

Undecidability of bounded security protocols*

N.A. Durgin P.D. Lincoln J.C. Mitchell A. Scedrov

Computer Science Lab
SRI International
Menlo Park, CA
lincoln@csl.sri.com

Computer Science Dept.
Stanford University
Stanford, CA 94305-9045
{nad, jcm}@cs.stanford.edu
(650) 723-8634, Fax:725-4671

Mathematics Dept.
University of Pennsylvania
Philadelphia, PA
andre@cis.upenn.edu

Abstract

Using a multiset rewriting formalism with existential quantification, it is shown that protocol security remains undecidable even when rather severe restrictions are placed on protocols. In particular, even if data constructors, message depth, message width, number of distinct roles, role length, and depth of encryption are bounded by constants, secrecy is an undecidable property. If protocols are further restricted to have no new data (nonces), then secrecy is DEXPTIME-complete. Both lower bounds are obtained by encoding decision problems from existential Horn theories without function symbols into our protocol framework. The way that encryption and adversary behavior are used in the reduction sheds some light on protocol analysis.

1 Introduction

Security protocols are difficult to design and analyze for several reasons. In addition to subtleties involving various cryptographic primitives, most protocols are intended to operate correctly when many instances of the protocol are executed in parallel. A basic client-server protocol, for example, might allow a client to request permission for a session, open that session, halt and reopen, and then finally close the session. This might seem fairly simple, except that several clients may simultaneously request several sessions with the same server, and a malicious attacker may combine data from separate sessions in order to confuse the server.

Therefore, verification of a simple protocol may involve analyzing relatively complicated attacks that combine data from any number of valid or aborted runs of the protocol.

Since many published security protocols have subtle flaws, past researchers have devised a variety of formal methods for protocol analysis. Most formal approaches in current use adopt the so-called “Dolev-Yao model” of protocol execution and attack. This model, which involves idealized assumptions about cryptographic primitives and a nondeterministic adversary, appears to have developed from the perspective of [13] and a simplified stateless model presented in [6].

A multiset rewriting framework for protocol analysis, using the Dolev-Yao model, was introduced in [7, 2]. In the present paper, the multiset formalism is used to prove upper and lower complexity bounds on protocol analysis in the Dolev-Yao model. A strong form of undecidability for protocol analysis is presented, along with DEXPTIME completeness for protocols and attacks that do not generate new data once the protocol has begun.

Since most properties of most classes of programs are undecidable, there are many ways to find undecidable classes of protocols. Previous protocol undecidability results [8, 10] have used general forms of protocols that allow agents to take any number of steps and communicate data of unbounded complexity. Since most protocols in use are finite length, we use our rewriting framework to identify a class of finite-length protocols, where each of a finite set of possible roles (such as sender, receiver, server) involves only a finite number of steps. In addition, we bound the use of data constructors, bound the number of symbols that may occur in any message, and bound the depth of encryption by constants. Even in this restrictive case, secrecy

*Partially supported by DoD MURI “Semantic Consistency in Information Exchange” as ONR Grant N00014-97-1-0505, and by NSF Grants CCR-9509931, CCR-9629754, and CCR-9800785 to various authors.

(as well as other properties) is undecidable. The intruder plays a central role by replaying data in a possibly unbounded set of protocol runs. In addition, encryption is used in a crucial way, to limit the power of the intruder to decompose and alter the message parts that are replayed.

2 Protocol Formalism

The protocol formalism we use involves *facts* and *transitions*. The facts are first-order atomic formulas, and transitions are given by rewrite rules containing a precondition and postcondition. One property of this formalism is that in applying a rule to a collection of facts, each fact that occurs in the precondition of the rule is removed. This gives us a direct way of representing state transitions, and provides the basis for a connection with linear logic [9]. Another key property is that the postconditions of a rule may contain existentially quantified variables. Following the standard proof rules associated with existential quantification (in natural deduction or sequent-style systems), this provides a mechanism for choosing new values that are distinct from any other in the system.

More formally, our syntax involves terms, facts and rules. If we want to represent a system in this notation, we begin by choosing a vocabulary, or *first-order signature*. As usual, the *terms* over a signature are the well-formed expressions produced by applying functions to arguments of the correct sort. A *fact* is a first-order atomic formula over the chosen signature. This means that a fact is the result of applying a predicate symbol to terms of the correct sorts. A *state* is a multiset of facts (all over the same signature).

A state transition is a *rule* written using two multisets of facts, and existential quantification, in the following syntactic form:

$$F_1, \dots, F_k \longrightarrow \exists x_1 \dots \exists x_j. G_1, \dots, G_n$$

The meaning of this rule is that if some state S contains facts F_1, \dots, F_k , then one possible next state is the state S' that is similar to S , but with:

- facts F_1, \dots, F_k removed,
- G_1, \dots, G_n added, where $x_1 \dots x_j$ are replaced by new symbols.

If there are free variables in the rule, these are treated as universally quantified throughout the rule. In an application of a rule, these variables may be replaced by any terms. A Theory \mathcal{T} is a finite set of facts and rewrite rules of form $l \rightarrow r$.

