State Space Reduction in the Maude-NRL Protocol Analyzer

Santiago Escobar[1,⋆], Catherine Meadows[2], and José Meseguer[3]

[1] Universidad Politécnica de Valencia, Spain
sescobar@dsic.upv.es
[2] Naval Research Laboratory, Washington, DC, USA
meadows@itd.nrl.navy.mil
[3] 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 supported equational reasoning in a more limited way. Maude-NPA supports a wide variety of algebraic properties that includes many crypto-systems of interest such as, for example, one-time pads and Diffie-Hellman. Maude-NPA, like the original NPA, looks for attacks by searching backwards from an insecure attack state, and assumes an unbounded number of sessions. Because of the unbounded number of sessions and the support for different equational theories, it is necessary to develop ways of reducing the search space and avoiding infinite search paths. As a result, we have developed a number of state space reduction techniques. In order for the techniques to prove useful, they need not only to speed up the search, but should not violate soundness so that failure to find attacks still guarantees security. In this paper we describe the state space reduction techniques we use. We also provide soundness proofs, and experimental evaluations of their effect on the performance of Maude-NPA.

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. The tool handles searches in the unbounded session model, and thus can be used to provide proofs of security as well as to search for attacks. It is the next generation of the NRL Protocol Analyzer [11], a tool that supported limited equational reasoning and was successfully applied to the analysis of many different protocols. In Maude-NPA we improve on the original NPA in three ways. First of all, unlike NPA, which required considerable interaction with the user,

⋆ S. Escobar has been partially supported by the EU (FEDER) and the Spanish MEC, under grant TIN 2007-68093-C02, and Integrated Action HA 2006-0007.

S. Jajodia, and J. Lopez (Eds.): ESORICS 2008, LNCS 5283, pp. 548–562, 2008.

Maude-NPA is completely automated. Secondly, its inference system has a formal basis in terms of rewriting logic and narrowing, which allows us to provide proofs of soundness and completeness [7]. Finally, the tool's inference system supports reasoning modulo the algebraic properties of cryptographic and other functions. Such algebraic properties are expressed as equational theories whose equations are confluent, coherent, and terminating modulo equational axioms such as commutativity (C), associativity-commutativity (AC), or associativity-commutativity plus identity (ACU) of some function symbols [6]. The Maude-NPA has then both dedicated and generic methods for solving unification problems in such theories [5,4], which under appropriate checkable conditions yield finitary unification algorithms [4].

Since Maude-NPA allows reasoning in the unbounded session model, and because it allows reasoning about different equational theories (which typically generate many more solutions to unification problems than syntactic unification, leading to bigger state spaces), it is necessary to find ways of pruning the search space in order to prevent infinite or overwhelmingly large search spaces. One technique for preventing infinite searches is the generation of formal grammars describing terms unreachable by the intruder described in [11,7]. However, grammars do not prune out all infinite searches, and there is a need for other techniques. Moreover, even when a search space is finite it may still be necessary to reduce it to a manageable size, and state space reduction techniques for doing that will be necessary. In this paper we describe some of the major state space reduction techniques that we have recently implemented in Maude-NPA, and provide soundness proofs and experimental evaluations demonstrating an average state-space size reduction of 96% (i.e., the average size of the reduced state space is 4% of that of the original one) in the examples we have evaluated. Furthermore, we show our combined techniques effective in obtaining a *finite* state space for all protocols in our experiments.

We first describe the model of computation used by the Maude-NPA and how we obtain a first state-space reduction by reducing the number of variables present in a state. Also, we briefly describe how automatically generated grammars provide a second reduction that cuts down the search space. The additional state space reduction techniques presented in this paper are: (i) giving priority to input messages in strands, (ii) early detection of inconsistent states (never reaching an initial state), (iii) a relation of transition subsumption (to discard transitions and states already being processed in another part of the search space), and (iv) the super lazy intruder (to delay the generation of substitution instances as much as possible). The rest of the paper is organized as follows. After some preliminaries in Section 2, we describe in Section 3 how Maude-NPA works. In Section 4, after a brief overview of the way grammars are used, we describe the various state space reduction techniques that have been introduced to control state explosion, and give proofs of their soundness. We also show their relations to other optimization techniques in the literature. In Section 5 we describe out our experimental evaluation of the state-space reduction techniques. In Section 6 we describe future work and conclude the paper.

2 Preliminaries

We follow the classical notation and terminology from [16] for term rewriting and from [12,13] for rewriting logic and order-sorted notions. We assume an *order-sorted signature* Σ with a finite poset of sorts $(\mathsf{S}, \leq)$ and a finite number of function symbols. We assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_\mathsf{s}$ countably infinite. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ is the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort s. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding term algebras. We write $\mathcal{V}ar(t)$ for the set of variables present in a term t. The set of positions of a term t is written $Pos(t)$, and the set of non-variable positions $Pos_\Sigma(t)$. The root of a term is Λ. The subterm of t at position p is $t|_p$, and $t[u]_p$ is result of replacing $t|_p$ by u in t. A *substitution* σ is a sort-preserving mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_\Sigma(\mathcal{X})$. The identity substitution is *id*. Substitutions are homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$. The restriction of σ to a set of variables V is $\sigma|_V$. The composition of two substitutions is $(\sigma \circ \theta)(X) = \theta(\sigma(X))$ for $X \in \mathcal{X}$.

A *Σ-equation* is an unoriented pair $t = t'$, where $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. Given Σ and a set E of Σ-equations such that $\mathcal{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ for every sort s, order-sorted equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$ (see [13]). Throughout this paper we assume that $\mathcal{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ for every sort s. An *E-unifier* for a Σ-equation $t = t'$ is a substitution σ s.t. $\sigma(t) =_E \sigma(t')$. A *complete* set of E-unifiers of an equation $t = t'$ is written $CSU_E(t = t')$. We say $CSU_E(t = t')$ is *finitary* if it contains a finite number of E-unifiers.

