Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 171 (2007) 23–36

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Equational Cryptographic Reasoning in the Maude-NRL Protocol Analyzer

Santiago Escobar[a,1 ,4] Catherine Meadows[b,2] José Meseguer[c,3]

[a] *Universidad Politécnica de Valencia, Spain.*

[b] *Naval Research Laboratory. Washington, DC, USA.*

[c] *University of Illinois at Urbana-Champaign, USA.*

Abstract

The NRL Protocol Analyzer (NPA) is a tool for the formal specification and analysis of cryptographic protocols that has been used with great effect on a number of complex real-life protocols. One of the most interesting of its features is that it can be used to reason about security in face of attempted attacks on low-level algebraic properties of the functions used in a protocol. Recently, we have given for the first time a precise formal specification of the main features of the NPA inference system: its grammar-based techniques for (co-)invariant generation and its backwards narrowing reachability analysis method; both implemented in Maude as the Maude-NPA tool. This formal specification is given within the well-known rewriting framework so that the inference system is specified as a set of rewrite rules modulo an equational theory describing the behavior of the cryptographic symbols involved. This paper gives a high-level overview of the Maude-NPA tool and illustrates how it supports equational reasoning about properties of the underlying cryptographic infrastructure by means of a simple, yet nontrivial, example of an attack whose discovery essentially requires equational reasoning. It also shows how rule-based programming languages such as Maude and complex narrowing strategies are useful to model, analyze, and verify protocols.

Keywords: Cryptographic Protocol Verification, Equational Reasoning, Narrowing, Rewriting Logic, Maude

1 Introduction

The NRL Protocol Analyzer (NPA) [16] is a tool for the formal specification and analysis of cryptographic protocols that has been used with great effect on a number of complex real-life protocols. One of the most interesting of its features is that it can be used not only to prove or disprove authentication and secrecy properties using the standard Dolev-Yao model [9], but also to reason about security in face of

[1] Email:sescobar@dsic.upv.es

[2] Email:meadows@itd.nrl.navy.mil

[3] Email:meseguer@cs.uiuc.edu

[4] S. Escobar has been partially supported by the EU (FEDER) and Spanish MEC TIN-2004-7943-C04-02 project, the Generalitat Valenciana under grant GV06/285, and the ICT for EU-India Cross-Cultural Dissemination ALA/95/23/2003/077-054 project.

doi:10.1016/j.entcs.2007.02.053

attempted attacks on low-level algebraic properties of the functions used in a protocol. Indeed, it has been used successfully to either reproduce or discover a number of such attacks, ranging from the discovery of an authentication attack based on the cancellation properties of encryption and decryption [15], to the reproduction of Bellovin's attack on a version of the Encapsulating Security Protocol that used cipher block chaining, and to the discovery of a sophisticated type confusion attack [20,17] that resulted in the redesign of a draft for a protocol standard.

NPA's ability to reason well about these low-level functionalities is its combination of symbolic reachability analysis using narrowing, together with its grammar-based techniques for reducing the size of the search space. On one hand, unification modulo algebraic properties (e.g., encryption and decryption, concatenation and deconcatenation) as narrowing using a finite convergent (i.e., confluent and terminating) set of rewrite rules where the right-hand side of each rule is either a subterm of the left-hand side or a ground term [8] allows the tool to represent behavior which is not captured by the usual Dolev-Yao free algebra model. On the other hand, techniques for reducing the size of the search space by using inductively defined co-invariants describing states unreachable to an intruder allows us to start with an infinite search space and reduce it in many cases to a finite one, thus freeing us from the requirement to put any a priori limits on the number of sessions.

Recently, we have given for the first time [10] a precise formal specification of the main features of the NPA inference system: its backwards reachability analysis method and its grammar-based techniques for co-invariant generation. This formal specification is given within the well-known rewriting framework so that the inference system is specified as a set of rewrite rules modulo an equational theory describing the behavior of the cryptographic symbols involved. The inference system has been implemented in Maude, which we called the *Maude-NRL Protocol Analyzer* (Maude-NPA) tool. We have used the rewriting framework to prove some important meta-logical properties, such as the soundness and completeness of its search algorithm, i.e., the tool will discover an attack of the type specified by the user if and only if such an attack exists at the level of abstraction supported by the model, and the unreachability of the states characterized by grammars, i.e., the drastic state space reductions afforded by such grammars do not compromise the completeness of the search algorithm.

But [10] provides also a modular basis that can be viewed as a first step towards reaching a longer-term goal, i.e., extending the Maude-NPA's inference system to support the analysis of cryptographic protocols under a wider range of algebraic properties than it currently is capable of, with the ultimate plan of building a next-generation rewriting-based analysis tool that takes into account the algebraic properties of the underlying cryptographic functions. In Section 3 we illustrate the ease with which new equational properties of the cryptographic infrastructure can be modularly added to the Maude-NPA by demonstrating how these techniques can be applied to the analysis of type confusion attacks.

