Constraint Solving for Bounded-Process Cryptographic Protocol Analysis

Jonathan Millen and Vitaly Shmatikov
Computer Science Laboratory, SRI International, Menlo Park, CA 94025 USA
{millen,shmat}@csl.sri.com

ABSTRACT

The reachability problem for cryptographic protocols with non-atomic keys can be solved via a simple constraint satisfaction procedure.

1. INTRODUCTION

Many protocol security properties can be characterized as reachability problems. This is the case for properties such as secrecy, where the objective of protocol analysis is to search for a state that violates a particular invariant, such as a state in which some secret data has been released publicly by an attacker or dishonest party.

It is known that reachability is undecidable for cryptographic protocols in the general case [9, 5]. Undecidability results from a context where the number of distinct processes instantiating protocol roles is unbounded, and there is an active attacker who can intercept and forge messages. It has been demonstrated that reachability is decidable for the finite number of processes [1, 13].

The main contribution of this paper is to develop a complete and practical decision procedure for the reachability problem in the presence of constructed (non-atomic) keys. Support for constructed keys is important for formal analysis of "real-world" protocols, as it is fairly common in protocol design to construct symmetric keys from shared secrets and other data exchanged as part of the protocol - see, for example, SSL 3.0 [7]. Some of the techniques for constructing symmetric keys involve commutative operators such as `xor` or Diffie-Hellman exponentiation, and thus lie beyond the scope of the unmodified free-algebra approach as taken in this paper.

We show how to convert the reachability problem into a constraint solving problem and present a relatively simple decision algorithm for the latter that is easy to understand and justify. The algorithm is sound and complete. We use the standard Dolev-Yao attacker model with a free term algebra for messages, and do not impose any bounds on the size of terms or cryptographic function applications by the attacker. Our cryptographic primitives include symmetric-key encryption with arbitrary non-atomic keys, public-key encryption, signatures, and hashes. There is a small, fast Prolog implementation.

CCS'01, November 5-8, 2001, Philadelphia, Pennsylvania, USA.

Protocol processes are specified here as instances of roles, using the parametric strands formalism. In the original strand space model [16, 15], messages are ground terms, but subsequent development and applications of the approach [4, 14] allow messages to contain variables (parameters) so that a single schema can represent all possible strands for a particular role in a protocol. Given a finite set of parametric strands representing processes running concurrently, they can be merged in a finite number of possible ways into a single event sequence. From such a sequence, we generate a sequence of symbolic constraints that can be solved efficiently.

Related work. Formal methods have been extensively applied to security protocol analysis. Typically, the tradeoff is between incompleteness (*e.g.*, for finite-state checking, it is necessary to impose a bound on *both* attacker capabilities and number of protocol instances) and possible non-termination (*e.g.*, in many methods based on theorem proving). Our work is closest in spirit to the approaches that use symbolic techniques to enumerate the infinite state space generated by a limited number of participants.

Huima [10] developed a method for symbolic state-space exploration, using a term algebra with canonical reductions (*e.g.*, decryption cancels encryption). Completeness was claimed but full details of the decision algorithm were not in the workshop paper, and, to the best of our knowledge, never published.

Amadio, Lugiez, and Vanackère [1, 2] also use symbolic techniques to characterize the unbounded set of possible messages generated by the attacker. These techniques are similar to ours in that they combine the use of variables in message schemas and constraints defining the allowable set of substitutions for those variables. Amadio *et al*. proved decidability in the bounded-process case, but only for atomic encryption keys (with variables in key positions handled by exhaustive substitutions) and using a simpler free term algebra. Fiore and Abadi [6] and Boreale [3] present algorithms for computing symbolic traces of infinite-state cryptographic protocols. Both methods are technically involved, and, in [6], completeness is proved only for atomic keys. Rusinowitch and Turuani [13] show the problem to be NP-complete with a free term algebra and constructed keys.

