Diffie-Hellman Cryptographic Reasoning in the Maude-NRL Protocol Analyzer

Santiago Escobar[1], Joe Hendrix[2], Catherine Meadows[3], and José Meseguer[4]

[1] Universidad Politécnica de Valencia, Spain. sescobar@dsic.upv.es
[2] University of Illinois at Urbana-Champaign, USA. jhendrix@cs.uiuc.edu
[3] Naval Research Laboratory, Washington, DC, USA. meadows@itd.nrl.navy.mil
[4] University of Illinois at Urbana-Champaign, USA. meseguer@cs.uiuc.edu

Abstract. The Maude-NRL Protocol Analyzer (Maude-NPA) is a tool and inference system for reasoning about the security of cryptographic protocols in which the cryptosystems satisfy different equational properties. It both extends and provides a formal framework for the original NRL Protocol Analyzer, which limited itself to an equational theory Δ of convergent rewrite rules. In this paper we extend our framework to include theories of the form $\Delta \uplus B$, where B is the theory of associativity and commutativity and Δ is convergent modulo B. Order-sorted B-unification plays a crucial role; to obtain this functionality we describe a sort propagation algorithm that filters out unsorted B-unifiers provided by the CiME unification tool. We show how extensions of some of the state reduction techniques of the original NRL Protocol Analyzer can be applied in this context. We illustrate the ideas and capabilities of the Maude-NPA with an example involving the Diffie-Hellman key agreement protocol.

1 Introduction

The Maude-NPA is a tool and inference system for reasoning about the security of cryptographic protocols in which the cryptosystems satisfy different equational properties. It is the next generation of the NRL Protocol Analyzer [21], a tool that supported limited equational reasoning and was successfully applied to the analysis of many different protocols. In Maude-NPA we improved on the original NPA in two ways. First of all, in [15] we formalized the inference system of NPA, providing the first formal description of the tool, in terms of rewriting logic and narrowing. We also provided proofs of soundness and completeness. More recently, we have been extending the equational reasoning capabilities of the tool. In [15], we considered equational theories for cryptographic primitives and other functions given by a confluent and terminating set of equations Δ, as did the original NPA. In this work, we extend the framework to theories of the form $\Delta \uplus B$, where B is the theory of associativity and commutativity (AC) and Δ is confluent and terminating modulo B. We illustrate the ideas and capabilities of this extension of the Maude-NPA with an example using the Diffie-Hellman protocol, involving exponentiation and associative-commutative multiplication.

Maude-NPA uses backwards search from an insecure state to find attacks or to prove unreachability. This is implemented using backwards narrowing with the protocol rules *modulo* $\Delta \uplus B$. There are two ways to do this. One is to use built-in unification

algorithms for each theory and combination of theories. The other is to use a hybrid approach, for example to use built-in algorithms for B, and a generic algorithm, such as narrowing modulo B, for Δ. We choose the second approach, as being more readily extensible to different theories. That is, we use narrowing at two levels: for backwards search from an insecure state (i.e., $\Delta \uplus B$-narrowing with the rewrite rules describing the protocol) and for the $\Delta \uplus B$-unification used in each backwards narrowing step (i.e., B-narrowing with the rewrite rules obtained by orienting Δ).

We have found important technical advantages in using *order-sorted* theories. In order-sorted theories, narrowing will terminate, providing a finitary unification algorithm, in many cases in which unsorted narrowing will not. Furthermore, even in the case in which both terminate, order-sorted narrowing will often produce a smaller search space. Two interesting examples of this use of order-sorted theories to obtain finitary unification algorithms are our approximate theory for associativity in [14], and the theory $\Delta \uplus B$ for Diffie-Hellman exponentiation in this paper. In both cases, narrowing with the corresponding unsorted theories is non-terminating, whereas narrowing with the order-sorted theories does terminate. The current support of order-sorted unification in the Maude-NPA leverages CiME's [11] rich library of composable unsorted unification algorithms, including any combination of associativity, commutativity and identity axioms (except associativity without commutativity). The Maude-NPA then filters out with order-sorted information the unsorted unifiers provided by CiME, using the sound and complete sort propagation algorithm presented in Section 4 to obtain the corresponding order-sorted unifiers.

1.1 Related work

Recently, the modeling and formal analysis of cryptographic protocols in which the cryptographic and other functions used obey different equational theories has been an active research topic. Much of the work in this area has concentrated on problems of secrecy and static equivalence in bounded session protocols, which have been proved to be decidable for an important class of equational theories [23,7,1,9,6,12]. For unbounded sessions, however, the problem is less well understood and has been recently studied in [5,4]. In [17] and [7] tree-automata based approximations are applied to associative-commutative theories to develop abstract approximations. These techniques have been implemented in tools (for Diffie-Hellman theories in [17], and for exclusive-or in [7]) that guarantee protocol security if they find no attacks, but may find spurious attacks even if the protocol is sound. This is in contrast to the aim of our work, which is to provide both genuine proofs of security and genuine attacks, while using heuristic techniques, such as grammars, to make termination more likely even if it is not guaranteed.

Our work, which relies on backwards narrowing, requires a sound and complete unification algorithm for associative-commutative theories such as exponentiation and exclusive-or. Probably the most closely related work to ours in this area is the unification algorithms for exponentiation in Narendan and Meadows [20] and Kapur, Narendran, and Wang [19]. The theory we use in this paper is an order-sorted version of a fragment of the theories for which unification algorithms are developed there. In this work, however, we use a hybrid approach that we believe is more readily extensible. Instead of developing special-purpose algorithms for individual theories, we

make use of an order-sorted unification algorithm for associative-commutative theories, obtained in this case by combining the AC algorithm from the CiME tool with our sort propagation algorithm. This is then combined with rewrite theories that are terminating and confluent with respect to the AC theory. Some other work that is closely related to ours is the work of Comon-Lundh and Delaune [10] on the finite variant property, in which techniques are developed for achieving termination even when narrowing by itself does not terminate. Although this paper does not make use of their results, we expect to find them helpful to our future work.