Type confusion attacks, first discussed in detail in [12], are attacks which exploit ambiguity in formatting to allow an attacker to convince a principal that data of one type (e.g., a name) is data of another type (e.g., a nonce). Because of the level of protocol model detailing that is required, realistic examples of type confusion attacks in the literature of formal analysis of cryptographic protocols are rare. Moreover, most existing protocol tools do not allow for the possibility, for example, of concatenations of terms being confused with a single term, or of pieces of terms with a whole term. However, in [17] the authors describe an example of using the NRL Protocol Analyzer to find an attack of this sort. The attack is on an early version of the Group Domain of Interpretation Protocol [4], which has two parts, each of which makes use of digitally signed messages in a different way. In the early version, the digitally signed messages began and ended with nonces, which made it possible for an attacker to pass off a string of data of different types as a nonce in a way that such string of data can be interpreted later as another type of signed message. However, the NRL Protocol Analyzer's inability to model associativity of concatenation meant that it was only able to find one of the attacks; the other had to be found by hand. In the following example, we consider a simpler protocol which is subject to a similar type confusion, and show how it can be handled by adding a limited version of associativity to the Maude-NPA specification.

Example 1.1 This protocol uses digital signatures to achieve authentication between two parties, Alice and Bob. The protocol involves an initiator, Alice, a responder, Bob, and a server S providing nonces. We use the common notation $A \hookrightarrow B : M$ to stand for "A sends message M to B". Encryption of message M using the public key of principal X is denoted by $\{M\}_X$, where we assume that public keys are known by all the principals, including the intruder. Encryption using the private key is denoted by $\{M\}_{X^{-1}}$, where we assume that a private key is known only by its owner. Decryption is done by combination of public and private keys, i.e., $\{M\}_X$ can be decrypted only by principal X, who knows the private key X^{-1}, and $\{M\}_{X^{-1}}$ can be decrypted by every principal. Nonces are represented by N_X, denoting a nonce created by principal X. The protocol description proceeds as follows.

(i) $S \hookrightarrow A : N_S$
The server S sends a nonce to A.

(ii) $A \hookrightarrow B : \{N_S, S\}_{A^{-1}}$
A sends the nonce provided by the server and the server's name to B, signed with its private key.

(iii) $A \hookrightarrow B : \{B, N_A, S\}_{A^{-1}}$
Then, A sends B name, a new nonce, and the server's name to B, signed again with its private key.

In a rule-based representation of this protocol, parts unknown for a principal are represented by a variable. That is, since nonces are known only to the principal who generated it, we say that Alice receives message X and generates messages $\{X, S\}_{A^{-1}}$ and $\{B, N_A, S\}_{A^{-1}}$, and Bob receives messages $\{X', S\}_{A^{-1}}$ and $\{B, X'', S\}_{A^{-1}}$.

If we ask whether a principal acting as Bob can get confused and understand the message $\{B, X'', S\}_{A^{-1}}$, expected as its second message, as if it was its first expected message $\{X', S\}_{A^{-1}}$, then we can intuitively answer no, but we will be assuming some type checking over the messages. In an actual implementation of the protocol type-checking would be implemented by format checking. If the format is defined ambiguously, then type-checking will not be implemented properly and types may be confused. For example, this is what happened in the analysis of the GDOI protocol. For the purpose of our simple type confusion analysis we will assume that, although names have a recognized format, and hence a recognized type, nonces do not. Thus, for example, a name followed by a nonce can be confused with a nonce. Therefore, in this paper, we do not perform type checking on the variable X' of the first expected message $\{X', S\}_{A^{-1}}$. In this situation, when associativity of the pairing (or concatenation) operator $\_,\_$ is taken into account, there is a possible confusion[5] between the generated message $\{B, N_A, S\}_{A^{-1}}$ and the expected message $\{X', S\}_{A^{-1}}$, i.e., message $\{B, N_A, S\}_{A^{-1}}$ can be understood as $\{B, (N_A, S)\}_{A^{-1}}$ or $\{(B, N_A), S\}_{A^{-1}}$ depending on who receives it.

In this example, associativity is a cryptographic property of the protocol that has to be considered in order to find the flaw in the protocol. However, it is well-known that general associative unification problems can yield an infinite number of unifiers [2] and, therefore, our approach is to provide an approximation of associativity for up to 3 elements. On the other hand, note that this confusion appears only when two sessions of the protocol are executed at the same time, i.e., the message $\{B, N_A, S\}_{A^{-1}}$ is generated in one session, and the message $\{X', S\}_{A^{-1}}$ is expected in the other session. This is relevant when compared to other approaches that impose several restrictions, as in the case of a bounded number of sessions, or that perform approximations of the protocol, as in the case of a finite number of messages or fresh nonces. The ProVerif tool [5] allows an unbounded number of sessions but abstractions are performed on fresh data and thus false attacks can be obtained. The OFMC tool in the AVISPA tool [1] follows a similar approach to ours, since they consider "depth parameters" for unification problems in some equational theories [3]. However, they consider only a bounded number of sessions. The Scyther tool [7] allows an unbounded number of sessions and does not apply abstractions on fresh data, but no cryptographic symbols are considered.

2 The Maude-NPA

In the Maude-NPA, protocols are specified with a notation quite close to that of strand spaces [11]. 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. Strands evolve in time and

[5] Technically speaking, this confusion is only possible under the assumption that the variable X' in the expected message $\{X', S\}_{A-1}$ is a variable of a generic sort message instead of a variable of a sort nonce and such that the generic sort message can contain a message of variable length, i.e., variables X' can be a sequence $\{t_1, \ldots, t_k\}$ of messages where $1 \leq k \leq 3$.