Our use of the strand space model is similar to that of Athena [14]. Athena, however, only supports atomic keys. Another difference is that Athena uses penetrator strands as specified in the original strand space model, while in our approach, no penetrator strands are constructed. Instead, we use a term closure operator based on message constructors, similar to Paulson's synth and analz [12], to characterize attacker capabilities. The result is an extremely concise representation of the problem and a clean transition from the process aspects of the protocol model to a pure constraint-solving problem.

2. PROTOCOL MODEL

2.1 Parametric strands and reachability

Protocol roles are specified with parametric strands, in which message terms may contain variables. Different strands are distinguished by different values for the set of variables (parameters) associated with a role. For example, the initiator role of Lowe's fixed version of the Needham-Schroeder public-key protocol handshake (NSL) [11] can be specified as:

$$\begin{aligned} &\mathit{Init}(A, B, N_A, N_B) = \\ &\quad +[A, N_A]^{\rightarrow}_{\mathrm{pk}(B)} - [N_A, N_B, B]^{\rightarrow}_{\mathrm{pk}(A)} + [N_B]^{\rightarrow}_{\mathrm{pk}(B)} \end{aligned}$$

The responder role $\mathit{Resp}(A, B, N_A, N_B)$ is the same except that $+$ and $-$ are interchanged.

The notation above uses $[x]^{\rightarrow}_{\mathrm{pk}(A)}$ for public-key encryption of x using the public key of principal A. The signs $+$ and $-$ denote messages sent and received, respectively, and a sent message is called a node. Term variables in messages are denoted by capital letters.

The protocol as given above is actually slightly different from Lowe's; the first message in the original is $[N_A, A]^{\rightarrow}_{\mathrm{pk}(B)}$, with the nonce first in the encrypted field. This apparently inconsequential difference leads to an attack, as shown below in Section 6.

Our constraint solving procedure analyzes a set of parametric strands, in which some of the parameters may be instantiated by constants. A set of parametric strands is called a *semibundle* in the Athena paper [14]. The sequence of nodes in each strand of a semibundle implicitly specifies the state sequencing relation $\Rightarrow$ in the strand space model. Thus, for example,

$$\{\mathit{Init}(A, B, N_A, N_B), \mathit{Resp}(A', B', N'_A, N'_B)\}$$

is a semibundle. A strand in a semibundle need not be complete; it may be an initial subsequence of the full node sequence of a role.

A semibundle can be completed to a bundle by supplying the attacker computations and the communication causality relation $\rightarrow$ between received messages and sent messages. Completion of a semibundle implies that the network state described by the bundle is reachable from an empty initial state. Some semibundles are not completable. The task of reachability analysis is to determine whether a semibundle is completable or not, and, if so, what substitution (instantiating the semibundle variables) is necessary to make it possible. The representation of security attacks as bundles is discussed in Section 2.4.

2.2 The term algebra

Messages and message fields are represented as terms in a free algebra generated by the operators in Fig. 1 from a finite set of symbolic constants. We do not distinguish different types of constants, *e.g.*, nonces, keys, and principal names, but such types could be introduced if desired. Analysis of particular semibundles will dictate how many distinct constants are needed. One particular constant is always available, the name of the attacker, ε.

The term algebra allows any term to be used as an encryption key both for public-key and symmetric encryption. We make, however, a (fairly realistic) assumption that private keys are never leaked. Also, while the model supports constructed keys, the only construction operators that can be used are those, such as hashing, that can be expressed in a free term algebra. Support for operators with associative and commutative properties such as `xor`, explicit decryption operators, and relaxing the secure private keys assumption will require us to add an equational theory to the term algebra. This is a topic of current research.