A *rewrite rule* is an oriented pair $l \to r$, where $l \notin \mathcal{X}$ and $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *(unconditional) order-sorted rewrite theory* is a triple $\mathcal{R} = (\Sigma, E, R)$ with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules. A *topmost rewrite theory* is a rewrite theory s.t. for each $l \to r \in R$, $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{State}}$ for a top sort State, $r \notin \mathcal{X}$, and no operator in Σ has State as an argument sort. The rewriting relation $\to_R$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $t \overset{p}{\to}_R t'$ (or $\to_R$) if $p \in Pos_\Sigma(t)$, $l \to r \in R$, $t|_p = \sigma(l)$, and $t' = t[\sigma(r)]_p$ for some σ. The rewriting relation $\to_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $t \overset{\Lambda}{\to}_{R,E} t'$ (or $\to_{R,E}$) if $l \to r \in R$, $t =_E \sigma(l)$, and $t' = \sigma(r)$.

The narrowing relation $\rightsquigarrow_R$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $t \overset{p}{\rightsquigarrow}_{\sigma,R} t'$ (or $\rightsquigarrow_{\sigma,R}$, $\rightsquigarrow_R$) if $p \in Pos_\Sigma(t)$, $l \to r \in R$, $\sigma \in CSU_\emptyset(t|_p = l)$, and $t' = \sigma(t[r]_p)$. Assuming that E has a finitary and complete unification algorithm, the narrowing relation $\rightsquigarrow_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $t \overset{p}{\rightsquigarrow}_{\sigma,R,E} t'$ (or $\rightsquigarrow_{\sigma,R,E}$, $\rightsquigarrow_{R,E}$) if $p \in Pos_\Sigma(t)$, $l \to r \in R$, $\sigma \in CSU_E(t|_p = l)$, and $t' = \sigma(t[r]_p)$.

3 The Maude-NPA's Execution Model

In the Maude-NPA [7], protocols are specified with a notation derived from strand spaces [10]. 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 over time and thus we use the symbol | to divide past and future in a strand, i.e., $[nil, msg_1^{\pm}, \ldots, msg_{j-1}^{\pm} \mid msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}, nil]$ where $msg_1^{\pm}, \ldots, msg_{j-1}^{\pm}$ are the past messages, and $msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}$ are the future messages ($msg_j^{\pm}$ is the immediate future message). The nils are present so that the bar may be placed at the beginning or end of the strand if necessary. A strand $[msg_1^{\pm}, \ldots, msg_k^{\pm}]$ is a shorthand for $[nil \mid msg_1^{\pm}, \ldots, msg_k^{\pm}, nil]$ and we often remove the nils for clarity. We write $\mathcal{P}$ for the set of strands in a protocol.

A *state* is a set of Maude-NPA strands unioned together with an associative and commutativity union operator _&_ with identity operator $\emptyset$, along with an additional term describing the intruder knowledge at that point. The *intruder knowledge* is represented as a set of facts unioned together with an associative and commutativity union operator _,_ with identity operator $\emptyset$ and wrapped by a function symbol $\{\_\}$ as a state component. There are two kinds of intruder facts: positive knowledge facts (the intruder knows m, i.e., $m \in \mathcal{I}$), and negative knowledge facts (the intruder does not yet know m but will know it in a future state, i.e., $m \notin \mathcal{I}$), where m is a message expression.

Strands communicate between them via a unique shared channel, i.e., by sending messages to the channel and retrieving messages from the channel. However, we do not explicitly represent the channel in our model. Instead, since the intruder is able to learn any message present in the channel, we use the intruder knowledge as the channel. When the intruder observes a message in the channel, then it learns it, i.e., a message m is learned in a transition (in a forward execution of the protocol) 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. The intruder has the usual ability to read and redirect traffic, and can also perform operations, e.g., encryption, decryption, concatenation, exclusive or, exponentiation, etc., on messages that it has received; the nature and algebraic properties of such operations depend on the given cryptographic theory $E_{\mathcal{P}}$. 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 on the number of principals, number of sessions, nonces, or time, i.e., no data abstraction or approximation is performed.

The user can make use of a special sort Fresh in the protocol-specific signature Σ for representing fresh unguessable values, e.g., for nonces. 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. We make the Fresh variables generated by a strand explicit 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 generated by $msg_1^{\pm}, \ldots, msg_n^{\pm}$.

The types and algebraic properties of the operators used in messages (cryptographic and otherwise) are described as an equational theory $E_{\mathcal{P}}$.

Example 1. [6] The Diffie-Hellman protocol uses exponentiation to achieve authentication between two parties, A and B. The informal textbook-level protocol description proceeds as follows.

$$1.\ A \to B : \{A, B, g^{N_A}\} \quad 2.\ B \to A : \{B, A, g^{N_B}\} \quad 3.\ A \to B : \{secret\}_{(g^{N_B})^{N_A}}$$

In the Maude-NPA formalization of the protocol, we explicitly specify the signature Σ describing messages, nonces, etc. A nonce N_A is denoted by $n(A, r)$, where r is a unique variable of sort Fresh. Concatenation of two messages, e.g., N_A and N_B, is denoted by the operator _;_, e.g., $n(A, r)\ ;\ n(B, r')$. Encryption of a message M is denoted by $e(A, M)$, e.g., $\{N_B\}_{K_B}$ is denoted by $e(K_B, n(B, r'))$. Decryption is similarly denoted by $d(A, M)$. Raising a message M to the power of an exponent E (i.e., M^E) is denoted by $exp(M_1, M_2)$, e.g., g^{N_B} is denoted by $exp(g, n(B, r'))$. Associative-commutative multiplication on nonces is denoted by _*_. A secret generated by a principal is denoted by $sec(A, r)$, where r is a unique variable of sort Fresh. The protocol-specific signature Σ is as follows (Maude-NPA expects a sort Msg denoting messages in the protocol specification):

$$\begin{array}{rr} a, b, i : \to \mathsf{Name} & e, d : \mathsf{Key} \times \mathsf{Msg} \to \mathsf{Enc} \\ n : \mathsf{Name} \times \mathsf{Fresh} \to \mathsf{Nonce} & \_;\_ : \mathsf{Msg} \times \mathsf{Msg} \to \mathsf{Msg} \quad g :\to \mathsf{Gen} \\ exp : \mathsf{GenvExp} \times \mathsf{NonceSet} \to \mathsf{Exp} & \_*\_ : \mathsf{NonceSet} \times \mathsf{NonceSet} \to \mathsf{NonceSet} \end{array}$$