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 strands together with the intruder knowledge at that point. The *intruder knowledge* is represented using two kinds of facts: a positive knowledge fact is denoted by $m{\in}\mathcal{I}$ and a negative knowledge fact is denoted by $m{\notin}\mathcal{I}$, where m is a message expression. The following example illustrates the notion of a state, where we have two strands at different time positions and the intruder does know the message $\{N, S\}_{A^{-1}}$ but does not know yet the message $\{B, N_A, S\}_{A^{-1}}$ (note that N is a variable whose concrete variable is still unknown):

$$\begin{array}{l}[\ N^{-},\ (\{N,S\}_{A^{-1}})^{+} \mid (\{B,N_A,S\}_{A^{-1}})^{+}\ ]\ \&\\ [\ nil \mid (\{M,S\}_{A^{-1}})^{-},\ (\{B,N_A,S\}_{A^{-1}})^{-}\ ]\ \&\\ \{\ (\{N,S\}_{A^{-1}}){\in}\mathcal{I},\ (\{B,N_A,S\}_{A^{-1}}){\notin}\mathcal{I},\ K\ \}\end{array}$$

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 multiple 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.

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}. States are (E-)unified with (reversed) rewrite rules describing state transitions via narrowing modulo an equational theory E. Grammars $\langle G_1, \ldots, G_m \rangle$ represent negative information (or co-invariants), i.e., 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. In this case, grammars can actually improve the time and memory consumption by reducing the number of states to analyze. If the protocol is safe, the algorithm can often prove it, provided the search space is finite. In this case, grammars are the key and provide a drastic reduction on the search space such that 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 [10] for further explanations.

The protocol to be analyzed is provided to the tool as a signature Σ including symbols, sorts, and subsort information (see [19,6]), 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 $m \leq n$. Although the user is required to supply the seed terms for the grammars, we are studying how to automatically generate them. These seed terms $\langle sd_1, \ldots, sd_n \rangle$ represent knowledge that the user believes is not known by the intruder and from which the tool generates the formal languages $\langle G_1, \ldots, G_m \rangle$ representing several infinite sets of states unreachable for the intruder. 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 and a set E of equations such that the sort of the terms of the equations must be Msg or smaller. 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 of 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 E. We say a term t is $\rightarrow_{\vec{E}}$*-irreducible* if it is a normal form w.r.t. the oriented version $\vec{E}$ of E, i.e., no rule in $\vec{E}$ can be applied to t. We say that a term t is *strongly* $\rightarrow_{\vec{E}}$-*irreducible* if for any substitution σ and variable x such that $\sigma(x)$ is $\rightarrow_{\vec{E}}$-irreducible, then $\sigma(t)$ is $\rightarrow_{\vec{E}}$-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 $\rightarrow_{\vec{E}}$-irreducible, except output messages in a strand (i.e., m^+) and positive knowledge facts (i.e., $m{\in}\mathcal{I}$).

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 write strands $[\ M^+\ ]$ for any message M the intruder is able to know at an initial state.

3 Finding Attacks by Equational Reasoning

3.1 Protocol Specification

Continuing Example 1.1, we can denote nonce N_X by $n(X,r)$, where r is a variable of sort Fresh. Concatenation of two messages is denoted by the operator $\_;\_$, e.g., $n(A,r);n(B,r')$. Encryption of a message M with the public key of principal A is denoted by $pk(A,M)$, e.g., $\{N_S,S\}_A$ is denoted by $pk(A,n(S,r);S)$. Encryption of a message with a private key is denoted by $sk(A,M)$, e.g., $\{N_S,S\}_{A^{-1}}$ is denoted by $sk(A,n(S,r);S)$. The names of all the principals are fixed, since they are just roles, where the intruder is denoted by constant i. The only secret key operation the intruder can perform is $sk(i,m)$ for a known message m. The order-sorted signature Σ defining the protocol is the following:

$$a : \rightarrow \mathsf{Name} \qquad b : \rightarrow \mathsf{Name} \qquad s : \rightarrow \mathsf{Name} \qquad i : \rightarrow \mathsf{Name}$$

$$pk : \mathsf{Name} \times \mathsf{Msg} \rightarrow \mathsf{Enc} \quad sk : \mathsf{Name} \times \mathsf{Msg} \rightarrow \mathsf{Enc} \quad n : \mathsf{Name} \times \mathsf{Fresh} \rightarrow \mathsf{Nonce}$$

$$\_\,;\,\_ : \mathsf{Msg} \times \mathsf{Elem} \rightarrow \mathsf{Msg} \quad \_\,;\,\_ : \mathsf{Msg} \times \mathsf{Msg} \rightarrow \mathsf{Msg}$$

together with the following subsort relations

$$\mathsf{Name}\ \mathsf{Nonce}\ \mathsf{Enc} < \mathsf{Elem} \qquad\qquad \mathsf{Elem} < \mathsf{Msg}$$

The encryption/decryption cancellation properties are described using the following equations in E:

$$pk(K,sk(K,M)) = M \qquad\qquad sk(K,pk(K,M)) = M$$

where K is a variable of sort Name and M is a variable of sort Msg. We now discuss the algebraic properties of the concatenation operator. The obvious requirement is to make $\_;\_$ associative. However, it is well-known that general associative unification problems can yield an infinite number of unifiers [2]. This could cause infinite branching in our narrowing algorithm and preclude termination of the Maude-NPA. Our approach is to provide a *sound* approximation of associativity by a weaker subtheory of it having a finitary unification algorithm. We achieve this by exploiting the order-sorted structure of our specification and using well-known narrowing results.