ε	The attacker or a principal compromised by the attacker (constant)
$[t_1, t_2]$	Pairing
$\mathrm{pk}(P)$	Messages encrypted with this public key can be decrypted by P using its corresponding private key. We assume that the private key of a public-private key pair is never transmitted as part of the protocol, or compromised in any way that might make it available as initial knowledge of the attacker. Therefore, the attacker can only decrypt terms encrypted with its own public key $\mathrm{pk}(\varepsilon)$.
$h(t)$	Hash (modeled as a one-way function)
$[t]^{\leftrightarrow}_k$	Term t encrypted with k using a symmetric algorithm. Keys are not required to be atomic terms.
$[t]^{\rightarrow}_k$	Term t encrypted with k using a public-key algorithm. Any term can be used as if it were a public key.
$\mathrm{sig}_k(t)$	Public-key signature of term t that is validated using key k. Since private keys of a key pair are never leaked, the attacker can only construct its own signatures $\mathrm{sig}_{\mathrm{pk}(\varepsilon)}(\ldots)$.

Figure 1: Message term constructors

2.3 The attacker model

In strand space models, attacker computations are represented by penetrator strands. In this paper, we use a term set closure operation, instead, to characterize attacker capabilities. Given a time ordering of nodes consistent with $\Rightarrow$, a minus node is *realizable iff* its message can be synthesized by the attacker from the set of messages sent in prior plus nodes. A semibundle is completable *iff* it has a node ordering in which every minus node is realizable.

We use the standard Dolev-Yao model for attacker computations. The attacker can simply pass along a sent message, or construct a new message by decomposing the previously sent messages into their parts and recombining those parts.

In the following definition we assume that T is a set of ground terms. If t is a term and S is a set of terms, we write $S \cup t$ rather than $S \cup \{t\}$ to avoid notational clutter, since it is unambiguous.

The *fake* operation $\mathcal{F}(T)$ is defined as the smallest set S containing T and closed under the following term set operators:

		Analysis	
$\phi_{\mathrm{split}}(S)$	=	$S \cup x \cup y$	if $[x, y] \in S$
$\phi_{\mathrm{pdec}}(S)$	=	$S \cup x$	if $[x]^{\rightarrow}_{\mathrm{pk}(\varepsilon)} \in S$
$\phi_{\mathrm{sdec}}(S)$	=	$S \cup x$	if $[x]^{\leftrightarrow}_y, y \in S$
		Synthesis	
$\phi_{\mathrm{pair}}(S)$	=	$S \cup [x, y]$	if $x, y \in S$
$\phi_{\mathrm{penc}}(S)$	=	$S \cup [x]^{\rightarrow}_y$	if $x, y \in S$
$\phi_{\mathrm{senc}}(S)$	=	$S \cup [x]^{\leftrightarrow}_y$	if $x, y \in S$
$\phi_{\mathrm{hash}}(S)$	=	$S \cup h(x)$	if $x \in S$
$\phi_{\mathrm{sig}}(S)$	=	$S \cup \mathrm{sig}_{\mathrm{pk}(\varepsilon)}(x)$	if $x \in S$
		Encryption hiding	
$\phi_{\mathrm{open}}(S)$	=	$S \cup [x]^{\leftrightarrow}_y$	if $\lceil x \rceil_y \in S$
$\phi_{\mathrm{hide}}(S)$	=	$S \cup \lceil x \rceil_y$	if $[x]^{\leftrightarrow}_y \in S$

Encryption hiding operators are a technical device needed to support analysis of constructed keys. They are explained in more detail in sections 4 and 5.

A ϕ operator can be applied to a term set whenever the set contains a "target" term with the appropriate structure. One can show that $\mathcal{F}$ is a closure operation: it is idempotent, monotonic (with respect to set inclusion), and extensive (a set is a subset of its closure).

$\mathcal{F}$ characterizes attacker capabilities in the sense that, for any non-empty term set T, $\mathcal{F}(T)$ is the (infinite) set of terms that can be constructed by the attacker from T. A received message m can be synthesized from a set T of sent messages if and only if it can be derived through $\mathcal{F}$, *i.e.*, $m \in \phi_n(\ldots\phi_1(T))$ for some $\phi_1, \ldots, \phi_n$.

