA) [35] used to reduce state space size. One was the use of grammars, which are used both by the NPA and Maude-NPA to prove that certain terms could not be learned by the intruder unless they were cases of a list of excepted terms. When the NPA encountered such a term, it would unify the term with the exceptions, creating a new state for each exception [34]. Maude-NPA merely checks whether or not the condition holds [22]. This results in a cleaner model, but could contribute to increasing the size of the search space, since the tool deals with more general terms when more specific ones are available. Another feature that the NPA offered was the ability to generate lemmas about the unreachability of patterns of states encountered in a search. The identification of such patterns was manual and extremely tedious. But in Maude-NPA we have automated many procedures that were done manually in the NPA, and so these patterns may be a good topic to investigate. A third is the ability to not search for terms that can be easily be shown to be findable by the intruder, as we did for the Needham-Schroeder shared key example. This again could be specified manually in NPA, but again we can explore automated techniques to recognize, for example, terms that could be generated in an honest execution of the protocol with which the attacker does not interact.

8. Conclusions and Future Work

In this paper we described the first steps in dealing with unification-based cryptographic protocol analysis using homomorphic encryp-

tion. We developed a Maude implementation of a unification algorithm for homormorphic encryption over a free operator, and integrated it into Maude-NPA using a modular framework. Finally, we demonstrated the modular approach by applying Maude-NPA to examples and analyzed the results.

We are continuing to develop a library of dedicated unification algorithms for various cryptographic theories. Indeed, progress in this area has already been made; an efficient unification algorithm for unification modulo exclusive-or has been developed and implemented in Maude, and is currently being integrated with Maude-NPA. It is also being extended to an Abelian group unification algorithm. We plan to compare these with the folding variant narrowing approach already available in Maude-NPA.

Of particular interest, of course, is the extension of our approach to unification modulo more sophisticated homomorphic encryption theories, specifically encryption homomorphic over an Abelian group, which will allow us to apply our work to a number of realistic applications. Unfortunately the most straightforward way of achieving this in our model, via folding variant narrowing over encryption homomorphic over an AC operator, is not available to us, because unification modulo encryption homomorphic over an AC theory is known to be undecidable [38]. However, unification modulo encryption homomorphic over an Abelian group *is* decidable, giving us several ways to proceed. We could either determine if we could restrict ourselves to decidable subcases via judicious use of sorts, or we could incorporate existing unification algorithms applicable to encryption homomorphic over Abelian groups, e.g. the algorithm of Kapur et al. [28] into Maude-NPA. We are investigating these options.

References

[1] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theor. Comput. Sci.*, 367(1-2):2–32, 2006.

[2] S. Anantharaman, H. Lin, C. Lynch, P. Narendran, and M. Rusinowitch. Cap unification: application to protocol security modulo homomorphic encryption. In *ASIACCS*, pages 192–203. ACM, 2010.

[3] S. Anantharaman, P. Narendran, and M. Rusinowitch. Intruders with caps. In *Proc. RTA 2007*, volume 4533 of *LNCS*, pages 20–35. Springer, 2007.

[4] A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *CAV*, pages 281–285, 2005.

[5] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *ACM Conference on Computer and Communications Security*, pages 17–26, 1998.

[6] F. Baader and K. U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. In *CADE*, volume 607 of *LNCS*, pages 50–65. Springer, 1992.

[7] D. Basin, S. Mödersheim, and L. Viganò. An on-the-fly model-checker for security protocol analysis. In *In Proceedings of Esorics'03, LNCS 2808*, pages 253–270. Springer-Verlag, 2003.

[8] M. Baudet, V. Cortier, and S. Delaune. YAPA: A generic tool for computing intruder knowledge. In *Proc. RTA'09*, volume 5595 of *LNCS*, pages 148–163, Brasília, Brazil, June-July 2009. Springer.

[9] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW*, pages 82–96. IEEE Computer Society, 2001.

[10] Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Tree automata for detecting attacks on protocols with algebraic cryptographic primitives. *Electr. Notes Theor. Comput. Sci.*, 239:57–72, 2009.

[11] J. Bull. The authentication protocol. *APM Report*, 1997.

[12] S. Bursuc and H. Comon-Lundh. Protocol security and algebraic properties: Decision results for a bounded number of sessions. In *RTA*, pages 133–147, 2009.

[13] Y. Chevalier, D. Lugiez, and M. Rusinowitch. Verifying cryptographic protocols with subterms constraints. In *LPAR*, LNCS vol. 4790, pages 181–195. Springer, 2007.