The weaker instance of associativity that we add to E is the equation [6]

$$X; (Y; Z) = (X; Y); Z$$

where the variables X, Y, Z are of sort Elem. Note that the rules obtained by orienting this and the previous equations in E from left to right provide a confluent, terminating, and sort-decreasing term rewriting system. It is then well-known that basic narrowing [13] provides a sound and complete E-unification algorithm, which is *finitary* if the right-hand side of each equation in E is either a constructor term [7], a subterm of the left-hand side, or a ground term [8]. Note that the encryption/decryption equations satisfy this requirement. Our associativity equation also does, because when the variables X, Y, Z are of sort Elem the term $(X; Y); Z$ is a constructor term, since the overloaded typing $\_;\_ : \mathsf{Msg} \times \mathsf{Elem} \to \mathsf{Msg}$ is a constructor operator in the sense explained in Footnote 7. Therefore, narrowing provides a finitary E-unification algorithm for our theory E.

Our associativity equation is of course a very restrictive subtheory of the full theory of associativity, but it has the important advantage of yielding, together with the encryption/decryption equations, a finitary E-unification algorithm by basic narrowing. Using order-sorted techniques, we can similarly define increasingly more expressive *bounded associativity theories*, having associativity laws for strings of up to n elements for any $n \geq 3$. The advantage of such theories is that, by arguments similar to the one given here for $n = 3$, their unification algorithms by narrowing are all finitary; see Footnote 6.

In the following, variables M, M', M_1, M_2, M'_1, M'_2 are of sort Msg, variables X, Y, Z are of sort Elm, variable N is of sort Nonce, variables r, r', r'', r''' are of sort Fresh, and variable A is of sort Name. Note that the strongly $\rightarrow_{\vec{E}}$-irreducibility requirement implies that no input message of the form $pk(A, M)^-$, $sk(A, M)^-$ or $(W; M)^-$ can appear in a strand of the protocol, where A is a variable of sort Name, W is a variable of sort S s.t. $\mathsf{S} \leq \mathsf{Msg}$, and M is a variable of sort Msg. For instance, M in $pk(A, M)^-$ could be instantiated to $sk(A, M')$ and then $pk(A, sk(A, M'))^-$ is not $\rightarrow_{\vec{E}}$-irreducible. The strands $\mathcal{P}$ associated to the three protocol steps shown in Example 1.1 are as follows, one for each principal (or role) in the protocol:

(s1) $(r : \mathsf{Fresh})\ [n(s, r)^+]$
This strand denotes the server s sending a nonce to a principal.

(s2) $(r' : \mathsf{Fresh})\ [N^-, sk(a, N; s)^+, sk(a, b; (n(a, r'); s))^+]$
This strand denotes principal a receiving a nonce from a principal and then

[6] Note that the reverse equation $(X; Y); Z = X; (Y; Z)$ is not useful as a rule for this example, since it is the term $sk(a, b; (n(a, r); s))$ that has to be converted into $sk(a, (b; n(a, r)); s)$. Moreover, such reverse equation will make the E-unification procedure intractable (i.e., infinitary), since we could apply, by narrowing, the rule $X; (Y; Z) \to (X; Y); Z$ and then the rule $(X; Y); Z \to X; (Y; Z)$ infinitely many times.

[7] A *constructor term* is a term built up with only constructor symbols and variables. Given a set of rewrite rules $l_1 \to r_1, \ldots, l_n \to r_n$ over a many-sorted signature Σ, a function symbol $f : s_1 \times \ldots \times s_n \longrightarrow s$ 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 somewhat 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. We illustrate this subtle point with our associativity axiom.

sending two messages encrypted with his private key, one with the received nonce and the server's name s, and another with names b and s, and a new generated nonce.

(s3) $[sk(a, M; s)^-, sk(a, (b; N); s)^-]$
This strand denotes principal b receiving two messages, where he/she looks for name s in the first and names b and s in the second.

The following strands describe the intruder ability to concatenate, deconcatenate, encrypt and decrypt messages according[8] to the Dolev-Yao attacker's capabilities [9]:

(s4) $[M_1^-, M_2^-, (M_1; M_2)^+]$ Concatenation

(s5) $[(M_1; M_2)^-, M_1^+, M_2^+]$ Deconcatenation

(s6) $[M^-, pk(A, M)^+]$ Encryption with public key

(s7) $[M^-, sk(i, M)^+]$ Encryption with private key

As explained above, 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. In this case (and many others) the grammars reduce the infinite search space to a finite one. For instance, by capturing the following infinite backwards narrowing sequence[9] :

$$m \rightsquigarrow (M_1; m) \rightsquigarrow (M_1'; M_1; m) \rightsquigarrow \cdots$$

generated by the Dolev-Yao strand for deconcatenation shown above. We describe the grammar generation below. A more detailed description is given in [10].

3.2 Grammar Generation

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 concrete seed terms for Example 1.1 are as follows:

$$sd_1 \equiv M_1{\notin}\mathcal{I} \mapsto (M_1; M_2){\in}\mathcal{L} \quad sd_2 \equiv M_2{\notin}\mathcal{I} \mapsto (M_1; M_2){\in}\mathcal{L}$$

$$sd_3 \equiv M{\notin}\mathcal{I} \mapsto sk(A, M){\in}\mathcal{L} \quad sd_4 \equiv M{\notin}\mathcal{I} \mapsto pk(A, M){\in}\mathcal{L}$$

They are understood as initial grammars, e.g., the language production $M{\notin}\mathcal{I} \mapsto pk(A, M){\in}\mathcal{L}$ denotes a formal language including any message

[8] Note that we have simplified the intruder rules w.r.t. [9]; see [10, Example 3] for further details.