The multiset-rewriting notation used in this paper is a first-order Horn fragment of linear logic [9], with existential quantification. Specifically, each transition rule

$$A_1, \dots, A_n \longrightarrow \exists \vec{x}. B_1, \dots, B_m$$

can be written as a linear logic formula

$$A_1 \otimes \dots \otimes A_n \multimap \exists \vec{x}. B_1 \otimes \dots \otimes B_m$$

Under this correspondence, every derivation using multiset rewriting corresponds to a linear logic derivation, and conversely.

3 Bounded Protocols

It is relatively straightforward to use the multiset rewriting framework summarized in the preceding section to describe finite-state and infinite-state systems. Using function symbols, it is possible to describe computation over unbounded data types. In particular, it is easy to encode counter machines or Turing machines, implying that secrecy is undecidable. However, the principal authentication and secrecy protocols of interest are all of bounded length, and most use data of bounded complexity (see [3] for a relevant survey).

In order to study finite-length protocols more carefully, we identify the syntactic form of a class of well-founded protocol theories, here called simply *well-founded theories*.

3.1 Creation, consumption, persistence

Some preliminary definitions from [2] involve the ways that a fact may be created, preserved, or consumed by a rule. While multiple copies of some facts may be needed in some derivations, we are able to eliminate the need for multiple copies of certain facts.

Definition 1. A rule $l \rightarrow r$ in a theory \mathcal{T} *creates* P facts if some $P(\vec{t})$ occurs more times in r than in l . A rule $l \rightarrow r$ in a theory \mathcal{T} *preserves* P facts if every $P(\vec{t})$ occurs the same number of times in r and l . A rule $l \rightarrow r$ in a theory \mathcal{T} *consumes* P facts if some fact $P(\vec{t})$ occurs more times in l than in r . A predicate P in a theory \mathcal{T} is *persistent* if every rule in \mathcal{T} which contains P either creates or preserves P facts.

As an example, a rule of form

$$P(\vec{x}) \rightarrow P(\vec{y})$$

does not preserve P facts, since it can be used to create a fact $P(\vec{t})$ and consume a fact $P(\vec{s})$.

Since a persistent fact is never consumed by any rule, there is no need to generate more than one copy of a particular fact – as long as that fact is never needed more than once by a single rule. However, by simple transformation, it is possible to eliminate the need for more than one copy of any persistent fact [2].

Definition 2. A rule $l \rightarrow r$ in a theory \mathcal{T} is a *single-persistent rule* if all predicates that are persistent in theory \mathcal{T} appear at most once in l . A theory \mathcal{T} is a *uniform theory* if all rules in \mathcal{T} are single-persistent rules.

Since any theory can be rewritten as a uniform theory, we will assume that all theories discussed from this point forward are uniform theories.

Definition 3. Let \mathbf{P} be a set of predicates, each persistent in a uniform theory \mathcal{T} . Two states S and S' are *P-similar* (denoted $S \simeq_P S'$) if, after removing all duplicate persistent P facts from each state, they are equal multisets.

Lemma 1. If $S \simeq_P S'$ and $S \xrightarrow{\mathcal{T}} T$, then $\exists T'. T \simeq_P T'$ with $S' \xrightarrow{\mathcal{T}} T'$.

3.2 Protocol theories

In many protocols, there is an implicit or explicit initialization phase that distributes keys or establishes other shared information. Following this initialization phase, each agent may choose to carry out the protocol any number of times, in any combination of roles. For example a principal A may play the role of initiator twice, and responder once, during the course of a single attack. We incorporate these ideas into our formal definitions by letting a protocol theory consist of an initialization theory, a role generation theory, and the disjoint union of bounded subtheories that each characterize a possible role. In order to bound the entire protocol, we must assume that the initialization theory is bounded, and that initialization can be completed prior to the execution of the protocol steps proper. An example of a protocol theory is given in Section 4.

Definition 4. A rule $R = l \rightarrow r$ enables a rule $l' \rightarrow r'$ if there exist σ, σ' such that some fact $P(\vec{t}) \in \sigma r$, is also in $\sigma' l'$. A theory \mathcal{T} precedes a theory \mathcal{R} if no rule in \mathcal{R} enables a rule in \mathcal{T} .

In particular, if a theory \mathcal{T} precedes a theory \mathcal{R} , then no predicates that appear in the left hand side of rules in \mathcal{T} are created by rules that are in \mathcal{R} .

Definition 5. A theory \mathcal{A} is a *bounded role theory* if it has an ordered set of predicates, called the *role states*

and numbered S_0, S_1, \dots, S_k for some k , such that each rule $l \rightarrow r$ contains exactly one state predicate $S_i \in l$ and one state predicate $S_j \in r$, with $i < j$. We call the first role state, S_0 , an *initial role state*.

By defining roles in this way, we ensure that each application of a rule in \mathcal{A} advances the state forward. Each instance of a role can only result in a finite number of steps in the derivation.

If $\mathcal{A}_1, \dots, \mathcal{A}_k$ is a set of bounded role theories, a *role generation theory* is a set of rules of the form

$$P(\vec{s}), Q(\vec{t}), \dots \rightarrow S_i(\vec{r}), P(\vec{s}), Q(\vec{t}), \dots$$

where $P(\vec{s}), Q(\vec{t}), \dots$ is a finite list of persistent facts not involving any role states, and S_i is the initial role state for one of $\mathcal{A}_1, \dots, \mathcal{A}_k$.