[14] Y. Chevalier and M. Rusinowitch. Symbolic protocol analysis in the union of disjoint intruder theories: Combining decision procedures. *Theor. Comput. Sci.*, 411(10):1261–1282, 2010.

[15] Ş. Ciobâcă, S. Delaune, and S. Kremer. Computing knowledge in security protocols under convergent equational theories. In *Proc. CADE'09*, LNAI, pages 355–370, Montreal, Canada, 2009. Springer.

[16] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. *All About Maude - A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[17] H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In *RTA*, pages 294–307, 2005.

[18] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *LICS*, pages 271–. IEEE Computer Society, 2003.

[19] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

[20] C. J. F. Cremers. The Scyther tool: Verification, falsification, and analysis of security protocols. In *CAV*, pages 414–418, 2008.

[21] 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*, 367(1-2):162–202, 2006.

[22] 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*, LNCS vol. 5705, pages 1–50. Springer, 2009.

[23] S. Escobar, R. Sasse, and J. Meseguer. Folding variant narrowing and optimal variant termination. *The Journal of Logic and Algebraic Programming*, 2010. In Press. Available at `http://www.dsic.upv.es/~sescobar/papers.html`.

[24] F. J. T. Fabrega, J. Herzog, and J. Guttman. Strand Spaces: What Makes a Security Protocol Correct? *Journal of Computer Security*, 7:191–230, 1999.

[25] J. Hendrix and J. Meseguer. Order-sorted equational unification revisited. *Electr. Notes Theor. Comput. Sci.*, 2008. To appear in Proc. of RULE 2008.

[26] J.-M. Hullot. Canonical forms and unification. In *CADE*, LNCS vol. 87, pages 318–334. Springer, 1980.

[27] J.-P. Jouannaud, C. Kirchner, and H. Kirchner. Incremental construction of unification algorithms in equational theories. In *Proc. ICALP*, volume 154 of *LNCS*, pages 361–373. Springer, 1983.

[28] D. Kapur, P. Narendran, and L. Wang. A unification algorithm for analysis of protocols with blinded signatures. In *Mechanizing Mathematical Reasoning*, pages 433–451. Springer, 2005.

[29] S. Kremer and M. D. Ryan. Analysing the vulnerability of protocols to produce known-pair and chosen-text attacks. In *Proceedings of SecCo'04*, ENTCS, pages 84–107, London, UK, May 2005. Elsevier Science Publishers.

[30] R. Küsters and T. Truderung. Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In *CSF*, pages 157–171. IEEE Computer Society, 2009.

[31] 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*, 2010. To appear.

[32] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS*, pages 147–166, 1996.

[33] C. Lynch and C. Meadows. On the relative soundness of the free algebra model for public key encryption. *Electr. Notes Theor. Comput. Sci.*, 125(1):43–54, 2005.

[34] C. Meadows. Language generation and verification in the nrl protocol analyzer. In *CSFW*, pages 48–61. IEEE Computer Society, 1996.

[35] C. Meadows. The NRL protocol analyzer: An overview. *J. Log. Program.*, 26(2):113–131, 1996.

[36] J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. *Higher-Order and Symbolic Computation*, 20(1–2):123–160, 2007.

[37] J. K. Millen. On the freedom of decryption. *Inf. Process. Lett.*, 86(6):329–333, 2003.

[38] P. Narendran. Solving linear equations over polynomial semirings. In *LICS*, pages 466–472, 1996.

[39] O. Pereira and J.-J. Quisquater. On the impossibility of building secure cliques-type authenticated group key agreement protocols. *Journal of Computer Security*, 14(2):197–246, 2006.

[40] P. Y. A. Ryan and S. A. Schneider. An attack on a recursive authentication protocol. A cautionary tale. *Inf. Process. Lett.*, 65(1):7–10, 1998.

[41] R. Sasse, S. Escobar, J. Meseguer, and C. Meadows. Protocol analysis modulo combination of theories: A case study in Maude-NPA. In *Proc. STM 2010*. Springer, 2010.

[42] M. Schmidt-Schauß. Unification in a combination of arbitrary disjoint equational theories. *J. Symb. Comput.*, 8(1/2):51–99, 1989.

[43] E. Tidén and S. Arnborg. Unification problems with one-sided distributivity. *J. Symb. Comput.*, 3(1/2):183–202, 1987.

[44] M. Turuani. The CL-Atse protocol analyser. In *RTA*, pages 277–286, 2006.