1.2 A Diffie-Hellman Example

In order to demonstrate the use of associativity and commutivity (AC) in the Maude-NPA, we provide an example involving the well-known Diffie-Hellman key agreement protocol. This protocol uses exponentiation in order to generate a shared secret between two parties, and is the basis for most existing key agreement protocols today. However, if it is used without authentication, it is subject to man-in-the-middle attacks in which an attacker can convince two principals who are trying to share a key with each other that they actually do, while in fact they share a key with the attacker. Analyzing Diffie-Hellman without authentication allows us not only to demonstrate how Maude-NPA handles cryptographic functions involving AC axioms, but also how it can be used to find attacks on protocols that use AC operators combined with other algebraic properties of the underlying cryptographic functions.

Example 1. This protocol uses exponentiation to share a secret between two parties, Alice and Bob. The protocol involves an initiator, Alice, and a responder, Bob. We use the common notation $A \hookrightarrow B : M$ to stand for "A sends message M to B". Raising message M to the power of exponent X is denoted by $(M)^X$. There is a public term denoted by g, which will be the base of our exponentiations. We represent the product of exponents by using the symbol $*$. Nonces are represented by N_X, denoting a nonce created by principal X. The protocol description is as follows.

1. $A \hookrightarrow B : A$
   Alice sends her name to Bob.
2. $A \hookrightarrow B : B$
   Alice sends Bob's name to Bob.
3. $A \hookrightarrow B : g^{N_A}$
   Alice creates a new nonce N_A and sends g^{N_A} to Bob.
4. $B \hookrightarrow A : B$
   Bob sends his name to Alice.
5. $B \hookrightarrow A : A$
   Bob sends Alice's name to herself.
6. $B \hookrightarrow A : g^{N_B}$
   Bob creates a new nonce N_B and sends g^{N_B} to Alice.

Intuitively, when Bob receives g^{N_A}, he raises it to the N_B, to obtain ${g^{N_A}}^{N_B} = g^{N_A * N_B}$. Likewise, when Alice receives g^{N_B}, she raises it to the N_A, to obtain ${g^{N_B}}^{N_A} = g^{N_B * N_A}$. And due to the commutativity of the symbol $*$, they know the equivalence $g^{N_B * N_A} = g^{N_A * N_B}$. An observer of the exchange who does not know N_A nor N_B cannot find

$g^{N_A * N_B}$, and so Alice and Bob have computed a shared secret, i.e., $g^{N_A * N_B}$. Of course, the attacker can always learn a term $g^{(N_A * N_I)}$, where N_I is a nonce created by the intruder, even by using a passive intruder model. The point is that he can also make believe to Alice that $g^{(N_A * N_I)}$ is the shared key she is sharing with *Bob*. This is usually modelled by adding to the protocol a new message where Alice sends to Bob some secret, encrypted by $g^{(N_A * N_I)}$. Existence of an attack is expressed by saying that the attacker can obtain this secret. For the sake of simplicity and because we are focused in AC-theories, we omit this last part of the protocol and concentrate just in whether the intruder can learn X^{N_A} for some exponentiation X, where X^{N_A} is the key calculated by Alice.

In a rule-based representation of this protocol, parts of a received message whose make-up cannot be verified by a principal are represented by variables. That is, since nonces are known only to the principal who generated it, and retrieving the nonce would require the computation of a discrete logarithm, we say that Bob receives a variable X of a generic message sort instead of g^{N_A} and similarly for Alice. The symbol $*$ is associative and commutative and satisfies the following additional property with respect to exponentiation:

$$(X^Y)^Z = X^{(Y * Z)}$$

The intruder abilities to create, manipulate, and delete messages according to the Dolev-Yao attackers capabilities [13] are described as follows, where we use the special symbol $\_ \in \mathcal{I}$ to represent that the intruder knows something, and I denotes the intruder's name:

$$\frac{M_1 \in \mathcal{I},\ M_2 \in \mathcal{I}}{(M_1 * M_2) \in \mathcal{I}} \qquad \frac{X \in \mathcal{I},\ Y \in \mathcal{I}}{X^Y \in \mathcal{I}} \qquad \frac{}{N_I \in \mathcal{I}}$$

The intruder also knows the names of all the principals and the base g.

If we ask ourselves whether the intruder can learn a message X^{N_A} for some variable X received by Alice (representing the nonce that Alice receives from Bob), the answer is yes for an infinite set of instances for X, e.g., g^{N_I}, $(g^{N_I})^{N'_I}$, $((g^{N_I})^{N'_I})^{N''_I}$, etc. If we take instantiation $X \mapsto g^{N_I}$, the intruder can learn the message $g^{(N_A * N_I)}$ by means of the following sequence of actions (only the three first steps are necessary but we need Alice to complete the protocol in order to believe she is sharing a shared key with Bob):

1. $A \hookrightarrow B : A$
   Alice sends her name to Bob, but it is intercepted by the intruder.
2. $A \hookrightarrow B : B$
   Alice sends Bob's name to Bob, but it is intercepted by the intruder.
3. $A \hookrightarrow B : g^{N_A}$
   Alice creates a new nonce N_A and sends g^{N_A} to Bob, but it is intercepted by the intruder.
4. $I \hookrightarrow A : B$
   The intruder sends Bob's name to Alice.
5. $I \hookrightarrow A : A$
   The intruder sends Alice's name to Alice.
6. $I \hookrightarrow A : g^{N_I}$
   The intruder creates a new nonce N_I and sends g^{N_I} to Alice.

The intruder is able to learn the message $g^{(N_A * N_I)}$ just by raising the intercepted message g^{N_A} to N_I. Note that the intruder does not need to know N_A, since he gets the desired effect thanks to the equational properties for exponentiation and product of exponents described above.