together with the following subsort relations $\mathsf{Name}, \mathsf{Nonce}, \mathsf{Enc}, \mathsf{Exp} < \mathsf{Msg}$, $\mathsf{Nonce} < \mathsf{NonceSet}$, and $\mathsf{Gen}, \mathsf{Exp} < \mathsf{GenvExp}$. In the following we will use letters A, B for variables of sort Name, letters r, r', r'' for variables of sort Fresh, and letters M, M_1, M_2, Z for variables of sort Msg; whereas letters X, Y will also represent variables, but their sort will depend on the concrete position in a term. The encryption/decryption cancellation properties are described using the equations $e(X, d(X, Z)) = Z$ and $d(X, e(X, Z)) = Z$ in $E_{\mathcal{P}}$. The key algebraic property on exponentiations $z^{x^y} = z^{x*y}$ is described using the equation $exp(exp(W, Y), Z) = exp(W, Y * Z)$ in $E_{\mathcal{P}}$ (where W is of sort Gen instead of the more general sort GenvExp in order to provide a finitary narrowing-based unification procedure modulo $E_{\mathcal{P}}$, see [6]). Although multiplication modulo a prime number has a unit and inverses, we have only included the algebraic property that is necessary for Diffie-Hellman to work. The two strands $\mathcal{P}$ associated to the three protocol steps shown above are:

(s1) $(r, r'{:}\mathsf{Fresh})[\ (A; B; exp(g, n(A, r)))^+, (B; A; X)^-, (e(exp(X, n(A, r)), sec(A, r')))^+]$
(s2) $(r''{:}\mathsf{Fresh})[\ (A; B; Y)^-,\ (B; A; exp(g, n(B, r'')))^+,\ (e(exp(Y, n(B, r'')), SR)^-]$

The following strands describe the intruder abilities according to the Dolev-Yao attacker's capabilities [3]. Note that the intruder cannot extract information from either an exponentiation or a product of exponents, only compose them.

(s3) $[nil\ |M_1^-, M_2^-, (M_1 * M_2)^+,\ nil\ ]$ Multiplication
(s4) $[nil\ |M_1^-, M_2^-, exp(M_1, M_2)^+, nil\ ]$ Exponentiation

(s5) $[nil\ |g^+,\ nil\ ]$ Generator

(s6) $[nil\ |A^+,\ nil\ ]$ All names are public

(s7) $(nil,\ r''' : \mathsf{Fresh})\ [nil|\ n(i,r''')^+\ , nil\ ]$ Generation of its own nonces

3.1 Backwards Reachability Analysis

In order to understand many of the optimizations described in this paper, it is important to know the execution rules in the Maude-NPA; see [7] for further details. In principle, these are represented by the following rewrite rules[1] (we use letters L, L_1, L_2 for variables of sort SMsgList, letters K, K' for variables of sort Knowledge, and letters SS, SS' for variables of sort StrandSet):

$$\begin{aligned}
\mathbb{R} = \{\ & SS\,\&\,[L \mid M^-, L']\,\&\,\{M{\in}\mathcal{I}, K\} \rightarrow SS\,\&\,[L, M^- \mid L']\,\&\,\{M{\in}\mathcal{I}, K\}, && (1)\\
& SS\,\&\,[L \mid M^+, L']\,\&\,\{K\} \rightarrow SS\,\&\,[L, M^+ \mid L']\,\&\,\{K\}, && (2)\\
& SS\,\&\,[L \mid M^+, L']\,\&\,\{M{\notin}\mathcal{I}, K\} \rightarrow SS\,\&\,[L, M^+ \mid L']\,\&\,\{M{\in}\mathcal{I}, K\}\ \} && (3)
\end{aligned}$$

Rule (1) synchronizes an input message with a message already learned by the intruder, Rule (2) accepts output messages but the intruder's knowledge is not increased, and Rule (3) accepts output messages but the intruder's knowledge is positively increased.

In a backwards execution of the protocol using narrowing, which is the one we are interested in, we start from an attack state, i.e., a term with variables, containing (i) some of the strands of the protocol with the bar at the end, e.g., Strands (s1) and (s2) for Example 1, (ii) some terms the intruder knows at the attack state, i.e., of the form $t{\in}\mathcal{I}$, (iii) a variable SS denoting a set of strands, and (iv) a variable IK denoting a set of intruder facts. We then perform narrowing with the Rules (1)–(3) in reverse to move the bars of the strands to the left. Note that variables SS and IK will be instantiated by narrowing to new strands or new intruder knowledge in order to find an initial state.

However, in an intermediate state we can have many partially executed strands together with many intruder strands from the Dolev-Yao attacker's capabilities, i.e., Strands (s3)–(s7). Thus, an initial or attack state in our tool may involve an unbounded number of strands, which would be unfeasible. To avoid this problem, while still supporting a complete formal analysis for an unbounded number of sessions, we can use a more perspicuous set of rewrite rules describing the protocol, where the necessary additional strands are introduced dynamically. The key idea is to specialize Rule (3) using the different protocol strands; see [7] for further details:

$$R_{\mathcal{P}} = \mathbb{R} \cup \{\ [\,l_1 \,|\, u^+,\, l_2\,]\,\&\,\{u{\notin}\mathcal{I}, K\} \rightarrow \{u{\in}\mathcal{I}, K\} \text{ s.t. } [\,l_1,\, u^+,\, l_2\,] \in \mathcal{P}\} \qquad (4)$$

[1] The top level structure of the state is a multiset of strands formed with the _&_ union operator. The protocol and intruder rewrite rules are "essentially topmost" in that, using an extension variable matching the "remaining strands" they can always rewrite the whole state. Therefore, as explained in [14], completeness results of narrowing for topmost theories also applies to them.