Define $\Phi_{\text{synth}} = \{\phi_{\text{pair}}, \phi_{\text{penc}}, \phi_{\text{senc}}, \phi_{\text{hash}}, \phi_{\text{sig}}\}$, and $\Phi_{\text{analz}} = \{\phi_{\text{split}}, \phi_{\text{pdec}}, \phi_{\text{sdec}}\}$. In Paulson's method [12], $\mathcal{F}(T)$ could be expressed as $\mathsf{synth}(\mathsf{analz}(T))$ where synth and analz are, respectively, sequences of Φ_{synth} and Φ_{analz} operators, but this formula is not general enough when keys can be constructed. Consider, for example, $T = [a]^{\leftrightarrow}_{h(b)} \cup b$ where $a \in \mathcal{F}(T)$ but $a \notin \mathsf{synth}(\mathsf{analz}(T))$.

It is important to keep in mind that the characterization of the attacker capability as $\mathcal{F}(T)$ works only when (1) T is a set of ground terms, and (2) we have chosen a time ordering of nodes to establish which sent messages are "prior" to a received message and are therefore included in T.

It is not difficult to see how any operator assumed available to the Dolev-Yao attacker can be represented either as one of the ϕ term set operators in the definition of $\mathcal{F}$, or a penetrator strand in the conventional strand space model. The formal presentation of the attacker model does not, in itself, imply any difference in attacker capabilities between the term closure approach and the penetrator strand approach. With respect to the class of attacks that can be represented as a reachability problem for an instantiation of a single protocol trace (*e.g.*, secrecy and authentication), the two approaches are equivalent - if an attack can be discovered by one, it can be discovered by the other. Moreover, our model supports "programmable" attacker capabilities by changing ϕ operators in a way that is similar to penetrator strands. Extending the model with new message term constructors and corresponding ϕ operators will, however, affect our ability to solve the generated constraint sets.

2.4 Secrecy and authentication goals

A *secrecy* goal states that some designated message should not be made public. Compromise of a secret message can be detected by adding an artificial secret reception strand to the semibundle. In the case of NSL, if the responder wants to keep N'_B secret, we would add the one-node strand $-N'_B$ to the semibundle. Then the problem of determining whether the secret is compromised is equivalent to deciding if the semibundle with the secret reception strand is reachable (completable in our terminology).

Secrecy is violated if the secret is made public at any time, not just after all honest strands have completed normally. Thus, to analyze secrecy, one must consider semibundles in which role strands terminate prematurely. "Secret" is a relative term, in the sense that there is no security violation if a "secret" is generated by, or knowingly given to, the attacker. In order to associate a secret with honest principals, we instantiate the secret and the principals with symbolic constants. In the NSL example, we supply constants in the responder strand for the responder $B' = b$, the initiator $A' = a$ with whom b intends to share his secret, and the secret nonce $N'_B = n_b$.

In proof terms, this is a skolemization step. A proof of reachability or non-reachability applies to all possible values of the constants, subject to the implicit assumption that constants with different names are unequal. Skolemization is used in general to instantiate a nonce in the strand that generates the nonce (or other session-specific data).

In strand space models, *authentication* is typically expressed as the unreachability of a bundle that contains a strand that receives a message to be authenticated but does not contain another legitimate strand that sends it [16, 14]. A goal of this kind can be tested by a procedure to determine completability of semibundles that contain the authenticated strand but no authenticating strand.

2.5 The origination assumption

In our model, strands in a semibundle satisfy the *origination assumption*, namely, that a variable always occurs for the first time in any strand in a minus node. This assumption helps us to state and prove secrecy goals, and it plays a role in proving completeness of the decision procedure. It is satisfied for nonces and session secrets because of skolemization. For non-secret session parameters, such as principals, we can make it true by prefixing a strand with an artificial received message containing the variables that would otherwise be sent first. Because they are variables, this does not constrain their values, and because they are not secret, exposing them in a message does no harm and does not, in principle, affect implementability of the attack.

In our NSL example, the initiator strand contains the variables A for the initiator principal, and B for the responder chosen by that principal. We add the node $-[A, B]$ to the beginning of the node sequence, so that it satisfies the origination assumption.