In this example, the commutativity of symbol $*$ and the equational property of exponentiation are the relevant cryptographic properties of the protocol that have to be considered in order to model both the correct execution of the protocol and to find the attack.

In Section 2, we briefly introduce the Maude-NPA tool. In Section 3, we define the protocol specification and show how the attack is found. We also motivate why we are able to find such an attack. In Section 4, we explain the sort propagation algorithm used to filter out unsorted AC-unifiers returned by the CiME tool. We conclude in Section 5.

2 The Maude-NPA

We briefly introduce the Maude-NPA tool. In [15], the reader can find further details for the case of a confluent and terminating equational theory $\Delta \uplus B$ where the set B of axioms is empty. We are currently extending the framework in [15] to the case where the set B of axioms is non-empty and satisfies appropriate requirements such as the existence of a finitary B-unification algorithm. This paper illustrates the use of the framework in an example where B is AC.

In the Maude-NPA, protocols are specified with a notation derived from strand spaces [16]. In a *strand*, a local execution of a protocol by a principal is indicated by a sequence of messages $[msg_1^-,\ msg_2^+,\ msg_3^-, \ldots,\ msg_{k-1}^-,\ msg_k^+]$ where nodes representing input messages are assigned a negative sign, and nodes representing output messages are assigned a positive sign. In Maude-NPA, strands evolve in time and thus we use the symbol | to divide past and future in a strand, i.e.,

$$[msg_1^{\pm}, \ldots, msg_{j-1}^{\pm} \mid msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}]$$

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).

A *state* is a set of Maude-NPA strands unioned together with an associative and commutativity union operator & along with an additional term describing the intruder knowledge at that point. The *intruder knowledge* is represented using two kinds of facts: positive knowledge facts (the intruder knows m, i.e., $m \in \mathcal{I}$), and negative knowledge facts (the intruder does not know m, i.e., $m \notin \mathcal{I}$), where m is a message expression. Negative facts are essential in our framework to denote terms the intruder will eventually learn in the future, and hence cannot know at the protocol state that we are processing. Negative facts are used by our grammars to describe states unreachable for the intruder; see Section 3.3. The following example illustrates the notion of a state, where we have two strands at different stages of execution, and the intruder does already know the messages g^{N_A}, A, and B, but does not yet know the nonce N_I and the message $(g^{N_I * N_A})$:

$$[\ A^+, B^+, (g^{N_A})^+ \mid B^-, A^-, X^-\ ]\ \&\ [\ nil \mid A^-, B^-, Y^-,\ B^+, A^+, (g^{N_B})^+\ ]\ \&$$
$$\{\ N_I \notin \mathcal{I},\ (g^{N_I * N_A}) \notin \mathcal{I},\ g^{N_A} \in \mathcal{I},\ A \in \mathcal{I},\ B \in \mathcal{I},\ K\ \}$$

Strands communicate between them via the intruder, i.e., by sending a message m to the intruder who will send it back to another strand. When the intruder receives a message, then it learns it, i.e., a message m is learned in a transition from a state with the fact $m \notin \mathcal{I}$ in its intruder knowledge part to a state with the fact $m \in \mathcal{I}$ in its intruder knowledge part (in a forward execution of the protocol). The intruder has the usual ability to read and redirect traffic, and can also perform operations, e.g., encryption, decryption, concatenation, etc., on messages that it has received. Intruder operations are described in terms of the intruder sending messages to itself, which are represented as different strands, one for each action. All intruder and protocol strands are described symbolically, using a mixture of variables and constants, so a single specification can stand for many concrete instances. There is no restriction in the number of principals, number of sessions, nonces, or time, i.e., no data abstraction or approximation is performed. It is also possible to include algebraic properties of the operators (cryptographic and otherwise) as an equational theory and also, as presented in this paper, we can include axioms such as associativity and commutativity with or without identity, or only commutativity; however, associativity without commutativity is problematic because in general it can produce and infinite set of unifiers (see [2]), although in some cases it can be approximated by weaker associative axioms with a finitary unification algorithm (see [14]).

The Maude-NPA's reachability analysis is based on two parameters: a protocol $\mathcal{P}$ represented by strands, and a grammar sequence $\mathcal{G} = \langle G_1, \ldots, G_m \rangle$ used to cut down the search space. Analysis is done in Maude-NPA via backwards narrowing search from an (insecure) goal state SS_{bad} to try to prove or disprove that the insecure state is unreachable from an initial state. States are ($\Delta \uplus B$-)unified with (reversed) rewrite rules describing state transitions via narrowing modulo an equational theory $\Delta \uplus B$. Grammars $\langle G_1, \ldots, G_m \rangle$ represent negative information (or co-invariants), i.e., infinite sets of states unreachable from the initial state. Seed terms $\langle sd_1, \ldots, sd_n \rangle$ represent knowledge that the user believes[5] is not known by the intruder and from which the tool generates the formal languages $\langle G_1, \ldots, G_m \rangle$ (with $m \leq n$) representing several infinite sets of states unreachable for the intruder. These grammars are very important in our framework, since in the best case they can reduce the infinite search space to a finite one, or, at least, can drastically reduce the search space.

The tool tries to deduce whether the protocol is safe for SS_{bad} or not. If the protocol is unsafe, Maude-NPA always terminates with an intruder learning sequence and the exchange message sequence, provided enough time and memory resources are available. If the protocol is unsafe, grammars can actually improve the time and memory consumption by reducing the number of states to be analyzed. If the protocol is safe, the algorithm can often prove it, provided the search space is finite. When the

[5] This initial belief from the user may not always be correct, since some exceptions of the form $X \not\leq t$, describing that the actual value of variable X in the seed term cannot match the term t with variables, may be added to the seed term. The grammar generation process *guarantees* that the grammars finally generated always describe unreachable states.

protocol is safe, grammars are the key technique for producing a finite search space, since they provide a drastic reduction on the search space so that often an infinite search space is reduced to a finite one. If the protocol is safe but the search space is infinite, Maude-NPA runs forever. This provides a semi-decision algorithm. See [15] for further explanations.