Definition 6. A theory $\mathcal{S} \subset \mathcal{T}$ is a *bounded sub-theory* of \mathcal{T} if all facts created by rules R in \mathcal{S} either contain existentials or are persistent in \mathcal{T} .

Definition 7. A theory \mathcal{P} is a *well-founded protocol theory* if $\mathcal{P} = \mathcal{J} \uplus \mathcal{R} \uplus \mathcal{A}_1 \uplus \dots \uplus \mathcal{A}_n$ where \mathcal{J} is a bounded sub-theory (called the *initialization theory*) not involving any role states, \mathcal{R} is a role generation theory involving only facts created by \mathcal{J} and the initial roles states of $\mathcal{A}_1, \dots, \mathcal{A}_n$, and $\mathcal{A}_1, \dots, \mathcal{A}_n$ are well-founded role theories, with \mathcal{J} preceding \mathcal{R} and \mathcal{R} preceding $\mathcal{A}_1, \dots, \mathcal{A}_n$.

This form allows derivations in a protocol theory to be broken down into three stages – the initialization stage, the role generation stage, and the protocol execution stage.

Lemma 2. Given a well-founded protocol theory $\mathcal{P} = \mathcal{J} \uplus \mathcal{R} \uplus \mathbf{A}$, where \mathcal{J} is an initialization theory, \mathcal{R} is a role generation theory, and \mathbf{A} is the disjoint union of one or more bounded role theories, if $S \xrightarrow{\mathcal{P}} T$ is a derivation over \mathcal{P} , then there exists a derivation $S \xrightarrow{\mathcal{J}, \mathcal{R}} S'$ and $S' \xrightarrow{\mathbf{A}} T$, where all rules from \mathcal{J} and \mathcal{R} are applied before any rules from \mathbf{A} .

3.3 Intruder theory

One of the original motivations for using multiset rewriting for protocol analysis was that this framework allows us to use essentially the same theory for all adversaries for all protocols. In this subsection, we specify the properties of intruder theories that are needed to bound the number of intruder steps needed to produce a given message. As explained in [4], the actions of the standard intruder can be separated into two phases, one in which messages are decomposed into smaller

parts, and one in which these parts are (re)assembled into a message that will be sent to some protocol agent.

In determining the *size* of a fact, we count the predicate name, each function name, and each variable or constant symbol. For example, fact $P(A, B)$ has size 3, and fact $P(f(A, B), C)$ has size 5.

Definition 8. A rule $R = l \rightarrow r$ is a *composition rule* if the size of the largest non-persistent fact in r is greater than the largest non-persistent fact in l . A rule $R = l \rightarrow r$ is a *decomposition rule* if the size of the largest non-persistent fact in r is less than the largest non-persistent fact in l .

For example,

$$C(A), C(B) \rightarrow C(\langle A, B \rangle)$$

is a composition rule, and

$$D(\langle A, B \rangle) \rightarrow D(A), D(B)$$

is a decomposition rule.

For the intruder theories we will consider, we allow persistent facts to appear in both the left and right hand sides. So, in general a decomposition rule is of form:

$$D(\langle A, B \rangle), \vec{P}(\dots) \rightarrow D(A), D(B), \vec{P}'(\dots)$$

where \vec{P} and \vec{P}' are sets of persistent predicates, with $\vec{P} \subseteq \vec{P}'$ (and similarly for composition rules).

We also need to introduce more complicated decomposition rules, which we call “Decomposition rules with Auxiliary facts”. These are pairs of rules of form:

$$D(t), \vec{P}(\dots) \rightarrow \vec{P}'(\dots), A(t)$$

and

$$A(t), \vec{Q}(\dots) \rightarrow \vec{Q}'(\dots), D(t')$$

where $\vec{P} \subseteq \vec{P}'$, $\vec{Q} \subseteq \vec{Q}'$, and $size(t') < size(t)$. Here, A represents an Auxiliary fact (which can appear only in a pair of rules of this form) which is used to amortize the decomposition of $D(t)$ into $D(t')$ across the two rules. Section 4.4 shows an example of this type of decomposition rule, used to allow decrypting an old fact with a newly learned encryption key.

Definition 9. A theory \mathcal{T} is a *two-phase* theory if its rules can be divided into three disjoint theories, $\mathcal{T} = \mathcal{J} \uplus \mathcal{C} \uplus \mathcal{D}$, where \mathcal{J} is a bounded sub-theory preceding \mathcal{C} and \mathcal{D} , \mathcal{C} contains only composition rules, \mathcal{D} contains only decomposition rules, and no rules in \mathcal{C} precede any rules in \mathcal{D} .

Definition 10. A *normalized derivation* is a derivation where all rules from the decomposition theory are applied before any rules from the composition theory.

As also shown in [4] in a slightly different context, all derivations in a two-phase theory can be expressed as normalized derivations.

Lemma 3. *If a theory \mathcal{T} is two-phase, and we limit the size of terms, and we limit the number of times each existential is instantiated, then there are only finitely many normalized derivations in the theory.*

In the Dolev-Yao model, the protocol adversary has the capability to overhear, remember, and block messages, to compose/decompose and decrypt/encrypt message fields, and to generate new messages and send them to any other protocol participant. The messages generated by the intruder may be composed of any information supplied to the intruder initially (such as public keys of protocol participants), fresh data generated by the intruder, and data obtained by overhearing or intercepting messages.