[9] For the ease of reading, we deliberately omit the rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, R_{\mathcal{P}}, E_{\mathcal{P}})$ for strands used in each narrowing step $\rightsquigarrow_{R_{\mathcal{P}}, E_{\mathcal{P}}}$ during the reachability stage; see [10] for further details. We also omit the extra strand information and just write the relevant messages to be learned by the intruder, i.e., all the messages appearing in strands as input messages m^-.

$pk(t_1, t_2)$ such that subterm t_1 is of sort Name and the intruder does not know yet the subterm t_2 of sort Msg. The Maude-NPA strategy for generating languages involves three stages.

Term Generation. In this stage, the Maude-NPA takes each term defined by a language production and finds, by backwards narrowing, a complete set S of paths to the state in which the intruder knows that term, where by "complete" we mean that any path from an initial state to that term must contain a path from S as a subpath. For example, with respect to seed terms sd_1 and sd_2, we have the following backwards narrowing steps[10], where we indicate the strand involved in such step, variables M_1, M_1', M_2, M_2' are of sort Msg, and variables X, Y, Z are of sort Elm:

$$\begin{array}{cc}
(M_1; M_2) \rightsquigarrow_{id,s4} M_1, M_2 & (M_1; M_2) \rightsquigarrow_{[M_1/(X;Y), M_2/Z], s4} X, (Y;Z) \\
(M_1; M_2) \rightsquigarrow_{id,s5} M_1'; (M_1; M_2) & (M_1; M_2) \rightsquigarrow_{id,s5} (M_1; M_2); M_2' \\
(M_1; M_2) \rightsquigarrow_{id,s7} pk(i, (M_1; M_2)) & (M_1; M_2) \rightsquigarrow_{id,s6} sk(A, (M_1; M_2))
\end{array}$$

Intuitively, the step $(M_1; M_2) \rightsquigarrow_{id,s4} M_1, M_2$ implies that in order for the intruder to learn message $(M_1; M_2)$, he/she needs to have learned messages M_1 and M_2 in a previous state of the protocol. And the step[11] $(M_1; M_2) \rightsquigarrow_{[M_1/(X;Y), M_2/Z], s4} X, (Y;Z)$ implies that the intruder must learn messages X and $(Y;Z)$ in order to send message $((X;Y);Z)$ to a principal.

Rule Verification. In this stage, the Maude-NPA examines each path and determines which paths are already captured by the grammars, i.e., which paths already require the intruder to know a member of the language in order to produce the goal. The tool removes those paths from consideration.

For example, with respect to seed term $sd_1 \equiv M_1 {\notin} \mathcal{I} \mapsto (M_1; M_2) {\in} \mathcal{L}$ and the backwards narrowing step $(M_1; M_2) \rightsquigarrow_{id,s4} M_1, M_2$, we have a contradiction, since the intruder must learn message M_1 but we have a constraint saying that the intruder does not know M_1 (i.e., $M_1 {\notin} \mathcal{I}$). We have a similar contradiction for seed term $sd_2 \equiv M_2 {\notin} \mathcal{I} \mapsto (M_1; M_2) {\in} \mathcal{L}$, the step $(M_1; M_2) \rightsquigarrow_{id,s4} M_1, M_2$, and the message M_2. For the step $(M_1; M_2) \rightsquigarrow_{[M_1/(X;Y), M_2/Z], s4} X, (Y;Z)$, however, the term $(Y;Z)$ belongs to the language of the seed term sd_2 and thus, we remove it. That is, $Z {\notin} \mathcal{I}$ implies $(Y;Z) {\in} \mathcal{L}$ by using the language production $M_2 {\notin} \mathcal{I} \mapsto (M_1; M_2) {\in} \mathcal{L}$. For the remaining backwards narrowing steps, none is captured by the formal languages associated to sd_1 or sd_2, respectively, and thus none is dicarded.

[10] Again, for ease of reading, we deliberately omit the rewrite theory $\mathcal{R}_{S\mathcal{P}} = (\Sigma_{S\mathcal{P}}, R_{S\mathcal{P}}, E_{S\mathcal{P}})$ used in each narrowing step $\rightsquigarrow_{R_{S\mathcal{P}}, E_{S\mathcal{P}}}$ during the grammar generation. The rewrite theory $\mathcal{R}_{S\mathcal{P}}$ is a simplification of the rewrite theory $\mathcal{R}_{\mathcal{P}}$ for the grammar generation stage and is automatically obtained from it; see [10] for further details.

[11] One could expect the step $(M_1; M_2) \rightsquigarrow_{[M_1/(X;Y), M_2/Z], s4} (X;Y), Z$ instead of $(M_1; M_2) \rightsquigarrow_{[M_1/(X;Y), M_2/Z], s4} X, (Y;Z)$. However, the E-unification problem considered here is $(M_1; M_2) =_E (M_1'; M_2')$, which arises when trying to apply the rule $(M_1'; M_2') \to M_1', M_2'$ (which is obtained from strand $s4$ and stored in $R_{S\mathcal{P}}$) to term $(M_1; M_2)$. The computed E-unifier is $[\ M_1/(X;Y),\ M_2/Z,\ M_1'/X,\ M_2'/(Y;Z)\ ]$, which is obtained by applying the rewrite rule $X; (Y;Z) \to (X;Y); Z$ (stored in $\overrightarrow{E}$) to the term $(M_1'; M_2')$.