The protocol to be analyzed is provided to the tool as an algebraic signature Σ including symbols, sorts, and subsort information (see [22,8]), together with the set $\mathcal{P}$ of strands defining the protocol. Moreover, the tool expects some *seed terms* $\langle sd_1, \ldots, sd_n \rangle$ for the generation of the grammars $\langle G_1, \ldots, G_m \rangle$ where $m \leq n$. In the specification of the protocol-specific signature Σ, there is a special sort Msg for messages. The user will add extra symbols involving the sort Msg. Special algebraic properties of a protocol may be specified with symbols in Σ by means of a set B of equational axioms and a set Δ of equations such that the terms in the axioms B or equations Δ must have sort Msg or a sort smaller than Msg. Note that we can control the level of type checking of the order-sorted protocol specification to fit the attacker model or protocol description (i.e., a completely unsorted protocol specification will be using only the sort Msg for the symbols describing the protocol without any other sort).

In security analyses it is often necessary to use fresh unguessable values, e.g., for nonces. The user can make use of a special sort Fresh in the protocol-specific signature Σ for representing them. The meaning of a variable of sort Fresh is that it will never be instantiated by an E-unifier generated during the backwards reachability analysis. This ensures that if nonces are represented using variables of sort Fresh, they will never be merged and no approximation for nonces is necessary. However, the framework is very flexible, and the user can specify some constant symbols of sort Fresh to play with nonces that can indeed be merged, i.e., approximated. Since variables of sort Fresh are treated in a special way, we make them explicit in the strand definition of the example by writing

$$(r_1, \ldots, r_k : \mathsf{Fresh})\ [msg_1^{\pm}, \ldots, msg_n^{\pm}],$$

where $r_1, \ldots, r_k$ are all the variables of sort Fresh appearing in $msg_1^{\pm}, \ldots, msg_n^{\pm}$.

We impose a requirement on the protocols that can be specified in our tool w.r.t. the algebraic properties specified with a set $E = B \uplus \Delta$ of axioms and equations. Intuitively, a principal can send whatever he/she likes but the pieces of information being processed by the intruder or by a principal are always simplified w.r.t. the oriented version of Δ modulo B. We say a term t is $\rightarrow_{\vec{\Delta}/B}$*-irreducible* if it is a normal form modulo B w.r.t. the oriented version $\vec{\Delta}$ of Δ, i.e., no rule in $\vec{\Delta}$ can be applied to t modulo B. We say that a term t is *strongly* $\rightarrow_{\vec{\Delta}/B}$*-irreducible* if for any substitution σ and variable x such that $\sigma(x)$ is $\rightarrow_{\vec{\Delta}/B}$-irreducible for each x in the domain of σ, then $\sigma(t)$ is $\rightarrow_{\vec{\Delta}/B}$-irreducible. Then, the requirement that we impose on the protocols is that all messages appearing in the reachability and grammar generation stages have to be strongly $\rightarrow_{\vec{\Delta}/B}$-irreducible, except for output messages in a strand (i.e., m^+) and positive knowledge facts (i.e., $m \in \mathcal{I}$); see [15] for details.

Another important aspect of our framework is that everything the intruder can learn must be learned through strands, i.e., the intruder knows nothing in an initial

state. However, this is not a limitation, since we can always include strands of the form $[\ M^+\ ]$ for any message M the intruder is able to know at an initial state.

3 Finding Attacks by Equational Reasoning

3.1 Specification of the Diffie-Hellman Example

Continuing with Example 1, we can denote nonce N_X by $n(X, r)$, where r is a variable of sort Fresh. The generator used as a base is denoted by the symbol g. Raising a message M to the power of an exponent X is denoted by $exp(M, X)$. And the product of exponents X_1, X_2 is denoted by $X_1 * X_2$. The names of all the principals are fixed using constants a and b, since they are just roles. The name of the intruder is denoted by constant i. The order-sorted signature Σ defining the Diffie-Hellman based protocol is the following:

$$a : \rightarrow \mathsf{Name} \qquad b : \rightarrow \mathsf{Name} \qquad i : \rightarrow \mathsf{Name}$$
$$n : \mathsf{Name} \times \mathsf{Fresh} \rightarrow \mathsf{Nonce} \qquad g : \rightarrow \mathsf{Gen}$$
$$exp : \mathsf{Gen}\vee\mathsf{Exp} \times \mathsf{NeNonceSet} \rightarrow \mathsf{Exp} \qquad exp : \mathsf{Gen} \times \mathsf{NeNonceSet} \rightarrow \mathsf{Exp}$$
$$\_*\_ : \mathsf{NeNonceSet} \times \mathsf{NeNonceSet} \rightarrow \mathsf{NeNonceSet}$$

together with the following subsort relations

$$\mathsf{Name}\ \mathsf{NeNonceSet}\ \mathsf{Gen}\vee\mathsf{Exp} < \mathsf{Msg}$$
$$\mathsf{Nonce} < \mathsf{NeNonceSet} \qquad \mathsf{Gen}\ \mathsf{Exp} < \mathsf{Gen}\vee\mathsf{Exp}$$

Algebraic properties are described using the following AC axioms B and equations Δ in E:

$$B = \{\ (X * Y) * Z = X * (Y * Z),\ (X * Y) = Y * X\ \}$$
$$\Delta = \{\ exp(exp(W, Y), Z) = exp(W, Y * Z)\ \}$$

where X, Y, Z are variables of sort $\mathsf{NeNonceSet}$ and W is a variable of sort Gen. Note that the sort of variable W has to be Gen to have a finitary unification algorithm, as discussed in Section 3.2 below. We omit the identity property for symbol $*$ because it is not relevant for this example, although we can handle it as explained in Section 4. In the following, we omit sorts of variables whenever there is no possible confusion. Variables are written in uppercase, except for variables r, r', r'' of sort Fresh.