The intruder is easily formalized as a set of rewrite rules. While the basic intruder steps remain the same from one protocol to the next, the exact formalization depends on the form of messages used in the protocol. A specific instance of the *standard intruder* is described in some detail in Section 4.4.

3.4 Protocol and intruder

Definition 11. Given a well-founded protocol theory $\mathcal{P} = \mathcal{J} \uplus \mathcal{R} \uplus \mathbf{A}$ and a two-phase intruder theory \mathcal{M} , a *standard trace* is a derivation that has all steps from the \mathcal{J} and \mathcal{R} first, then interleaves steps from the principal theories \mathbf{A} with normalized derivations from the intruder theory \mathcal{M} .

Theorem 1. *Let \mathcal{P} be any well-founded protocol theory and \mathcal{M} be any two-phase intruder theory. If we bound the number of uses of each existential, and we bound the number of roles generated, and we bound the size of each term, then the set of standard traces of $\mathcal{P} \cup \mathcal{M}$ is finite.*

Later, we show that secrecy is decidable under the conditions of Theorem 1, even without a bound on the number of roles.

4 Example: Needham-Schroeder Public Key Protocol

As an example, we give the full theory of the three-step core of the Needham-Schroeder public-key protocol.

$$\begin{array}{lcl}
A & \longrightarrow & B : \{A, N_a\}_{K_b} \\
B & \longrightarrow & A : \{N_a, N_b\}_{K_a} \\
A & \longrightarrow & B : \{N_b\}_{K_b}
\end{array}$$

Note that this is a simplified version of the protocol, where the use of a trusted server to distribute the public keys is omitted.

4.1 Initialization Theory

The Initialization Theory is shown in Table 1. Here, the predicate *GoodGuy* indicates an uncompromised principal, parameterized by its encryption and decryption (public and private) keys. For simplicity, we identify the principal with its public key (i.e. where “A” appears in the protocol, we use the public key “ K_a ”). The GOODGUY rule allows for the creation of an unlimited number of principals, each with a unique key pair, denoted by the predicate *KP*.

The BADKEY rule provides a mechanism for specifying an unlimited number of compromised key pairs, which appear to belong to valid principals, but whose private keys are known to the intruder. The predicate *BadKey* denotes these compromised key pairs.

Since we are omitting the trusted server from this example, we accomplish key distribution by having the principals announce their keys. The ANNK rule accomplishes this for the *GoodGuy* participants, while the ANNKB rule does the same for the *BadKey* pairs. Note that both rules generate a predicate *AnnK* indicating a public key that is available for communication, so from this point the valid participants can’t distinguish the good guys from the bad guys.

4.2 Role Generation Theory

In the Role Generation Theory, ROLA and ROLB allow an unlimited number of sessions to be started for any principal to act in the role of either “Alice” (the initiator) or “Bob” (the responder). A_0 and B_0 denote the initial role state for the A and B roles, respectively, parameterized by the public key (principal) acting in that role.

4.3 Protocol Role Theories

The Role Theories, shown in Table 1, are derived directly from the specification of the Needham-Schroeder protocol. Theory \mathcal{A} corresponds to the role of “Alice”, and theory \mathcal{B} corresponds to “Bob”.

In rule A1, which corresponds to the first line of the protocol, a principal k_e , in its initial state A_0 , decides

to talk to another principal k'_e , whose key has been announced. A new nonce x is generated, along with a network message N_{S1} corresponding to the first message sent in the protocol, and the principal moves to the new state A1, remembering the values of x and k'_e . Note that since *AnnK* is persistent, it must also appear on the right hand side of the rule.

In step B1, corresponding to the second step of the protocol, a principal k_e , in the initial state B_0 , responds to a message on the network which is of the expected format (i.e. encrypted with k_e ’s public key, and with the identity of a participant whose key has been announced, embedded inside). k_e generates another nonce, and replies to the message, moving to a new state B_1 where all the information (the two nonces and the two principals) is remembered.

Similarly, A2 corresponds to the third line of the protocol, and B2 corresponds to the implicit step where the responder actually receives the final message.

Note that sent messages are denoted by N_{Si} and received messages are denoted by a corresponding predicate N_{Ri} . The minimal intruder theory can be thought of as providing a network that, transforms N_{Si} ’s to N_{Ri} ’s, so the protocol can execute. The subscripts on the network messages indicate format information that allows participants to distinguish the messages of a protocol. These do not limit the analysis, since the intruder can transform messages from one type to another.

4.4 Intruder Theory

The Intruder Theory, which is a typical Standard Intruder, is shown in Table 2. Here the M predicate denotes persistent facts known to the intruder, while D and C represent non-persistent facts which can be decomposed and composed into other facts.

LRNKB and LRNK are initialization rules that allow the intruder to learn keys. Since the *BadKey* and *AnnK* predicates are generated only by the initialization theory, we know from Lemma 1 that this rule only needs to be applied once per derivation, per *BadKey* and *AnnK* fact.

The REC and SND rules are used to connect the intruder to the network being used by the participants. The REC rule intercepts a message from the network and saves it as a decomposable fact. The SND rule sends composed facts onto the network.