Intuitively, at this stage, messages X, $(Y;Z)$, $(M_1';(M_1;M_2))$, $((M_1;M_2);M_2')$, $pk(i,(M_1;M_2))$, and $sk(A,(M_1;M_2))$ have not yet been defined to be a member of the language for sd_1, so these paths stay in for the next stage. Similarly, messages $(M_1';(M_1;M_2))$, $((M_1;M_2);M_2')$, $pk(i,(M_1;M_2))$, and $sk(A,(M_1;M_2))$ stay in for seed term sd_2.

Rule Generation. In the *rule generation* stage, the Maude-NPA looks at the remaining paths, and generates a new set of grammar rules according to a set of heuristics. We have four heuristics $H1, H2a, H2b, H3$ that are used according to two global strategies $S1$ and $S2$.

($H1$) This heuristic extends the language and, essentially, looks for terms that could be captured by it but are not. For example, with respect to the backwards narrowing step $(M_1;M_2) \leadsto_{id,s5} M_1';(M_1;M_2)$, we have that message $(M_1';(M_1;M_2))$ is learned by the intruder from message $(M_1;M_2)$ and, since $(M_1;M_2)$ is already in the language for sd_1 and is a proper subterm of $(M_1';(M_1;M_2))$, we add a grammar rule $Y{\in}\mathcal{L} \mapsto (M_1';Y){\in}\mathcal{L}$ that allows the new grammar to capture message $(M_1';(M_1;M_2))$. Similarly, we generate the grammar rule $Y{\in}\mathcal{L} \mapsto (Y;M_2'){\in}\mathcal{L}$ from the narrowing step $(M_1;M_2) \leadsto_{id,s5} (M_1;M_2);M_2'$, the grammar rule $Y{\in}\mathcal{L} \mapsto pk(i,Y){\in}\mathcal{L}$ from the narrowing step $(M_1;M_2) \leadsto_{id,s7} pk(i,(M_1;M_2))$, and the grammar rule $Y{\in}\mathcal{L} \mapsto sk(A,Y){\in}\mathcal{L}$ from the narrowing step $(M_1;M_2) \leadsto_{id,s6} sk(A,(M_1;M_2))$.

($H2a$) This heuristic restricts the language. It detects that there is a constraint $u{\notin}\mathcal{I}$ that has been instantiated by the computed E-unifier. This implies that the intruder might learn some partial data (symbols introduced by unification) and thus such partial data should be excluded from the language. For the narrowing step $(M_1;M_2) \leadsto_{[M_1/(X;Y),M_2/Z],s4} X,(Y;Z)$, the constraint $M_1{\in}\mathcal{I}$ is instantiated and therefore we introduce the constraint $M_1{\not\leq}(X;Y)$, where X, Y are variables of sort Elm.

($H2b$) This heuristic detects also that some partial data might be learned by the intruder and also excludes them from the language. For the narrowing step $(M_1;M_2) \leadsto_{[M_1/(X;Y),M_2/Z],s4} X,(Y;Z)$, we introduce the constraint $(M_1;M_2){\not\leq}((X;Y);Z)$, where X, Y, Z are variables of sort Elm. The use of $H2a$ and $H2b$ is explained below.

($H3$) This heuristic extends the current grammar G but using constraints $u{\notin}\mathcal{I}$. For example, with respect to the step $(M_1;M_2) \leadsto_{id,s7} pk(i,(M_1;M_2))$ and the seed term sd_1, we have the constraint $M_1{\notin}\mathcal{I}$. Since M_1 is unknown by the intruder and is a proper subterm of the message $pk(i,(M_1;M_2))$, then $pk(i,(M_1;M_2))$ must be unknown by the intruder. Therefore, we would add a grammar rule $Y{\notin}\mathcal{I} \mapsto pk(i,(Y;M_2)){\in}\mathcal{L}$; note that we do not really include such language production in the final grammar, since the application of heuristic $H1$ precludes heuristic $H3$.

These heuristics are applied following one of the following global strategies $S1$ or $S2$. Apply heuristics in the following order for strategy $S1$: try $H1$, if it fails try $H2a$, if it fails try $H3$, otherwise stop with failure the whole grammar generation process for this grammar. Strategy $S2$ is similar to $S1$, but tries $H2b$ instead of

$H2a$. The difference between strategies $S2$ and $S1$ (i.e., between heuristics $H2b$ and $H2a$) is that $S2$ generates more restricted grammars, thus cutting less terms and providing bigger formal languages, than strategy $S1$. However, strategy $S1$ can be applied in fewer situations than strategy $S2$, because of the $\_\notin\mathcal{I}$ constraint for heuristic $H2a$. Therefore, for a seed term of the form $\emptyset \mapsto t\in\mathcal{L}$ only strategy $S2$ can be applied, whereas for a seed term of the form $t|_q\notin\mathcal{I} \mapsto t\in\mathcal{L}$ it is usually better to start with strategy $S2$ and if it fails, then try strategy $S1$.

After the rule generation stage, the Maude-NPA reiterates the three stages until either all paths are eliminated in the rule verification stage, in which case it has successfully defined a language, or it can define no new language rules in the rule generation stage, in which case it has failed to define a language. It can also conceivably fail to terminate adding new rules forever. See [10] for examples of grammars that fail to define a language or fail to terminate.