The strands $\mathcal{P}$ associated to the three protocol steps shown in Example 1 are as follows, one for each principal (or role) in the protocol:

(s1) $(r : \mathsf{Fresh})\ [\ a^+,\ b^+,\ exp(g, n(a, r))^+,\ b^-,\ a^-,\ X^-\ ]$
This strand denotes principal Alice sending her name, Bob's name, and the generator g raised to the power of a new nonce generated by Alice using the Fresh variable r. Then, Alice waits for Bob's name, her name, and an unknown message X.

(s2) $(r' : \mathsf{Fresh})\ [\ a^-,\ b^-,\ Y^-,\ b^+,\ a^+,\ exp(g, n(b, r'))^+\ ]$
This strand denotes principal Bob waiting for Alice's name, his name, and an unknown message Y. Then, Bob sends his name, Alice's name, and the generator g raised to the power of a new nonce generated by Bob using the Fresh variable r'.

The following strands describe the intruder abilities according to the Dolev-Yao attacker's capabilities [13]. Note that the intruder cannot extract information from either an exponentiation or a product of exponents, only compose them.

(s4) $[M_1^-, M_2^-, (M_1 * M_2)^+]$ Multiplication

(s5) $[M_1^-, M_2^-, exp(M_1, M_2)^+]$ Exponentiation

(s6) $[g^+]$ Generator

(s7) $[A^+]$ All names are public

(s8) $(r'' : \mathsf{Fresh})$ $[\ n(i, r'')^+\ ]$ Generation of its own nonces

The strongly $\rightarrow_{\vec{\Delta/B}}$-irreducibility requirement implies that if a message of the form $exp(X, Y)$ where X is of sort Exp appears in an input message of a protocol strand or as a negative knowledge fact $\_ \not\in \mathcal{I}$ of the intruder's knowledge, it must be split into two cases (corresponding to two different protocol states) the message $exp(X, Y)$, which is restricted to strongly $\rightarrow_{\vec{\Delta/B}}$-irreducibility, and the message $exp(g, X' * Y)$, where substitution $\{X \mapsto exp(g, X')\}$ is propagated.

3.2 A Finite Unification Algorithm for $B \uplus \Delta$

We explain why narrowing modulo B provides a finitary unification algorithm for the theory $B \uplus \Delta$, allowing us to automate the backwards reachability analysis. Note that the theory $B \uplus \Delta$ is closely related to a fragment of the theory of exponentiation given in [20] and to larger exponentiation theories in [19], for which finitary unification algorithms are known [20,19]. However, instead of using those algorithms, we adopt the modular hybrid approach described in Section 1 and perform narrowing with Δ modulo B as a $B \uplus \Delta$-unification procedure. We say that $B \uplus \Delta$ is "closely related" to a fragment of the theories in [20,19], because these exponentiation theories are *unsorted*, whereas $B \uplus \Delta$ is an *order-sorted* theory, which turns out to be crucial for narrowing with Δ modulo B to terminate.

The key point is that the term $exp(W, Y * Z)$ is a *constructor term* in the order-sorted sense, but obviously *not* a constructor term in the unsorted sense. A *constructor term* in the unsorted sense is a term built up with only constructor symbols and variables. Given a set of rewrite rules $l_1 \rightarrow r_1, \ldots, l_n \rightarrow r_n$ over an unsorted signature Σ, a function symbol f in Σ is called a *constructor* if it does not appear as the root symbol of any left-hand side $l_1, \ldots, l_n$. In an order-sorted context this notion is more subtle, since the symbol f can be overloaded and can be a constructor (in the sense that no rules apply to it) for some typings and a defined symbol for other typings. That is, the *symbol* f can indeed appear in a lefthand side l_i, but it should never be possible to *type* that lefthand side, or any of its well-sorted substitution instances $\theta(l_i)$, with any of the constructor versions of f. In our case, exp is an overloaded function symbol, for which the typing $exp : \mathsf{Gen} \times \mathsf{NeNonceSet} \rightarrow \mathsf{Exp}$ is indeed a constructor operator in the order-sorted sense, since any well-typed substitution instance of Δ's lefthand side $exp(exp(W, Y), Z)$ must necessarily have for its top symbol the typing $exp : \mathsf{Gen} \vee \mathsf{Exp} \times \mathsf{NeNonceSet} \rightarrow \mathsf{Exp}$, and can never have the typing $exp : \mathsf{Gen} \times \mathsf{NeNonceSet} \rightarrow \mathsf{Exp}$. In particular, $exp(W, Y * Z)$ is a constructor term as claimed.

The next key observation is that Δ is confluent, terminating, and coherent modulo B (see [18] for these notions). Coherence modulo B is particularly easy to check, since the associative-commutative multiplication symbol in B does not appear in Δ's lefthand side $exp(exp(W, Y), Z)$. We can then use Theorem 5 in [18] to ensure that narrowing with Δ modulo B provides a complete $B \uplus \Delta$-unification algorithm; and Theorem 9 in [18], plus the fact that $exp(W, Y * Z)$ is a constructor term, to further conclude that narrowing with Δ modulo B terminates and therefore such a $B \uplus \Delta$-unification algorithm is finitary.

For all this to work we need the Maude-NPA to support *order-sorted* unification modulo B. How this is achieved, not only for the above theory B, but for any combination of associativity, commutativity, and identity axioms (except for associativity without commutativity) is explained in Section 4.

3.3 Reducing the Size of the Search Space

Grammars representing several infinite sets of states unreachable for the intruder are very important in our framework, since they represent negative information (co-invariants) that will be used to cut down the search space.