The COMP rule allow the user to compose small terms into larger ones, while the DCMP rule allows for decomposition of large terms into smaller ones..

LRN converts a decomposable fact into intruder knowledge, and USE converts intruder knowledge into

Initialization Theory \mathcal{I} :

GOODGUY:		$\rightarrow \exists k_e.k_d. GoodGuy(k_e, k_d), KP(k_e, k_d)$
BADKEY:		$\rightarrow \exists k_e.k_d. KP(k_e, k_d), BadKey(k_e, k_d)$
ANNK:	$GoodGuy(k_e, k_d)$	$\rightarrow AnnK(k_e), GoodGuy(k_e, k_d)$
ANNKB:	$BadKey(k_e, k_d)$	$\rightarrow AnnK(k_e), BadKey(k_e, k_d)$

Role Generation Theory \mathcal{R} :

ROLA:	$GoodGuy(k_e, k_d)$	$\rightarrow GoodGuy(k_e, k_d), A_0(k_e)$
ROLB:	$GoodGuy(k_e, k_d)$	$\rightarrow GoodGuy(k_e, k_d), B_0(k_e)$

Protocol Theories \mathcal{A} and \mathcal{B} :

A1:	$AnnK(k'_e), A_0(k_e)$	$\rightarrow \exists x. A_1(k_e, k'_e, x), N_{S1}(enc(k'_e, \langle x, k_e \rangle)), AnnK(k'_e)$
A2:	$A_1(k_e, k'_e, x), N_{R2}(enc(k_e, \langle x, y \rangle))$	$\rightarrow A_2(k_e, k'_e, x, y), N_{S3}(enc(k'_e, y))$
B1:	$B_0(k_e), N_{R1}(enc(k_e, \langle x, k'_e \rangle)), AnnK(k'_e)$	$\rightarrow \exists y. B_1(k_e, k'_e, x, y), N_{S2}(enc(k'_e, \langle x, y \rangle)), AnnK(k'_e)$
B2:	$B_1(k_e, k'_e, x, y), N_{R3}(enc(k_e, y))$	$\rightarrow B_2(k_e, k'_e, x, y)$

Table 1. Needham Schroeder Theory

a composable fact.

The ENC and DEC rules allow the intruder to decrypt a message if it knows the private key, and to generate encrypted message from known public keys.

Note that LRNA and DECA are decomposition rules with auxiliary facts that handle a special case for encrypted messages. If the message can't be decrypted because the key isn't currently known, LRNA remembers the decrypted message with the special "Auxiliary" predicate, A . The DECA rule allows Auxiliary messages to be decrypted at a later time, if the decryption key becomes known.

Finally, GEN allows the intruder to generate new facts (i.e. nonces) as needed.

This Intruder Theory can be divided into Composition and Decomposition rules, as shown in Table 2. So, this is a Two-Phase Intruder Theory, as described in Section 3.

5 Undecidability

In this section, we show that secrecy is undecidable for bounded protocols of a restricted form. In general, a secrecy specification stipulates that certain "secret" data must not fall into the hands of the intruder. This is a derivability or reachability problem in our framework: given an initial fact such as $Alice(k_a, k_a^{-1})$, indicating that principal Alice has private key k_a^{-1} , is there

a run of the protocol and intruder in which the adversary knows k_a^{-1} , i.e., $M(k_a^{-1})$ appears in the state of the system?

5.1 Restricted protocol form

In general, it is convenient to write the steps of a protocol agent A in the form

$$A_i(\dots), N_j(\dots), P(\dots), Q(\dots), \dots \\ \rightarrow \exists \dots . A_k(\dots), N_\ell(\dots), P(\dots), Q(\dots), \dots$$

where $P(\dots), Q(\dots), \dots$ are persistent facts appearing on the left and right of the rule. However, for the purpose of proving a stronger negative result, we restrict our attention to a simpler form of protocol step in this section. Specifically, we say a role is in *restricted form* if it consists only of rules of the form

$$A_i(\dots), N_j(\dots) \rightarrow \exists \dots . A_k(\dots), N_\ell(\dots)$$

with one principal role state and one network message on the left and one principal role state and one network message on the right. As with all finite-length protocols, we assume the states of agent A are given by a finite list of predicates A_1, \dots, A_a . We also assume that set of possible network messages are given by a finite list of predicates N_1, \dots, N_n , with $i < k \leq a$ and $j < \ell \leq n$ in each rule of the form above.

Initialization Rules:

LRNKB:	$BadKey(k_e, k_d)$	\rightarrow	$M(k_e), M(k_d), BadKey(k_e, k_d)$
LRNK:	$AnnK(k_e)$	\rightarrow	$M(k_e), AnnK(k_e)$

I/O Rules:

REC:	$N_{S_i}(x)$	\rightarrow	$D(x)$
SND:	$C(x)$	\rightarrow	$N_{R_i}(x)$

Decomposition Rules:

DCMP:	$D(\langle x, y \rangle)$	\rightarrow	$D(x), D(y)$
LRN:	$D(x)$	\rightarrow	$M(x)$
DEC:	$M(k_d), KP(k_e, k_d), D(enc(k_e, x))$	\rightarrow	$M(k_d), KP(k_e, k_d), D(x), M(enc(k_e, x))$
LRNA:	$D(enc(k_e, x))$	\rightarrow	$M(enc(k_e, x)), A(enc(k_e, x))$
DECA:	$M(k_d), KP(k_e, k_d), A(enc(k_e, x))$	\rightarrow	$M(k_d), KP(k_e, k_d), D(x)$