Example 2. (Example 1 continued) The attack state we are looking for is one in which Bob completes the protocol and the intruder is able to learn the secret. The attack state pattern to be given as input to the system is:

$$(r'{:}\mathsf{Fresh})[\ (A;B;Y)^{-},(B;A;exp(g,n(B,r')))^{+},(e(exp(Y,n(B,r')),sec(a,r''))^{-}\mid nil\ ]$$
$$\&\ SS\ \&\ \{sec(a,r'')\in\mathcal{I},\ IK\}$$

Using the above attack state pattern our tool is able to find the following initial state of the protocol, showing that the attack state is reachable:

$$[nil \mid exp(g,n(a,r)))^{-},Z^{-},exp(g,Z*n(a,r))^{+}]\ \&$$
$$[nil \mid exp(g,n(b,r')))^{-},W^{-},exp(g,W*n(b,r'))^{+}]\ \&$$
$$[nil \mid exp(g,Z*n(a,r))^{-},e(exp(g,W*n(a,r)),sec(a,r''))^{-},sec(a,r'')^{+}]\ \&$$
$$[nil \mid exp(g,W*n(b,r'))^{-},sec(a,r'')^{-},e(exp(g,W*n(b,r')),sec(a,r''))^{+}]\ \&$$
$$[nil \mid (a;b;exp(g,n(b,r')))^{-},(b;exp(g,n(b,r')))^{+}]\ \&$$
$$[nil \mid (a;A';exp(g,n(a,r)))^{-},(A';exp(g,n(a,r)))^{+}]\ \&$$
$$[nil \mid (b;exp(g,n(b,r')))^{-},exp(g,n(b,r'))^{+}]\ \&$$
$$[nil \mid (A';exp(g,n(a,r)))^{-},exp(g,n(a,r))^{+}]\ \&$$
$$(r' : \mathsf{Fresh})$$
$$[nil \mid (a;b;exp(g,Y))^{-},(a;b;exp(g,n(b,r')))^{+},e(exp(g,W*n(b,r')),sec(a,r''))^{-}]\ \&$$
$$(r'',r : \mathsf{Fresh})$$
$$[nil \mid (a;A';exp(g,n(a,r)))^{+},(a;A';exp(g,Z))^{-},e(exp(g,Z*n(a,r)),sec(a,r''))^{+}]\ \&$$
$$\{\ sec(a,r'')\notin\mathcal{I},e(exp(g,Z*n(a,r)),sec(a,r''))\notin\mathcal{I},e(exp(g,W*n(b,r')),sec(a,r''))\notin\mathcal{I},$$
$$exp(g,n(a,r))\notin\mathcal{I},\quad exp(g,n(b,r'))\notin\mathcal{I},\quad exp(g,Z*n(a,r))\notin\mathcal{I},\quad exp(g,W*n(b,r'))\notin\mathcal{I},$$
$$(a;b;exp(g,n(b,r')))\notin\mathcal{I},\qquad (a;A';exp(g,n(a,r)))\notin\mathcal{I},\qquad (b;exp(g,n(b,r')))\notin\mathcal{I},$$
$$(A';exp(g,n(a,r)))\notin\mathcal{I}\ \}$$

Note that strands not producing Fresh variables are intruder strands, while the two strands producing fresh variables r, r', r'' are protocol strands. The concrete message exchange sequence obtained by the reachability analysis is the following:

$1.(a;b;exp(g,W))^{-}$
$2.(a;b;exp(g,n(b,r')))^{+}$
$3.(a;b;exp(g,n(b,r')))^{-}$
$4.(b;exp(g,n(b,r')))^{+}$
$5.(b;exp(g,n(b,r')))^{-}$
$6.(exp(g,n(b,r')))^{+}$
$7.(exp(g,n(b,r')))^{-}$
$8.W^{-}$
$9.exp(g,W*n(b,r'))^{+}$
$10.(a;A';exp(g,n(a,r)))^{+}$
$11.(a;A';exp(g,n(a,r)))^{-}$
$12.(A';exp(g,n(a,r)))^{+}$
$13.(A';exp(g,n(a,r)))^{-}$
$14.(exp(g,n(a,r)))^{+}$
$15.(exp(g,n(a,r)))^{-}$
$16.Z^{-}$
$17.exp(g,Z*n(a,r))^{+}$
$18.(a;A';exp(g,Z))^{-}$
$19.e(exp(g,Z*n(a,r)),sec(a,r''))^{+}$
$20.e(exp(g,Z*n(a,r)),sec(a,r''))^{-}$
$21.sec(a,r'')^{+}$
$22.exp(g,W*n(b,r'))^{-}$
$23.sec(a,r'')^{-}$
$24.e(exp(g,W*n(b,r'),sec(a,r''))^{+}$
$25.e(exp(g,W*n(b,r')).sec(a,r''))^{-}$

Step 1) describes principal b receiving an initiating message (no corresponding send because of the super-lazy intruder). Step 2) describes b sending the response, and 3) describes the intruder receiving it. Steps 4) through 9) describe the intruder computing the key she will use to communicate with b. Step 10) describes a initiating the protocol with a principal A'. Step 11) describes the intruder receiving it, and steps 11) through 17) describe the intruder constructing the key she will use to communicate with a. Steps 18) and 19) describe a receiving the response from the intruder impersonating A' and a sending the

encrypted message. Steps 20) through 23) describe the intruder decrypting the message to get the secret. In step 24) the intruder re-encrypts the secret in the key she shares with b and sends it, and in step 25) b receives the message.

4 State Space Reduction Techniques

In this section, we describe the different state-reduction techniques identifying unproductive narrowing steps $St \rightsquigarrow_{\sigma, R_{\mathcal{P}}^{-1}, E_{\mathcal{P}}} St'$. There are three reasons for doing this. One is to reduce the initially infinite search space to a finite one, as in the use of grammars. Another is to reduce the size of a (possibly finite) search space by eliminating unreachable states early, i.e., before they are eliminated by exhaustive search. The latter can have an effect far beyond eliminating a single node in the search space, since a single unreachable state could appear multiple times and/or have multiple descendants before being eliminated. Finally, it is also possible to use various partial order reduction techniques.