As explained in [15], the Maude-NPA starts out with the seed terms, which represent knowledge that the user believes is not known by the intruder. There are only two types of seed terms: (i) $\emptyset \mapsto t \in \mathcal{L}$, denoting that the term t of sort Msg is unknown for the intruder without any restriction, and (ii) $t|_p \notin \mathcal{I} \mapsto t \in \mathcal{L}$, denoting that the term t of sort Msg is unknown for the intruder provided that the subterm $t|_p$ is certainly unknown by the intruder, i.e., provided that the fact $t|_p \notin \mathcal{I}$ appears in the intruder's knowledge of the state of the protocol that we will be processing at each moment. The initial grammar for Example 1 containing four language productions is as follows:

$$\emptyset \mapsto X * n(a, r) \in \mathcal{L} \qquad \emptyset \mapsto n(a, r) \in \mathcal{L} \qquad \emptyset \mapsto X * n(b, r) \in \mathcal{L} \qquad \emptyset \mapsto n(b, r) \in \mathcal{L}$$

For instance, language production $\emptyset \mapsto X * n(a, r) \in \mathcal{L}$ denotes a formal language including any message $t_1 * t_2$ such that subterm t_1 is of sort NeNonceSet and subterm t_2 is of the form $n(a, r)$ where a is the constant denoting Alice and r is a variable. The final grammar turns out to be the same as the initial grammar, and thus we omit the details about the grammar generation, which can be found in [15]. Therefore, any message m belonging to the formal language denoted by $\emptyset \mapsto X * n(a, r) \in \mathcal{L}$ is unknown for the intruder and any state containing m as an input message (i.e., m^-) or as part of the intruder knowledge (i.e., $m \in \mathcal{I}$) is unreachable for the intruder and discarded.

In this work we also make use of an additional state-reduction feature, namely a notion similar to the "lazy intruder" of Basin et al. [3], but for backwards instead of forward search. This feature was already implemented in the original NRL Protocol Analyzer [21] and is independent of the use of any equational theory. Note that this feature is an *additional* state-reduction feature complementing the state reduction provided by grammars, but both features are useful.

We give the main intuitions but do not explain this feature in detail; a detailed explanation will appear elsewhere. The key idea behind our lazy intruder is to refrain from searching for terms that we know can easily be found by the intruder. At the same time, if the term becomes later instantiated to something that is not so readily

obtainable, then we rollback to the state in which the term first appeared in the intruder knowledge. Terms that the intruder can easily find include variable terms, publicly known terms, and terms which an intruder could compute from variables and publicly known terms. The expression $exp(g, X)$ in our Diffie-Hellman protocol is an example of such a lazy term, since g is publicly known, X is a variable, and exp is computable by the intruder.

3.4 Backwards Reachability Analysis

The final state we are looking for is one in which Alice receives Y as the final message and computes $exp(Y, n(a, r))$, while the intruder also learns $exp(Y, n(a, r))$. As explained in Section 3.1, we actually specify two final attack states, one in which the intruder learns $exp(Y, n(a, r))$, and one in which the intruder learns $exp(g, X * n(a, r))$, where both terms are strongly $\rightarrow_{\overrightarrow{\Delta/B}}$-irreducible. Since Y is of type Gen or Exp, the first case is irreducible only when $Y = g$. This corresponds to the case in which the intruder sends g to a, which can be detected and discarded by a. We treat the second case in more detail below.

The *final attack state pattern* to be given as input to the system is:

$$(r : \mathsf{Fresh})\ [\ a^+,\ b^+,\ exp(g, n(a, r))^+,\ b^-,\ a^-,\ exp(g, X)^-\ |\ nil\ ]\ \&$$
$$\{\ exp(g, X * n(a, r)) \in \mathcal{I},\ K\ \}$$

The intruder knowledge is essential in the backwards reachability process and thus variable K will be appropriately instantiated by narrowing. Using the above attack state pattern, the grammar with four language productions, and the lazy intruder feature described above, our tool is able to find the following initial state of the protocol:

$$(r : \mathsf{Fresh})\ [\ nil\ |\ a^+,\ b^+,\ exp(g, n(a, r))^+,\ b^-,\ a^-,\ exp(g, X)^-\ ]\ \&$$
$$[\ nil\ |\ exp(g, n(a, r))^-,\ X^-,\ exp(g, X * n(a, r))^+\ ]\ \&$$
$$\{\ exp(g, n(a, r)) \notin \mathcal{I},\ exp(g, X * n(a, r)) \notin \mathcal{I}\ \}$$

Note that the second strand is, indeed, an intruder strand introduced by the backwards reachability process denoting that whenever the intruder knows the message $exp(g, n(a, r))$ (variable r is bound in this context) and knows any message X of sort $\mathsf{NeNonceSet}$, he can combine them and generate message $exp(g, X * n(a, r))$. Intruder strands generating messages b^-, a^-, and $exp(g, X)^-$ are not included in the initial state, since the lazy intruder avoids searching for them. The previous state is an initial state of the protocol because all the strands are in their initial position (i.e., every message is in the future), and the intruder does not know but eventually will learn messages exchanged in the protocol run. The concrete message exchange sequence leading to this attack is:

$$a^+ . b^+ . exp(g, n(a, r))^+ . exp(g, n(a, r))^- . X^- . exp(g, X * n(a, r))^+ . b^- . a^- . exp(g, X)^-$$

which corresponds to the intuitive description of the attack explained in Section 1.2. Actually, the tool returns an infinite number of attacks, since X can also be the

product of any number k of intruder nonces; one attack is returned for each positive k.

4 Order-sorted Unification Modulo Equations

As the protocol is specified using an order-sorted theory with AC axioms, narrowing requires an order-sorted AC unification procedure. In fact, the use of order-sorted features is essential for AC-narrowing to terminate with the given protocol's equations Δ. One major challenge in developing such a procedure is that producing an efficient AC unification procedure takes considerable effort, yet existing tools supporting AC unification such as CiME [11] only support *unsorted* theories rather than order-sorted theories.