The tool generates the following grammars $G^!_{sd_1}$, $G^!_{sd_2}$, $G^!_{sd_3}$, and $G^!_{sd_4}$, respectively, using strategy $S1$:

(g1.1) $M\in\mathcal{L} \mapsto pk(i,M)\in\mathcal{L}$

(g1.2) $M\in\mathcal{L} \mapsto sk(A,M)\in\mathcal{L}$

(g1.3) $M_2\in\mathcal{L} \mapsto M_1;M_2\in\mathcal{L}$

(g1.4) $M_1\in\mathcal{L} \mapsto M_1;M_2\in\mathcal{L}$

(g1.5) $M_1\notin\mathcal{I},\ M_1\not\leq n(a,r),$
$M_1\not\leq(b;n(a,r')) \ \mapsto\ M_1;M_2\in\mathcal{L}$

(g2.1) $M\in\mathcal{L} \mapsto pk(i,M)\in\mathcal{L}$

(g2.2) $M\in\mathcal{L} \mapsto sk(A,M)\in\mathcal{L}$

(g2.3) $M_2\in\mathcal{L} \mapsto M_1;M_2\in\mathcal{L}$

(g2.4) $M_1\in\mathcal{L} \mapsto M_1;M_2\in\mathcal{L}$

(g2.5) $M_2\notin\mathcal{I},\ M_2\not\leq n(a,r),$
$M_2\not\leq(n(a,r');s) \ \mapsto\ M_1;M_2\in\mathcal{L}$

(g3.1) $M\in\mathcal{L} \mapsto pk(i,M)\in\mathcal{L}$

(g3.2) $M\in\mathcal{L} \mapsto sk(A,M)\in\mathcal{L}$

(g3.3) $M_2\in\mathcal{L} \mapsto M_1;M_2\in\mathcal{L}$

(g3.4) $M_1\in\mathcal{L} \mapsto M_1;M_2\in\mathcal{L}$

(g3.5) $M\notin\mathcal{I},\ M\not\leq((b;n(a,r));s),$
$M\not\leq(N;s) \ \mapsto\ sk(A,M)\in\mathcal{L}$

(g4.1) $M\in\mathcal{L} \mapsto pk(i,M)\in\mathcal{L}$

(g4.2) $M\in\mathcal{L} \mapsto sk(A,M)\in\mathcal{L}$

(g4.3) $M_2\in\mathcal{L} \mapsto M_1;M_2\in\mathcal{L}$

(g4.4) $M_1\in\mathcal{L} \mapsto M_1;M_2\in\mathcal{L}$

(g4.5) $M\notin\mathcal{I} \mapsto pk(A,M)\in\mathcal{L}$

For instance, grammar rule g3.5 says that a message $sk(A,M)$ is in the language of the grammar for seed term sd_3, if the subterm M is not known by the intruder at the current state of the protocol run (i.e., the fact $M\notin\mathcal{I}$ appears in the intruder knowledge of such state), and the message M is not of the form $(b;(n(a,r));s$ nor $N;s$. Note that in this case (and many others) the grammars reduce the infinite search space to a finite one. For instance, capturing the following infinite backwards narrowing sequence

$$sk(a,(b;N);s) \rightsquigarrow (M_1;sk(a,(b;N);s)) \rightsquigarrow (M_1';M_1;sk(a,(b;N);s)) \rightsquigarrow \cdots$$

since expression $(M_1;sk(a,(b;N);s))$ belongs to the language for seed term sd_2. That is, $(M_1;sk(a,(b;N);s))$ is captured by grammar production g2.5, since the fact $sk(a,(b;N);s)\notin\mathcal{I}$ is introduced into the intruder knowledge after the backwards narrowing step $sk(a,(b;M);s) \rightsquigarrow (M_1;sk(a,(b;M);s))$.

3.3 Backwards Reachability Analysis

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

$$[\ sk(a,M;s)^-,\ sk(a,(b;N);s))^-\ |\ nil\ ]\ \&\ \{\ K\ \}$$

where the strand is at its final position (i.e., every message is in the past), M is a variable of sort Msg, N is a variable of sort Nonce, and K is a variable representing the intruder knowledge. Note that M, being of sort Msg instead of sort Nonce, is the key fact for the confusion attack, see Footnote 5. The intruder knowledge is essential in the backwards reachability process and thus variable K will be appropriately instantiated; see [10] for details. Our tool is able to find the following initial state of the protocol:

$$\begin{array}{l}
[\ nil\ |\ n(s,r)^+\ ]\ \&\ [\ nil\ |\ n(s,r')^+\ ]\ \& \\
[\ nil\ |\ n(s,r)^-,\ (sk(a,n(s,r);s))^+,\ (sk(a,b;(n(a,r'');s)))^+\ ]\ \& \\
[\ nil\ |\ n(s,r')^-,(sk(a,n(s,r');s))^+,(sk(a,b;(n(a,r''');s)))^+\ ]\ \& \\
[\ nil\ |\ (sk(a,(b;n(a,r''));s))^-,(sk(a,(b;n(a,r'''));s))^-\ ]\ \& \\
\{\ n(s,r)\notin\mathcal{I},\ n(s,r')\notin\mathcal{I},\ sk(a,(b;n(a,r''));s)\notin\mathcal{I},\ sk(a,(b;n(a,r'''));s)\notin\mathcal{I}\ \}
\end{array}$$