4.1 Limiting Dynamic Introduction of New Strands

As pointed out in Section 3.1, Rules (4) allow a dynamic introduction of new strands. However, new strands can also be introduced by unification of a state containing a variable SS denoting a set of strands and one of the Rules 1-3, where variables L and L' denoting lists of input/output messages will be introduced by instantiation of SS. The same can happen with new intruder facts of the form $X{\in}\mathcal{I}$, where X is a variable. In order to avoid a huge number of unproductive narrowing steps, we allow the introduction of new strands and/or new intruder facts only by rule application instead of just by instantiation. For this, we do two things: (i) remove any of the following variables from actual states: SS denoting a set of strands, K denoting a set of intruder facts, and L, L' denoting a set of input/output messages; and (ii) replace Rule (1) by the following Rule (5), since we do no longer have a variable denoting a set of intruder facts that has to be instantiated:

$$SS \,\&\, [L \mid M^-, L'] \,\&\, \{M{\in}\mathcal{I}, K\} \rightarrow SS \,\&\, [L, M^- \mid L'] \,\&\, \{K\} \tag{5}$$

Note that in order to replace Rule (1) by Rule (5) we have to assume that the intruder knowledge is a set of intruder facts without repeated elements, i.e., the union operator $\_,\_$ is $ACUI$ (associative-commutative-identity-idempotent). This is completeness-preserving, since it is in line with the restriction in [7] that the intruder learns a term only once; if the intruder needs to use a term twice he must learn it the first time it is needed; if he learns a term and needs to learn it again in the backwards search, the state will be discarded as unreachable. Therefore, the set of rewrite rules used for backwards narrowing are $R_{\mathcal{P}} = \{(5), (2), (3)\} \cup (4)$.

4.2 Grammars

Grammars, unlike the other mechanisms discussed in this paper, appeared in the original Maude-NPA paper [7]. We include a brief discussion here for completeness. In [7], it is shown that Maude-NPA's ability to reason well about low-level

algebraic properties is a result of its combination of symbolic reachability analysis using narrowing, together with its grammar-based techniques for reducing the size of the search space. Here we briefly explain how grammars work as a state space reduction technique and refer the reader to [7] for further details.

Automatically generated grammars $\langle G_1, \ldots, G_m \rangle$ represent unreachability information (or co-invariants), i.e., typically infinite sets of states unreachable for the intruder. That is, given a message m and an automatically generated grammar G, if $m \in G$, then there is no initial state St_{init} and substitution θ such that the intruder knowledge of St_{init} contains the fact $\theta(m) \notin \mathcal{I}$. These automatically generated 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. See [7] for further explanations.

Unlike NPA and the version of Maude-NPA described in [7], in which initial grammars needed to be specified by the user, Maude-NPA now generates initial grammars automatically. Each initial grammar consists of a single seed term of the form $C \mapsto f(X_1, \cdots, X_n) \in \mathcal{L}$, where f is an operator symbol from the protocol specification, the X_i are variables, and C is either empty or consists of the single constraint $X_i \in \mathcal{I}$.

4.3 Partial Order Reduction Giving Priority to Input Messages

The different execution rules are in general executed nondeterministically. This is because the order of execution can make a difference as to what subsequent rules can be executed. For example, an intruder cannot receive a term until it is sent by somebody, and that send within a strand may depend upon other receives in the past. There is one exception, Rule (5) (originally Rule (1)), which, in a backwards search, only moves a negative term appearing right before the bar into the intruder knowledge. The execution of this transition in a backwards search does not disable any other transitions; indeed, it only enables send transitions. Thus, it is safe to execute it at each stage before any other transition. For the same reason, if several applications of Rule 5 are possible, it is safe to execute them all at once before any other transition. Requiring all executions of Rule 5 to execute first thus eliminates interleavings of Rule 5 with send and receive transitions, which are equivalent to the case in which Rule 5 executes first. In practice, this has cut down on the search space size on the order of 50%.

Similar strategies have been employed by other tools in forward searches. For example, in [15], a strategy is introduced that always executes send transitions first whenever they are enabled. Since a send transition does not depend on any other part of the state in order to take place, it can safely be executed first. The original NPA also used this strategy; it had a receive transition which had the effect of adding new terms to the intruder knowledge, and which always was executed before any other transition once it was enabled.

Proposition 1. *Given a topmost rewrite theory* $\mathcal{R} = (\Sigma, E_{\mathcal{P}}, R_{\mathcal{P}})$ *representing protocol* $\mathcal{P}$ *and a state* St*. If* $St \rightsquigarrow_{\sigma_1, R_{\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_1$ *using Rule* (5) *in reverse (thus with* $\sigma_1 = id$*) and* $St \rightsquigarrow_{\sigma_2, R_{\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_2$*, then* $St_1 \rightsquigarrow_{\sigma_2, R_{\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_2$*.*

4.4 Detecting Inconsistent States Early

There are several types of states that are always unreachable or inconsistent. If the Maude-NPA attempts to search beyond them, it will never find an initial state. For this reason, we augment the Maude-NPA search engine to always mark the following types of states as unreachable, and not search beyond them any further:

1. A state St containing two contradictory facts $t{\in}\mathcal{I}$ and $t{\notin}\mathcal{I}$ for a term t.
2. A state St whose intruder knowledge contains the fact $t{\notin}\mathcal{I}$ and a strand of the form $[m_1^{\pm}, \ldots, t^{-}, \ldots, m_{j-1}^{\pm} \mid m_j^{\pm}, \ldots, m_k^{\pm}]$.
3. A state St containing a fact $t{\in}\mathcal{I}$ such that t contains a fresh variable r and the strand in St indexed by r, i.e., $(r_1, \ldots, r, \ldots, r_k : \mathsf{Fresh})$ $[m_1^{\pm}, \ldots, m_{j-1}^{\pm} \mid m_j^{\pm}, \ldots, m_k^{\pm}]$, cannot produce r, i.e., r is not a subterm of any output message in $m_1^{\pm}, \ldots, m_{j-1}^{\pm}$.
4. A state St containing a strand of the form $[m_1^{\pm}, \ldots, t^{-}, \ldots, m_{j-1}^{\pm} \mid m_j^{\pm}, \ldots, m_k^{\pm}]$ for some term t such that t contains a fresh variable r and the strand in St indexed by r cannot produce r.

Note that case 2 will become an instance of case 1 after some backwards narrowing steps, and the same happens with cases 4 and 3. The proof of inconsistency of cases 1 and 3 is obvious and we do not include it here.

4.5 Transition Subsumption

We define here a state relation in the spirit of both partial order reduction techniques (POR) and the folding relations of [9], though a detailed study of the relationship of such a state relation with folding and POR is left for future work.