Composition Rules:

COMP:	$C(x), C(y)$	\rightarrow	$C(\langle x, y \rangle)$
USE:	$M(x)$	\rightarrow	$C(x), M(x)$
ENC:	$M(k_e), C(x)$	\rightarrow	$C(enc(k_e, x)), M(k_e)$
GEN:		\rightarrow	$\exists x.M(x)$

Table 2. Two-Phase Intruder Theory

The purpose of the initialization theory, in our framework, is to formalize the choice of initial conditions, such as shared public or private keys. However, as we have defined protocol theories, there are few restrictions on the form of initialization theories. Since derivability in multiset rewriting is undecidable, we can prove trivial lower bounds by encoding complex problems into the initialization theory. However, this kind of lower bound would not shed any light on protocol analysis. In order to avoid this essentially degenerate case, we will analyze decidability and complexity for protocol theories with initialization theories that consist only of a finite set of ground facts, and no rewrite rules. Intuitively, this means that we analyze decidability and complexity of the role generation and protocol execution phases, under the assumption that initialization has already been completed.

Definition 12. A protocol theory $\mathcal{T} = \mathcal{J} \uplus \mathcal{R} \uplus \mathbf{A}$ is in *restricted form* if

- The initialization theory \mathcal{J} is a set of ground facts.
- \mathbf{A} is a finite set of roles, each in restricted form.

Note that the role generation theory \mathcal{R} is not bounded: a protocol theory in restricted form may have a standard role generation phase that allows each initialized principal to participate in an arbitrary number of bounded roles.

Theorem 2. *Secrecy is undecidable for protocol theories in restricted form. More specifically, there is no algorithm for deciding whether a given protocol, run in combination with the standard intruder, allows the intruder to gain access to a given initial secret.*

The proof involves representing existential Horn theories as protocols, as described below. While space limitations prevent us from presenting the proof in detail, we will sketch the main ideas.

5.1.1 Representation of existential Horn theories

Given a set of existential Horn formulas, of the form described in Appendix A and repeated below, we can construct a protocol so that, when combined with the standard intruder theory, the intruder memory may contain formulas representing all consequences of the theory.

In order to do this, we must use the intruder in an essential way. Specifically, each agent can only execute a finite sequence of steps. Therefore, we use a separate agent for each Horn clause. The role of the intruder is to convert the final message sent by one agent to

an initial message received by another agent. As a result of intruder actions, a datum may pass through an unbounded number of protocol steps.

At the same time, in order to represent the Horn theory faithfully, we cannot give the intruder complete access to all to atomic formulas used in a Horn clause. In particular, we cannot let the intruder combine data from different messages. For example, if one agent sends a message representing $P(a, b)$, we cannot allow the intruder to intercept this message and replace it with $P(b, a)$. However, it is easy to prevent this form of interference if we encrypt atomic formulas with a shared private key.

Putting these two ideas together, we represent an existential Horn clause theory by a protocol with one agent per clause. The agent A for a clause

$$\forall x_1 \dots \forall x_i [(\alpha_1 \wedge \dots \wedge \alpha_k) \implies \exists y_1 \dots \exists y_j (\beta_1 \wedge \dots \wedge \beta_\ell)]$$

is

$$A_0, N_{a_0}([\alpha_1 \wedge \dots \wedge \alpha_k]) \rightarrow \exists y_1 \dots \exists y_j. A_1, N_{a_1}([\beta_1 \wedge \dots \wedge \beta_\ell])$$

where $j \geq 0$, $k, \ell \geq 1$, and where the encoding $[\alpha_1 \wedge \dots \wedge \alpha_k]$ of a conjunction of atomic formulas as a single atomic formula is described below. In particular, the agent A for a pure Horn clause

$$\forall x_1 \dots \forall x_i [(\alpha_1 \wedge \dots \wedge \alpha_k) \implies \beta]$$

is

$$A_0, N_{a_0}([\alpha_1 \wedge \dots \wedge \alpha_k]) \rightarrow A_1, N_{a_1}([\beta]).$$

There are several ways to represent a conjunction of atomic formulas as a single network message, containing only one predicate symbol, hidden from the adversary. One way, intended to keep syntactic complication to a minimum, is to assume that for each k , we have a k -ary encryption function Encrypt_k . For notational simplicity, we will suppress k in the description below. We also assume that for each sequence of predicates P_1, \dots, P_k that occurs together in the left or right hand side of a given Horn clause, we have a constant symbol $P_1.P_2 \dots .P_k$. (Although we have used a sequence of letters, numbers and subscripts to write out our name for this constant symbol, we assume it is an atomic constant symbol of the language.) Given these assumptions, we let

$$\begin{aligned} & [P_1(t_{1,1}, \dots, t_{1,i_1}) \wedge \dots \wedge P_j(t_{j,1}, \dots, t_{j,i_j})] \\ & = \text{Encrypt}(P_1.P_2 \dots .P_j, t_{1,1}, \dots, t_{j,1}, \dots, t_{j,i_j}) \end{aligned}$$

We assume that all encryption is done using the same key, and that the key is shared among honest participants but not revealed to the intruder. (It suffices

to have one principal executing several roles, using a private key.)