where r,r',r'',r''' are variables of sort Fresh, all the strands are in their initial position (i.e., every message is in the future), and the intruder does not know but it will learn four messages exchanged in the protocol run. Note that the messages $sk(a,n(s,r);s)$ and $sk(a,n(s,r');s)$ are of no interest to the intruder. The concrete message exchange sequence is:

$$\begin{array}{l}
n(s,r')^+\ .\ n(s,r')^-\ .\ sk(a,n(s,r');s)^+\ .\ sk(a,b;(n(a,r''');s))^+\ .\ n(s,r)^+\ . \\
n(s,r)^-\ .\ sk(a,n(s,r);s)^+\ .\ sk(a,b;(n(a,r'');s))^+\ .\ sk(a,(b;n(a,r''));s)^-\ . \\
sk(a,(b;n(a,r'''));s)^-
\end{array}$$

4 Conclusions

We have presented a high-level overview of the rewriting-based formalization of the NPA reachability analysis and language generation mechanisms that we call the Maude-NPA. And we have illustrated its use, in contexts beyond the Dolev-Yao perfect cryptography or bounded number of sessions assumptions, by means of a protocol with an attack that can only be found by taking into account equational properties of the underlying cryptographic infrastructure: in our example the associativity of the message concatenation operator. The Maude-NPA prototype has been used for this example, both to produce the grammars and to find the attack.

As pointed out in the Introduction, this work is a first step within a longer-term research project to use NPA-like mechanisms in the analysis of protocols in which attacks may make use of the algebraic properties of underlying cryptographic functions. For example, we are working on extending the Maude-NPA with narrowing capabilities *modulo* equational theories for which finitary unification algorithms exist, such as, for example, associativity-commutativity, the Boolean theory, and some forms of modular exponentiation [18,14].

References

[1] Armando, A., 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: K. Etessami and S. K. Rajamani, editors, *CAV*, Lecture Notes in Computer Science **3576** (2005), pp. 281–285.

[2] Baader, F. and W. Snyder, *Unification theory*, in: A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, Volume **1**, Elsevier Science, 2001 pp. 445–532.

[3] Basin, D. A., S. Mödersheim and L. Viganò, *Algebraic intruder deductions.*, in: G. Sutcliffe and A. Voronkov, editors, *LPAR*, Lecture Notes in Computer Science **3835** (2005), pp. 549–564.

[4] Baugher, M., B. Weis, T. Hardjono and H. Harney, *The group domain of intepretation*, in: *Internet Engineering Task Force, RFC 3547*, 2003 .

[5] Blanchet, B., *An efficient cryptographic protocol verifier based on prolog rules.*, in: *Proceedings of the IEEE Computer Security Foundations Workshop (CSFW-14)* (2001), pp. 82–96.

[6] Clavel, M., F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer and J. Quesada, *Maude: specification and programming in rewriting logic*, Theoretical Computer Science **285** (2002), pp. 187–243.

[7] Cremers, C., "Scyther - Semantics and Verification of Security Protocols," Ph.D. dissertation, Eindhoven University of Technology (2006).

[8] Dershowitz, N., S. Mitra and G. Sivakumar, *Decidable matching for convergent systems (preliminary version).*, in: D. Kapur, editor, *11th International Conference on Automated Deduction (CADE-11)*, Lecture Notes in Computer Science **607** (1992), pp. 589–602.

[9] Dolev, D. and A. Yao, *On the security of public key protocols*, IEEE Transaction on Information Theory **29** (1983), pp. 198–208.

[10] Escobar, S., C. Meadows and J. Meseguer, *A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties*, Theoretical Computer Science **367** (2006), pp. 162–202.

[11] Fabrega, F. J. T., J. C. Herzog and J. Guttman, *Strand spaces: Proving security protocols correct*, Journal of Computer Security **7** (1999), pp. 191–230.

[12] Heather, J., G. Lowe and S. Schneider, *How to prevent type flaw attacks on security protocols*, in: *Proceedings of the 13th IEEE Computer Security Foundations Workshop* (2000), pp. 255–268.

[13] Hullot, J., *Canonical forms and unification*, in: W. Bibel and R. Kowalski, editors, *5^{th} Conference on Automated Deduction*, Lecture Notes in Computer Science **87** (1980), pp. 318–334.

[14] Kapur, D., P. Narendran and L. Wang, *An E-unification algorithm for analyzing protocols that use modular exponentiation*, Technical Report TR-03-2, CS Dept. SUNY Albany (2003).

[15] Meadows, C., *Applying formal methods to the analysis of a key management protocol*, Journal of Computer Security **1** (1992).

[16] Meadows, C., *The NRL Protocol Analyzer: An overview*, The Journal of Logic Programming **26** (1996), pp. 113–131.

[17] Meadows, C., I. Cervesato and P. Syverson, *Specification of the Group Domain of Interpretation Protocol using NPATRL and the NRL Protocol Analyzer*, Journal of Computer Security **12** (2004), pp. 893–932.

[18] Meadows, C. 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.

[19] Meseguer, J., *Conditional rewriting logic as a unified model of concurrency*, Theoretical Computer Science **96** (1992), pp. 73–155.

[20] Stubblebine, S. and C. Meadows, *Formal characterization and automated analysis of known-pair and chosen-text attacks*, IEEE Journal on Selected Areas in Communications **18** (2000), pp. 571–581.