In the following, we write $IK^{\in}$ (resp. $IK^{\notin}$) to denote the subset of intruder facts of the form $t{\in}\mathcal{I}$ (resp. $t{\notin}\mathcal{I}$) appearing in the set of intruder facts IK. We abuse the set notation and write $IK_1 \subseteq_{E_\mathcal{P}} IK_2$ for IK_1 and IK_2 sets of intruder facts to denote that all the intruder facts of IK_1 appear in IK_2 (modulo $E_\mathcal{P}$).

Definition 1. *Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E_\mathcal{P}, R_\mathcal{P})$ representing protocol $\mathcal{P}$, and given two non-initial states $St_1 = SS_1 \,\&\, \{IK_1\}$ and $St_2 = SS_2 \,\&\, \{IK_2\}$, we write $St_1 \triangleright St_2$ (or $St_2 \triangleleft St_1$) if $IK_1^{\in} \subseteq_{E_\mathcal{P}} IK_2^{\in}$, and for each non-initial strand $[m_1^{\pm}, \ldots, m_{j-1}^{\pm} \mid m_j^{\pm}, \ldots, m_k^{\pm}] \in SS_1$, there exists $[m_1^{\pm}, \ldots, m_{j-1}^{\pm} \mid m_j^{\pm}, \ldots, m_k^{\pm}, m_{k+1}^{\pm}, \ldots, m_{k'}^{\pm}] \in SS_2$. Note that the comparison of the non-initial strand in SS_1 with the strands in SS_2 is performed modulo $E_\mathcal{P}$.*

Definition 2 ($\mathcal{P}$-subsumption relation). *Given a topmost rewrite theory $\mathcal{R} = (\Sigma, E_\mathcal{P}, R_\mathcal{P})$ representing protocol $\mathcal{P}$ and two non-initial states St_1, St_2. We write $St_1 \preceq_\mathcal{P} St_2$ and say that St_1 is* $\mathcal{P}$-subsumed *by St_2 if there is a substitution θ s.t. $St_1 \triangleleft \theta(St_2)$.*

The following result provides the appropriate connection between the transition $\mathcal{P}$-subsumption and narrowing transitions. In the following, $\leadsto^{\{0,1\}}_{\sigma, R_\mathcal{P}^{-1}, E_\mathcal{P}}$ denotes zero or one narrowing steps.

Proposition 2. *Given a topmost rewrite theory* $\mathcal{R} = (\Sigma, E_{\mathcal{P}}, R_{\mathcal{P}})$ *representing protocol* $\mathcal{P}$ *and two non-initial states* St_1, St_2. *If* $St_1 \succeq_{\mathcal{P}} St_2$ *and* $St_2 \rightsquigarrow_{\sigma_2, R_{\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_2'$, *then there is a state* St_1' *and a substitution* σ_1 *such that* $St_1 \rightsquigarrow^{\{0,1\}}_{\sigma_1, R_{\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_1'$ *and* $St_1' \succeq_{\mathcal{P}} St_2'$.

Therefore, we keep all the states of the backwards narrowing-based tree and compare each new leaf of the tree with all the previous states in the tree. If a leaf is $\mathcal{P}$-subsumed by a previously generated node in the tree, we discard such leaf.

4.6 The Super Lazy Intruder

Sometimes terms appear in the intruder knowledge that are trivially learnable by the intruder. These include terms initially available to the intruder (such as names) and variables. In the case of variables, the intruder can substitute any arbitrary term of the same sort as the variable,[2] and so there is no need to try to determine all the ways in which the intruder can do this. For this reason it is safe, at least temporarily, to drop these terms from the state. We will refer to those terms as *lazy intruder* terms. The problem of course, is that later on in the search the variable may become instantiated, in which case the term now becomes relevant to the search. In order to avoid this problem, we take an approach similar to that of the lazy intruder of Basin et al. [1] and extend it to a more general case, that we call the *super-lazy terms*. We note that this use of what we here call the super-lazy intruder was also present in the original NPA.

Super-lazy terms are defined inductively as the union of the set of lazy terms, i.e., variables, with the set of terms that are produced out of other super-lazy terms using operations available to the intruder. That is, $e(K, X)$ is a super-lazy term if the intruder can perform the e operation, and K and X are variables. More precisely, the set of super-lazy intruder terms is defined as follows.

Definition 3. *Given a topmost rewrite theory* $\mathcal{R} = (\Sigma, E_{\mathcal{P}}, R_{\mathcal{P}})$ *representing protocol* $\mathcal{P}$, *and a state* St *where* $IK^{\not\in}(St) = \{x \mid x \not\in \mathcal{I} \in St\}$, *its set of* super-lazy terms w.r.t. St *(or simply super-lazy terms) is defined as the union of the following:*

- *the set of variables of sort* Msg *or one of its subsorts,*
- *the set of terms* t *appearing in strands of the form* $[t^+]$, *and*
- *the set of terms of the form* $f(t_1, \ldots, t_n)$ *where* $\{t_1, \ldots, t_n\}$ *are super-lazy intruder terms w.r.t.* St, $\{t_1, \ldots, t_n\} \not\subseteq IK^{\not\in}(St)$, *and there is an intruder strand* $[(X_1)^-, \ldots, (X_n)^-, (f(X_1, \ldots, X_n))^+]$ *with* $X_1, \ldots, X_n$ *variables.*

[2] This, of course, is subject to the assumption that the intruder can produce at least one term of that sort. But since the intruder is assumed to be a member of the network with access to all the operations available to an honest principal, this is a safe assumption to make.

The idea behind the super-lazy intruder is that, given a term made out of lazy intruder terms, such as $a; e(K, Y)$, where a is a public name and K and Y are variables, the term $a; e(K, Y)$ is also a (super) lazy intruder term by applying the operations e and $\_;\_$.

Let us first briefly explain how the (super) lazy intruder mechanism works before formally describing it. When we detect a state St with a super lazy term t, we replace the intruder fact $t{\in}\mathcal{I}$ in St by a new expression $ghost(t)$ and keep the modified version of St in the history of states used by the transition subsumption of Section 4.5. If later in the search tree we detect a state St' containing an expression $ghost(t)$ such that t is no longer a super lazy intruder term (or *ghost expression*), then t has been instantiated in an appropriate way and we must reactivate the original state St that introduced the $ghost(t)$ expression (and that precedes St' in the narrowing tree) with the new binding for variables in t applied. That is, we "roll back" and replace the current state St' with an instantiated version of state St.