3. CONSTRAINT GENERATION

Constraints are created from a node sequence consistent with the given semibundle, obtained by interleaving the strands in any of the possible ways. The result may be viewed as a singleton semibundle. For example, one possible merge of the NSL initiator and responder strands is:

$$-[A, B] + [A, n_a]^{\rightarrow}_{\text{pk}(B)} - [a, N_A]^{\rightarrow}_{\text{pk}(b)} + [N_A, n_b, b]^{\rightarrow}_{\text{pk}(a)} \ldots$$
$$\ldots - [n_a, N_B, B]^{\rightarrow}_{\text{pk}(A)} + [N_B]^{\rightarrow}_{\text{pk}(B)} - n_b$$

This node sequence includes the secret reception strand consisting of the single node $-n_b$, and omits the last responder node $-[n_b]^{\rightarrow}_{\text{pk}(b)}$ because it cannot affect the outcome.

3.1 Enumeration of interleavings

Different interleavings give rise to different constraint sets, and we attempt to solve each constraint set until we find one that has a solution or show that none of them are satisfiable. The number of possible interleaved node sequences can be very large. In general, the number of interleavings of a sequence of length m and one of length n (if the elements are distinct) is $\binom{m+n}{m}$, yielding an exponential number of cases. However, it is not necessary to consider all possible interleavings, because some interleavings are *dominated* by others, in the sense that any solution to one is a solution to the other. Dominated interleavings are redundant, and an optimization technique would eliminate them. For example, one can show that once a send node is enabled, there is no need to consider interleavings in which it is delayed until after later send or receive nodes. With this optimization, the number of interleavings is determined by the number of receive nodes alone. Further optimizations are possible as well. Since there are cases where different interleavings lead to incompatible constraint sets with different solution characteristics, we must deal with particular interleavings (rather than just the partial order of nodes given by a semibundle) before proceeding to the constraint solving phase.

3.2 Generation of a constraint set

A *constraint* is a pair $m : T$ where m is a term and T is a term set. The term set represents the set of messages known to the attacker. The constraint $m : T$ asserts that the attacker must be able to synthesize term m from the term set T.

A sequence of constraints and term sets is constructed from the sequence of nodes. Each plus node expands the last term set with its message, and each minus node creates a constraint $m : T$ where m is the message in the node and T is the last term set.

The first term set T_0 contains ground terms assumed known to the attacker. This should include constants representing principals, including at least the attacker's identity ε, and their public keys, including at least $\mathrm{pk}(\varepsilon)$. For example, we could have T_0 = {*alice*, *bob*, *srv*, ε, pk(*srv*), $\mathrm{pk}(\varepsilon)$} or $T_0 = \{a, b, \mathrm{pk}(a), \mathrm{pk}(b), \varepsilon, \mathrm{pk}(\varepsilon)\}$. The first example assumes that the server *srv* will deliver public keys on request, or that they are made available in the protocol through certificates or some other way so that they need not be known initially.

The node sequence above generates the following term sets and constraints:

$$\begin{aligned}
[A, B] &: T_0 = \{a, b, \varepsilon, \mathrm{pk}(\varepsilon)\} \\
[a, N_A]^{\rightarrow}_{\mathrm{pk}(b)} &: T_1 = T_0 \cup \{[A, n_a]^{\rightarrow}_{\mathrm{pk}(B)}\} \\
[n_a, N_B, B]^{\rightarrow}_{\mathrm{pk}(A)} &: T_2 = T_1 \cup \{[N_A, n_b, b]^{\rightarrow}_{\mathrm{pk}(a)}\} \\
n_b &: T_3 = T_2 \cup \{[N_B]^{\rightarrow}_{\mathrm{pk}(B)}\}
\end{aligned}$$

Note that the term sets are non-decreasing in this order, so that $T_i \subseteq T_{i+1}$ for all i. This is a special case of an invariant property called *monotonicity* in Section 5.1.