Our solution has been to partition order-sorted AC unification into two steps: (1) we drop the sort information and use CiME [11] to compute a complete set of unsorted unifiers; and (2) we apply a *sort propagation* algorithm, described below, which constructs zero or more order-sorted unifiers from each unsorted unifier. Our approach is not restricted only to AC unification, since we allow any combination of associativity, commutativity, and identity, except associativity without commutativity.

Although theoretically there may be many more order-sorted unifiers than unsorted unifiers, in practice we have found that the opposite is usually the case. Many unsorted unifiers can be eliminated when considering sort constraints, because a variable is bound to a term with an incompatible type. For example, we can eliminate an unsorted unifier θ which binds a variable `X:Nonce` to a term `(Y:Nonce ; Z:Nonce)` representing a nonce set.

Due to the fact that the unsorted unification procedure ignores sort information and the sort propagation algorithm ignores the equations, the order-sorted theory $\mathcal{E} = (\Sigma, B)$ where $\Sigma = (S, F, \leq)$ is an order-sorted signature with poset of sorts $(S, \leq)$ and function symbols $F = \{F_{w,s}\}_{(w,s)\in S^* \times S}$, is required to satisfy several requirements:

- Each connected component $[s] \in S/\equiv_{\leq}$, where $\equiv_{\leq}$ is the equivalence generated by the subsort ordering $\leq$, must have a top-most sort $\top_{[s]}$;
- Each axiom $l = r \in B$ must be *sort-preserving* and each variable in $\mathcal{V}ar(l) \cup \mathcal{V}ar(r)$ must have a top-most sort; and
- Each term $t \in T_\Sigma(X)$ must have a unique *least sort* s with $t \in T_\Sigma(X)_s$.

These requirements are not too restrictive for our purposes. We can guarantee that each commutativity and identity equation is sort-preserving by introducing extra sorts and operators declarations to complete the theory. The other requirements are already required by Maude for efficiency purposes, and are checked automatically by Maude.

4.1 Unsorted Unification

As a first step, the sorts from the order-sorted signature $\Sigma = (S, F, \leq)$ are dropped to obtain an unsorted signature $\overline{F}$, where each operator $f \in F_{s_1,\ldots,s_n,s}$ is mapped to an unsorted and disambiguated operator $f_{\top_{s_1},\ldots,\top_{s_n},\top_s}$ where its original connected components are made explicit. Given the unification problem $t =_B u$, we rename the

operators in t and u and make the variables unsorted in a set $\overline{X}$ to obtain unsorted terms $\overline{t}, \overline{u} \in T_{\overline{F}}(\overline{X})$. We then pass the unsorted unification problem $\overline{t} =_{\overline{B}} \overline{u}$ to CiME. The unsorted most-general unifiers modulo $\overline{B}$, denoted $\overline{\theta}_1, \ldots, \overline{\theta}_n$, are obtained from CiME, and we reverse the renaming process to obtain mappings $\theta_1, \ldots, \theta_n : X \rightarrow T_\Sigma(X)$. At this point, each mapping θ_i is not necessarily an order-sorted unifier, because variables may be bound to terms that have a sort incompatible with the sort of the variable. To remedy this situation and obtain a complete set of order-sorted unifiers, we pass each mapping θ_i to the sort propagation algorithm described below, and take the union of all the order-sorted unifiers returned for each unsorted unifier.

4.2 Sort Propagation

The *sort propagation algorithm* generates a complete set of order-sorted unifiers $\{\theta_1, \ldots \theta_n\}$ from a given mapping $\theta : X \rightarrow T_\Sigma(X)$ such that any possible order-sorted unifier ψ to the problem $t =_B u$ that is an instance of the mapping θ is an instance of one of the unifiers θ_i generated by the algorithm. The algorithm maintains a disjunctive set $D = \{ C_0, \ldots C_m \}$ of *sort constraint problems* where each C_i is a conjunction $(t_1 : s_1) \wedge \cdots \wedge (t_{n_i} : s_{n_i})$. A *solution* to C_i is a substitution $\phi : X \rightarrow T_\Sigma(X)$ such that $t_j\phi \in T_\Sigma(X)_{s_j}$ for $j \leq n_i$. A solution to D is a substitution ϕ that is a solution to at least one problem in the set. The correctness of the algorithm is obtained by observing that although the problems change while the algorithm executes, the set of solutions to the problems does not.

From θ, we construct the initial set $D_0 = \{(\theta(x_1) : s_1) \wedge \cdots \wedge (\theta(x_n) : s_n)\}$, where $x_1, \ldots x_n \in X$ are all the variables such that $\theta(x_i) \neq x_i$ and s_i is the sort of the variable x_i for $i \leq n$. We then exhaustively apply the transformation rules in Fig. 1 on D_0 to obtain disjunctive sets $D_1, D_2, \ldots, D_r$. This process terminates, because each application either reduces the number of operator symbols appearing in a problem or reduces the total number of predicates in a problem. In the final set $D_r = \{ C_1, \ldots, C_n \}$, each problem C_i has the form $(x_1 : s_1) \wedge \cdots \wedge (x_{n_i} : s_{n_i})$ where a variable $x \in X$ appears in a most one predicate $(x_i : s_i)$. From each C_i, we generate a sort specialization mapping $\phi_i : X \rightarrow X$ that maps each variable x_j with $j \leq n_i$ to a fresh variable with sort s_j. The final set of order-sorted unifiers generated by θ is then the set $\{\theta_1, \ldots, \theta_m\}$ where $\theta_i = \phi_i \circ \theta$.

The final set of order-sorted unifiers is complete, because each order-sorted unifier ϕ can be seen as an instance of a mapping derived from an unsorted unifier $\overline{\theta}$ returned by CiME, and a sort specialization mapping ϕ_i. The latter can be seen by observing that each application of an inference rule in Fig. 1 preserves the set of solutions under the assumptions that the equations B are sort-preserving and that each term $t \in T_\Sigma(X)$ has a least sort.