The final main idea in the representation of Horn theories is the way that a set of conjunctions may be combined to produce the conjunction of atomic formulas needed to apply another Horn clause. This is perhaps best illustrated by example.

Suppose that the existential Horn clause

$$\forall x \forall y [(P(x) \wedge Q(x, y) \wedge R(y)) \implies \exists z (P(z) \wedge Q(y, z))]$$

is part of the Horn theory we wish to represent by a protocol. In order to use this implication, the protocol must produce a message containing $[(P(a) \wedge Q(a, b) \wedge R(b))]$ for some a and b . However, the protocol agents that represent Horn clauses produce encodings of conjunctions of atomic formulas, and the atomic formulas here may come from different rules. Therefore, we need additional protocol agents that select atomic formulas out of conjunctions and combine them.

The process is very similar to the encoding of the intruder, except that protocol agents can manipulate encrypted values. For each conjunction form (including variables) that appears on the right-hand side of a Horn clause, such as $P(z) \wedge Q(y, z)$, we include *decomposition agents* of the form

$$A_0, N_0([P(z) \wedge Q(y, z)]) \rightarrow A_1, N_1([P(z)])$$

and

$$B_0, N_0([P(z) \wedge Q(y, z)]) \rightarrow B_1, N_1([Q(y, z)])$$

for predicates A_0, A_1, B_0, B_1 not used for other agents. We also need a *composition agent* for the left-hand-side of each original Horn clause. For the clause above, the agent will have states A_0, A_1, A_2 and A_3 . At each step, the agent reads one of the atomic formulas in its target conjunction, sending out either a dummy message or, at the last step, a message containing the conjunction of atomic formulas needed. In order to assemble $[(P(x) \wedge Q(x, y) \wedge R(y))]$, we can use an agent A with the following steps:

$$\begin{aligned} A_0, N_0([P(x)]) &\rightarrow A_1([P(x)], N_1() \\ A_1([P(x)], N_2([Q(x, y)]) &\rightarrow \\ &A_2([P(x) \wedge Q(x, y)], N_3() \\ A_2([P(x) \wedge Q(x, y)], N_4([R(y)]) &\rightarrow \\ A_3(), N_5([P(x) \wedge Q(x, y) \wedge R(y)]) \end{aligned}$$

After this agent sends message N_5 , the intruder can read the data $[P(x) \wedge Q(x, y) \wedge R(y)]$ contained in this message and forward it to the agent representing the Horn clause with hypothesis $P(x) \wedge Q(x, y) \wedge R(y)$.

6 Protocols without nonces

If we further restrict the protocol and intruder theories from Section 5.1 to have no existential quantification, secrecy is decidable in DEXPTIME. The elimination of existential quantification means that no new data can be generated during protocol execution, and therefore there is an exponential bound on the number of distinct messages that may appear in protocol execution. This is essentially the form of protocol analysis that is done by model checkers such as FDR and Mur φ [14, 12], where protocols that choose new nonces are formalized as protocols that select these nonces from a finite set of initial data of the appropriate type. PSPACE-completeness for this class was mistakenly stated in [2]; the argument outlined indicated only PSPACE-hardness. In fact, DEXPTIME-hardness follows by the same encoding of Horn formulas (Datalog programs) as in our undecidability proof, applied to Horn clauses without function symbols and without existential quantification. For these Horn theories, DEXPTIME-hardness of the implication problem (measured as a function of the size of the theory) is implicit in [11, 16], as explained in [5]. The lower bound for Horn theories is similar to the machine representation outlined in Appendix A, using a form of “symbolic counter” instead of Skolem symbols to name the cells in a bounded section of the Turing machine tape.

The DEXPTIME upper bound uses the observation that because role-generation is unlimited and no role creates new data, each role can be re-run as many times as desired. More specifically, if \mathcal{J} is the initialization theory, and fact $P(\vec{t})$ is in a state S derivable from \mathcal{J} by the role generation and protocol rules, then there is another state $S' \supseteq S$ containing an additional occurrence of $P(\vec{t})$ that can also be derived from \mathcal{J} by the role generation and protocol rules. Therefore, we can decide whether a fact is derivable by computing a list of all possible derivable facts. Beginning with \mathcal{J} in this list, we repeatedly select a rule and apply it to see if we can derive a new fact that is not already on the list. Since there is an exponential bound on the number of bounded-length facts that can be written over the signature, this process will terminate in exponential time. This decision procedure is essentially the same as the DEXPTIME upper bound for Datalog, once we observe that all role steps can be repeated as many times as needed [5, 11, 16].

Theorem 3. *Secrecy is DEXPTIME-complete for finite-length protocols of restricted form that have no new data. More specifically, the following decision problem is DEXPTIME-complete: Given a finite-length protocol and a limited intruder, neither of which can generate*

new data, does the protocol allow the intruder to gain access to a given initial secret?

7 Conclusion

In this paper, we have shown that secrecy is undecidable, even for a restricted class of protocols. In the undecidability proof, the intruder stores encryptions of all atomic formulas derivable from a given Horn theory without function symbols, replaying these messages as needed in order for the protocol steps to carry out an arbitrary deduction.