Let σ be a substitution of ground terms for all the variables, and let $C = \{m_i : T_i\}$ be a set of constraints. We say that σ is a *solution* of C, or σ *satisfies* C, if $\sigma \vdash C$ by the definition

$$\sigma \vdash C \quad \stackrel{\triangle}{=} \quad (\forall i)\, \sigma m_i \in \mathcal{F}(\sigma T_i)$$

Deciding satisfiability of the constraint set induced by the protocol requires reasoning about $\mathcal{F}$ sets that can be generated from a set of arbitrary terms, possibly involving variables.

As a shorthand, in the context of a particular C, we'll write $T_1 \preceq T_2$ if $(\forall\, \sigma \vdash C)\ \mathcal{F}(\sigma T_1) \subseteq \mathcal{F}(\sigma T_2)$. If $T_1 \preceq T_2$ and $T_2 \preceq T_1$, then $T_1 \cong T_2$.

4. SOLVING THE CONSTRAINT SET

In general, a constraint set is solved by the reduction procedure in which each application of a reduction rule replaces or eliminates some constraint. (We will often refer to a constraint sequence as a constraint set when we do not need to emphasize the ordering.) A sequence of reductions terminates successfully when the constraint set is reduced to a *simple* set, in which the left side of each remaining constraint, if any are left, is a variable. Such sets are always satisfiable (see section 4.4). A sequence of reductions can also terminate unsuccessfully by producing a constraint set that is neither simple, nor reducible.

A constraint set may be reducible in more than one way. The reduction procedure therefore creates a directed tree rooted in the initial constraint set C_0. Set C_0 has a solution (*i.e.*, there exists a successful attack on the protocol) if at least one path in the tree leads to a simple constraint set.

The reduction procedure is terminating, sound and complete, as proved in section 5. Therefore, substitution σ is a solution of the initial constraint set if and only if it is a solution of the simple constraint set at the end of at least one path in the tree.

C := initial constraint sequence
$\sigma := \emptyset$
`repeat`
  `let` $c^* = m : T$ be the first constraint in C
    s.t. m is not a variable
  `if` c^* not found
    `output` **Satisfiable!**
  apply rule *(elim)* to c^* until no longer applicable
  $\forall r \in R$
    `if` r is applicable to C
      $\langle C'; \sigma' \rangle := r(C; \sigma)$
      create node with C'; add $C \rightarrow C'$ edge
      `push` $\langle C'; \sigma' \rangle$
  $\langle C; \sigma \rangle$:= `pop`
`until emptystack`

Figure 2: Reduction procedure P

4.1 Reduction procedure

A *reduction tree* has reduction states containing constraint sets as nodes, and instances of reduction rules as edges. The root of the tree is the initial constraint set induced by the protocol. The reduction tree is created by the *reduction procedure* **P** in Fig. 2 where *(elim)* is the variable elimination rule (see section 4.2) and R is the set of reduction rules (see section 4.3).

Procedure **P** finds the first constraint where the left side m is not a variable (m may contain variables inside terms). We will call the constraint selected by **P** *active*. It then applies rule *(elim)* repeatedly to remove all standalone variables from the term set on the right side of the active constraint. Then one reduction rule is applied, and the procedure is repeated. If more than one rule is applicable to the active constraint, the reduction tree branches. Reduction rules maintain the relative ordering of the constraints. This is necessary for variable elimination to be sound (see section 4.2).

The state of the reduction is represented by a pair $C; \sigma$ where C is the current constraint set and σ is a partial substitution for variables that occurred in the initial constraint set. The initial state is associated with a null substitution. If application of a reduction rule requires a substitution that instantiates some of the variables, we apply the substitution immediately to the entire constraint set and add it to σ (see rules *(un)* and *(ksub)* in section 4.3). The accumulated substitution is thus carried with the reduced constraint set along every path in the reduction tree. If the path terminates in a satisfiable constraint set, σ contains variable instantiations that the attacker has to make in order to stage a successful attack.