However, if the substitution θ binding variables in t includes variables of sort Fresh, since they are unique in our model, we have to keep them in the reactivated version of St. Therefore, the strands indexed by these fresh variables must be included in the "rolled back" state, even if they were not there originally. Moreover, they must have the bar at the place it was when the strands were originally introduced. We show below how this is accomplished.

Furthermore, if any of the strands thus introduced have other variables of sort Fresh as subterms, then the strands indexed by those variables must be included too, and so on. Thus, when a state St' properly instantiating a ghost expression $ghost(t)$ is found, the procedure of rolling back to the original state St that gave rise to that ghost expression implies not only applying the bindings for the variables of t to St, but also introducing in St all the strands from St' that produced fresh variables and that either appear in the variables of t or are recursively connected to them.

First, before formally defining the super-lazy intruder technique, we must modify Rules 4 introducing new strands:

$$\{\, [\, l_1 \,|\, u^+ \,] \,\&\, \{u{\notin}\mathcal{I}, K\} \rightarrow \{u{\in}\mathcal{I}, K\} \text{ s.t. } [\, l_1,\, u^+,\, l_2 \,] \in \mathcal{P}\} \tag{6}$$

Therefore, the set of rewrite rules used by narrowing in reverse are now $R_{\mathcal{P}} = \{(5), (2), (3)\} \cup (6)$. Note that Rules (4) introduce strands $[\, l_1 \mid u^+, l_2 \,]$, whereas here Rules (6) introduce strands $[\, l_1 \mid u^+ \,]$. This slight modification allows to safely move the position of the bar back to the place where the strand was introduced.

We extend the intruder knowledge to allow an extra fact $ghost(t)$. We first describe how to reactivate a state. Given a strand $s = (r_1, \ldots, r_k : \mathsf{Fresh})\ [m_1^{\pm}, \ldots \mid \ldots, m_n^{\pm}]$, when we want to move the bar to the rightmost position (denoting a final strand), we write $s{\gg} = (r_1, \ldots, r_k : \mathsf{Fresh})\ [m_1^{\pm}, \ldots, m_n^{\pm} \mid nil]$.

Definition 4. *Given a state St containing an intruder fact $ghost(t)$ for some term t with variables, we define the set of strands associated to t, denoted $SS_{St}(t)$, as follows: for each strand s in St of the form $(r_1, \ldots, r_k : \mathsf{Fresh})\ [m_1^{\pm}, \ldots \mid \ldots, m_n^{\pm}]$,*

if there is $i \in \{1,\ldots,k\}$ *s.t.* $r_i \in \mathcal{V}ar(t)$*, then* $s{\gg} \in SS_{St}(t)$*; or if there is another strand* $s' \in SS_{St}(t)$ *of the form* $(r'_1,\ldots,r'_{k'} : \mathsf{Fresh})\ [w_1^{\pm},\ldots \mid \ldots, w_{n'}^{\pm}]$*,* $i \in \{1,\ldots,k\}$*, and* $j \in \{1,\ldots,n'\}$ *s.t.* $r_i \in \mathcal{V}ar(w_j)$*, then* $s{\gg} \in SS_{St}(t)$*.*

Given the previous definition, the following result is immediate.

Proposition 3. *Given a topmost rewrite theory* $\mathcal{R} = (\Sigma, E_{\mathcal{P}}, R_{\mathcal{P}})$ *representing protocol* $\mathcal{P}$ *and a state* St *containing an intruder fact* $t{\in}\mathcal{I}$ *such that* t *is a super-lazy term, let* $\overline{St}$ *denote the state obtained by replacing* $t{\in}\mathcal{I}$ *by ghost*(t)*. Let* St' *be a state such that* $\overline{St} \leadsto^*_{\sigma,R_{\mathcal{P}}^{-1},E_{\mathcal{P}}} St'$*, where* $\sigma(t)$ *is not a super-lazy term, and let* $\sigma' = \sigma|_{\mathcal{V}ar(t)}$*. Let the reactivated state be* $\widehat{St} = \sigma'(St) \cup SS_{St'}(\sigma(t))$*. If there is an initial state* St_{init} *such that* $St \leadsto^*_{\theta,R_{\mathcal{P}}^{-1},E_{\mathcal{P}}} St_{init}$ *and there is a substitution* ρ *such that* $\sigma' \circ \rho =_{E_{\mathcal{P}}} \theta$*, then* $\widehat{St} \leadsto^*_{\rho,R_{\mathcal{P}}^{-1},E_{\mathcal{P}}} St_{init}$*.*

Improving the super lazy intruder. When we detect a state St with a super lazy term t, we may want to analyze whether the variables of t may be eventually instantiated or not before creating a ghost state. Therefore, if for each strand $[m_1^{\pm},\ldots,m_{j-1}^{\pm} \mid m_j^{\pm},\ldots,m_k^{\pm}]$ in St and each $i \in \{1,\ldots,j-1\}$, $\mathcal{V}ar(t) \cap \mathcal{V}ar(m_i) = \emptyset$, and for each term $w{\in}\mathcal{I}$ in the intruder knowledge, $\mathcal{V}ar(t) \cap \mathcal{V}ar(w) = \emptyset$, then we can clearly infer that the variables of t can never be instantiated and adding a ghost to state St is unnecessary.

Interaction with transition subsumption. When a ghost state is reactivated, we see from the above definition that such a reactivated state will be $\mathcal{P}$-subsumed by the original state that raised the ghost expression. Therefore, the transition subsumption of Section 4.5 has to be slightly modified to avoid checking a resuscitated state with its predecessor ghost state, i.e., $St_1 \succeq'_{\mathcal{P}} St_2$ iff $St_1 \succeq_{\mathcal{P}} St_2$ and St_2 is *not* a resuscitated version of St_1.

5 Experimental Evaluation

In Table 1, we summarize the experimental evaluation of the impact of the different state space reduction techniques for various example protocols searching up to depth 4. We measure several numerical values for the techniques: (i) number of states at each backwards narrowing step, and (ii) whether the state space is finite or not. The experiments have been performed on a MacBook with 2 Gb RAM using Maude 2.4. The protocols are the following: (i) NSPK, the standard Needham-Schroeder protocol, (ii) SecReT06, a protocol with an attack using type confusion and a bounded version of associativity that we presented in [8], (ii) SecReT07, a short version of the Diffie-Hellman protocol that we presented in [6], and (iv) DH, the Diffie-Hellman protocol of Example 1.