In some ways, undecidability might not be expected for this class of protocols. The reason is that without function symbols, there is only a finite number of possible messages, except for the unbounded number of new nonces that repeated runs of a protocol might generate. However, our undecidability proof shows that nonces may be used as a form of “pointer,” linking together messages that contain only simple data. However innocuous they may seem, nonces are at the heart of the problem in analyzing this class of security protocols. For protocols and attacks that do not generate new data, secrecy and other properties that can be stated as derivability questions are DEXPTIME-complete.

There are several open problems that remain to be fully investigated. One problem is the complexity of secrecy and related properties when the protocol does not generate new data, but the intruder is allowed to generate an unbounded number of new data in order to carry out an attack. Another is the complexity when a limit is placed on the number of protocol roles. Under the assumption that message size is bounded (as in all of our analysis in this paper), this case appears to be NP-complete. Another direction for further investigation is to examine broader classes of protocols. For example, the protocols considered in this paper can test for equality (by pattern matching in rules), but cannot test for inequality in the left-hand-side of a rule. Finally, in spite of the negative results proved in this paper, we believe there may be some parts of unbounded protocol analysis that may yield to special-purpose decision procedures. As in other theorem proving efforts, it is possible that decision procedures for special cases or subsidiary notions may be useful in developing formal proofs of protocol correctness.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
 [2] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis.

In P. Syverson, editor, *12-th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.
 [3] J. Clark and J. Jacob. A survey of authentication protocol literature. Web Draft Version 1.0 available from <http://www.cs.york.ac.uk/~jac>, 1997.
 [4] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.
 [5] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. In *Proc. 12th Annual IEEE Conference on Computational Complexity*, 1997.
 [6] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.
 [7] N. A. Durgin and J. C. Mitchell. Analysis of security protocols. In *Caclulational System Design, Series F: Computer and Systems Sciences, Vol. 173*. IOS Press, 1999.
 [8] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. Technical Report 285, Technion – Israel Institute of Technology, 1983.
 [9] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
 [10] N. Heintze and J. D. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22:16–30, 1996. Special section from 1994 IEEE Symposium on Security and Privacy.
 [11] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.
 [12] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murφ. In *Proc. IEEE Symp. Security and Privacy*, pages 141–151, 1997.
 [13] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
 [14] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, pages 98–107. IEEE Computer Soc Press, 1995.
 [15] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.
 [16] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *14-th Annual ACM Symposium on Theory of Computing*, pages 137–146, 1982.

A Horn Clauses with Existential Quantification

An *existential Horn clause* is a closed first-order formula of the form

$$\forall x_1 \dots \forall x_i [(\alpha_1 \wedge \dots \wedge \alpha_k) \implies \exists y_1 \dots \exists y_j (\beta_1 \wedge \dots \wedge \beta_\ell)]$$

where $\alpha_1, \dots, \alpha_k, \beta_1, \dots, \beta_\ell$ are first-order atomic formulas. Without restriction on the form of the atomic formulas, undecidability of the implication problem for existential Horn clauses follows immediately from the undecidability of Horn clauses without existential quantifiers. The problem of interest to us, however, is implication when the atomic formulas contain no function symbols.

The formulas we are interested in are a special case of database dependencies [1]. While database dependencies include existential quantification and do not involve function symbols, database dependencies also allow equality in the conclusions of Horn clauses. Since we do not use equality in our protocols (except by pattern matching in the hypotheses of rules), it is not immediately clear to us whether the following theorem is a consequence of standard results in database dependency theory.

Theorem 4. *The implication problem for existential Horn clauses without function symbols is undecidable. In particular, there is no algorithm for deciding whether a set of existential Horn clauses without function symbols implies a single atomic formula $A(b_1, \dots, b_k)$ without function symbols or variables.*

This theorem has a straightforward direct proof based on axiomatizing a Cook's-theorem-style Turing machine tableau. Specifically, given any Turing machine, we write axioms of the form

$$\begin{aligned} & \forall x, y, z. [(Adj(x, y) \wedge Adj(y, z) \wedge \\ & Cont(x, 0) \wedge Cont(y, 1.q_i) \wedge Cont(z, 1)) \\ & \implies \exists x', y', z' ((Adj(x', y') \wedge Adj(y', z') \wedge \\ & Below(x', x) \wedge Below(y', y) \wedge Below(z', z)) \wedge \\ & Cont(x', 0.q_j) \wedge Cont(y', 0) \wedge Cont(z, 1)) \end{aligned}$$

In this formula, the variables represent cells of the Turing machine tableau (i.e., cells of the tape at some stage of the computation); a nice picture of the tableau we use appears in [15, page 255]. The constants 0 and 1 indicate symbols in these cells, and constants of the form $0.q_i$ or $1.q_j$ indicate that the cell contains a symbol 0 or 1 and is the location of the tape head, with machine in state q_i or q_j (respectively). A fact $Adj(x, y)$ means that cell x is adjacent to y , $Below(x', x)$ that cell x' is below cell x , and $Cont(x, c)$ that the cell x has contents described by constant c , possibly giving the machine state in addition to the symbol contained in the cell. The atomic formula $A(b_1, \dots, b_k)$ mentioned in the statement of the theorem can be an atomic formula that is derivable by a rule that requires, in its hypothesis, that the Turing machine is in a halting state.