5 Conclusions

In this paper we have illustrated the extension of the Maude-NPA to reason about protocols whose equational theories are given by confluent and terminating equations Δ modulo some axioms B, using a Diffie-Hellman example where $B = AC$. Furthermore, the use of order-sorted theories has been shown to be important in obtaining unification algorithms for $\Delta \uplus B$. Although at present, the Maude-NPA can

$$\text{Valid} \qquad \frac{\{\, (t : s) \wedge C \,\} \cup D}{\{\, C \,\} \cup D} \quad \text{if } t \in T_\Sigma(X)_s$$

$$\text{Conjoin} \qquad \frac{\{\, (t : s_1) \wedge (t : s_2) \wedge C \,\} \cup D}{\bigcup_{s \,\in\, \mathrm{glb}(s_1, s_2)} \{\, (t : s) \wedge C \,\} \cup D}$$

$$\text{Prop.} \qquad \frac{\{\, (f(t_1, \ldots, t_n) : s) \wedge C \,\} \cup D}{\bigcup_{s_1 \ldots s_n \in \sup(\mathrm{ar}(f,s))} \{\, (t_1 : s_1) \wedge \cdots \wedge (t_n : s_n) \wedge C \,\} \cup D}$$

where
$$\mathrm{glb}(s_1, s_2) = \sup(\{\, s \in S \mid s \leq s_1 \wedge s \leq s_2 \,\}),$$
$$\mathrm{ar}(f, s) = \{\, w \in S^* \mid f \in F_{w,s} \,\}, \text{ and}$$
$$\sup(W) = \{\, w \in W \subseteq S^* \mid (\forall w' \in W)\, w \leq w' \Rightarrow w = w' \,\}.$$

Fig. 1. Sort Propagation Transformation Rules

handle order-sorted theories $\Delta \uplus B$ where B can contain any combination of associativity, commutativity, and identity (except associativity without commutativity), much work remains ahead, such as: (i) building in order-sorted unification algorithms for efficiency purposes; (ii) improving grammar formalisms to reduce the search space in the modulo B case; (iii) developing additional optimization techniques; and (iv) developing other case studies involving other equational theories.

References

1. Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under (many more) equational theories. In *CSFW*, pages 62–76. IEEE Computer Society, 2005.
2. F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1 of *Volume*, chapter 8, pages 445–532. Elsevier Science, 2001.
3. David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.
4. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 2007. To appear.
5. Yohan Boichut, Pierre-Cyrille Héam, and Olga Kouchnarenko. Handling algebraic properties in automatic analysis of security protocols. In Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone, editors, *Theoretical Aspects of Computing - ICTAC 2006, Third International Colloquium, Tunis, Tunisia, November 20-24, 2006, Proceedings*, volume 4281 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2006.
6. Sergiu Bursuc, Hubert Comon-Lundh, and Stéphanie Delaune. Associative-commutative deducibility constraints. In Wolfgang Thomas and Pascal Weil, editors, *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS'07)*, volume 4393 of *Lecture Notes in Computer Science*, pages 634–645, Aachen, Germany, February 2007. Springer.
7. Yannick Chevalier, Ralf Küsters, Michael Rusinowitch, and Mathieu Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In 23^{rd} *Conference on Foundations Software Technology and Theoretical Computer Science*, volume 2914 of *Lecture Notes in Computer Science*, pages 124–135, 2003.

8. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.
9. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive-or. In 18^{th} *Annual IEEE Symposium on Logic in Computer Science (LICS '03)*, pages 271–280, 2003.
10. Hubert Comon-Lundh and Véronique Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In R. Nieuwenhuis, editor, *Proc. of 14th International Conference on Rewriting Techniques and Applications, RTA'03*, volume 2706, pages 148–164, 2003.
11. E. Contejean, C. Marché, B. Monate, and X. Urbain. Proving termination of rewriting with CiME. In A. Rubio, editor, *6th International Workshop on Termination*, Techreport DSIC II/15/03, pages 71–73. Universidad Politécnica de Valencia, 2003.
12. Stéphanie Delaune, Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Symbolic protocol analysis in presence of a homomorphism operator and *exclusive or*. In Michele Buglesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP'06) — Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 132–143, Venice, Italy, July 2006. Springer.
13. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transaction on Information Theory*, 29(2):198–208, 1983.
14. Santiago Escobar, Catherine Meadows, and José Meseguer. Equational cryptographic reasoning in the Maude-NRL Protocol Analyzer. In *Proc. of the First International Workshop on Security and Rewriting Techniques (SecReT 2006)*, Electronic Notes in Theoretical Computer Science. Elsevier Sciences Publisher, 2006. To appear.
15. Santiago Escobar, Catherine Meadows, and José 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.
16. F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.
17. Jean Goubault-Larrecq, Muriel Roger, and Kumar Neeraj Verma. Abstraction and resolution modulo AC: How to verify Diffie-Hellman-like protocols automatically. *Journal of Logic and Algebraic Programming*, 64(2):219–251, 2005.
18. J.-P. Jouannaud, C. Kirchner, and H. Kirchner. Incremental construction of unification algorithms in equational theories. In *Proc. ICALP'83*, pages 361–373. Springer LNCS 154, 1983.
19. Deepak Kapur, Paliath Narendran, and Lida Wang. An E-unification algorithm for analyzing protocols that use modular exponentiation. In R. Nieuwenhuis, editor, *Proc. of 14th International Conference on Rewriting Techniques and Applications, RTA'03*, volume 2706, pages 165–179, 2003.
20. C. Meadows and P. Narendran. A unification algorithm for the group Diffie-Hellman protocol. In *Proc. Workshop on Issues in the Theory of Security WITS 2002*, 2002.
21. Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of logic programming*, 26(2):113–131, 1996.
22. José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
23. J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proceedings of the IEEE Computer Security Foundations Workshop (CSFW-16)*, pages 47–61, 2003.