The overall percentage of state-space reduction for each protocol and an average (96%) suggest that our combined techniques are remarkably effective (the reduced number of states is on average only 4% of the original number of states). The state reduction achieved by consuming input messages first is difficult to

Table 1. Number of states for 1,2,3, and 4 backwards narrowing steps

Protocol	none	Input First	%	Inconsistent	%	Grammar	%	Subsump	%	Lazy	%	All	%
NSPK	5-15-104-427	1-11-11-145	69	5-14-71-176	51	3-7-27-86	77	5-15-61-107	65	5-15-104-405	3	1-4-3-6	97
SecReT06	1-7-26-154	1-19-33-222	0	1-7-26-152	1	1-2-6-14	87	1-7-14-18	78	1-7-26-154	0	1-2-2-1	96
SecReT07	6-15-94-283	1-11-30-242	28	6-10-32-75	69	5-13-70-201	27	6-15-74-192	27	6-11-24-52	76	1-4-4-5	96
DH	2-24-78-385	2-24-29-435	0	2-22-27-212	46	2-8-22-53	82	2-14-26-102	70	2-24-78-369	3	2-4-4-6	96
% Reduction			24		41		68		60		20		96

Table 2. Finite state space achieved by reduction techniques

Protocol	Finite State Space Achieved by:
NSPK	Grammars and Subsumption
SecReT06	Subsumption or (Grammars and Lazy)
SecReT07	Subsumption and Lazy
DH	Grammars and Subsumption

analyze because it can reduce the number of states in protocols that contain several input messages in the strands, as in the NSPK protocol, but in general simply reduces the length of the narrowing sequences and therefore more states are generated at a concrete depth of the narrowing tree. The use of grammars and the transition subsumption are clearly the most useful techniques in general. Indeed, all examples have a *finite search space* thanks to the use of the different state space reduction techniques. Figure 2 summarizes the different techniques providing a finite space. Note that grammars are insufficient for the SecReT07 example, while the super lazy intruder is essential.

6 Concluding Remarks

The Maude-NPA can analyze the security of cryptographic protocols, modulo given algebraic properties of the protocol's cryptographic functions, in executions with an unbounded number of sessions and with no approximations or data abstractions. In this full generality, protocol security properties are well-known to be undecidable. The Maude-NPA uses backwards narrowing-based search from a symbolic description of a set of attack states by means of patterns to try to reach an initial state of the protocol. If an attack state is reachable from an initial state, the Maude-NPA's complete narrowing methods are guaranteed to prove it. But if the protocol is secure, the backwards search may be infinite and never terminate.

It is therefore very important, both for efficiency and to achieve full verification whenever possible when a protocol is secure, to use *state-space reduction techniques* that: (i) can drastically cut down the number of states to be explored; and (ii) have in practice a good chance to make the, generally infinite, search space finite without losing soundness of the analysis; that is, so that if a protocol is indeed secure, failure to find an attack in such a finite state space guarantees the protocol's security for all reachable states. We have presented a number of state-space reduction techniques used in combination by the Maude-NPA for exactly these purposes. We have given precise characterizations of theses techniques and have shown that they preserve soundness, so that if no attack is found and the state space is finite, full verification of the given security property is achieved.

Using several representative examples we have also given an experimental evaluation of these techniques. Our experiments support the conclusion that, when used in combination, these techniques: (i) typically provide drastic state space reductions; and (ii) they can often yield a *finite* state space, so that whether the desired security property holds or not can in fact be decided automatically, in spite of the general undecidability of such problems.

References

1. Basin, D., Mödersheim, S., Viganò, L.: OFMC: A symbolic model checker for security protocols. Int'l Journal of Information Security 4(3), 181–208 (2005)
2. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2001)
3. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transaction on Information Theory 29(2), 198–208 (1983)
4. Escobar, S., Meseguer, J., Sasse, R.: Effectively checking or disproving the finite variant property. In: Proc. of Term Rewriting and Applications, RTA 2008. LNCS. Springer, Heidelberg (to appear, 2008)
5. Escobar, S., Meseguer, J., Sasse, R.: Variant narrowing and equational unification. In: Proc. of Rewriting Logic and its Applications, WRLA 2008 (2008)
6. Escobar, S., Hendrix, J., Meadows, C., Meseguer, J.: Diffie-hellman cryptographic reasoning in the Maude-NRL Protocol Analyzer. In: Proc. of Security and Rewriting Techniques, SecReT 2007 (2007)
7. Escobar, S., Meadows, C., Meseguer, J.: A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties. Theoretical Computer Science 367(1-2) (2006)
8. Escobar, S., Meadows, C., Meseguer, J.: Equational cryptographic reasoning in the maude-nrl protocol analyzer. Electronic Notes in Theoretical Computer Science 171(4), 23–36 (2007)
9. Escobar, S., Meseguer, J.: Symbolic model checking of infinite-state systems using narrowing. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 153–168. Springer, Heidelberg (2007)
10. Thayer Fabrega, F.J., Herzog, J., Guttman, J.: Strand Spaces: What Makes a Security Protocol Correct? Journal of Computer Security 7, 191–230 (1999)
11. Meadows, C.: The NRL protocol analyzer: An overview. Journal of logic programming 26(2), 113–131 (1996)
12. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science 96(1), 73–155 (1992)
13. Meseguer, J.: Membership algebra as a logical framework for equational specification. In: Parisi-Presicce, F. (ed.) WADT 1997. LNCS, vol. 1376, pp. 18–61. Springer, Heidelberg (1998)
14. Meseguer, J., Thati, P.: Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. Higher-Order and Symbolic Computation 20(1-2), 123–160 (2007)
15. Shmatikov, V., Stern, U.: Efficient finite-state analysis for large security protocols. In: 11th Computer Security Foundations Workshop — CSFW-11. IEEE Computer Society Press, Los Alamitos (1998)
16. Terese (ed.): Term Rewriting Systems. Cambridge University Press, Cambridge (2003)