In the rule definitions below, we refer to all constraints $v_i : T_i$ preceding the active constraint $m : T$ as $C_<$, and to all constraints $m_j : T_j$ following $m : T$ as $C_>$.

4.2 Variable elimination

Rule *(elim)* removes a standalone variable from the term set of a constraint. **P** applies it as many times as necessary to eliminate all standalone variables from the term set of the active constraint.

$$\frac{C_<, m : v \cup T, C_>; \sigma}{C_<, m : T, C_>; \sigma} \quad \textit{(elim)} \qquad v \text{ is a variable}$$

This rule is formally justified by proposition 5.1. Informally, removing a standalone variable v from a term set T_m does not change $\mathcal{F}(\sigma T_m)$ for any $\sigma \vdash C$. By the origination assumption, each variable appears for the first time on the left side of some constraint. Since **P** selects as the active constraint the first constraint where the

$$\frac{C_<, m : T, C_> ; \sigma}{\tau C_<, \tau C_> ; \tau \cup \sigma} \quad \textit{(un)}$$
where $\tau = \text{mgu}(m, t), t \in T$;

$$\frac{C_<, [m_1, m_2] : T, C_> ; \sigma}{C_<, m_1 : T, m_2 : T, C_> ; \sigma} \quad \textit{(pair)}$$

$$\frac{C_<, h(m) : T, C_> ; \sigma}{C_<, m : T, C_> ; \sigma} \quad \textit{(hash)}$$

$$\frac{C_<, [m]^{\rightarrow}_k : T, C_> ; \sigma}{C_<, k : T, m : T, C_> ; \sigma} \quad \textit{(penc)}$$

$$\frac{C_<, [m]^{\leftrightarrow}_k : T, C_> ; \sigma}{C_<, k : T, m : T, C_> ; \sigma} \quad \textit{(senc)}$$

$$\frac{C_<, \text{sig}_{\text{pk}(\varepsilon)}(m) : T, C_> ; \sigma}{C_<, m : T, C_> ; \sigma} \quad \textit{(sig)}$$

$$\frac{C_<, m : [t_1, t_2] \cup T, C_> ; \sigma}{C_<, m : t_1 \cup t_2 \cup T, C_> ; \sigma} \quad \textit{(split)}$$

$$\frac{C_<, m : [t]^{\rightarrow}_{\text{pk}(\varepsilon)} \cup T, C_> ; \sigma}{C_<, m : t \cup T, C_> ; \sigma} \quad \textit{(pdec)}$$

$$\frac{C_<, m : [t]^{\rightarrow}_k \cup T, C_> ; \sigma}{\tau C_<, \tau m : \tau[t]^{\rightarrow}_k \cup \tau T, \tau C_> ; \tau \cup \sigma} \quad \textit{(ksub)}$$
where $\tau = \text{mgu}(k, \text{pk}(\varepsilon)), k \neq \text{pk}(\varepsilon)$

$$\frac{C_<, m : T \cup [t]^{\leftrightarrow}_k, C_> ; \sigma}{C_<, k : T \cup \lceil t \rceil_k, m : T \cup t \cup k, C_> ; \sigma} \quad \textit{(sdec)}$$

Note: $\lceil x \rceil_y$ unifies with $[x']^{\leftrightarrow}_{y'}$ *iff* $\exists \sigma$ s.t. $\sigma x = \sigma x', \sigma y = \sigma y'$

Figure 3: Reduction rules

left side is not a variable, it must be the case that $m : v \cup T$ is preceded by a constraint $v : T_v \in C_<$. We can show that $T_v \preceq T$, thus any term that might be used to instantiate v can instead be constructed directly from T.

4.3 Constraint reduction rules

Reduction rules are listed in Fig. 3. They should be understood as rewrite rules on the constraint set, and read from top to bottom. To facilitate explanation, we gave matching names to term set operators ϕ and reduction rules. Each reduction rule applies to the same term(s) as the corresponding term set operator.

Notice that analysis operators correspond to reduction rules that decompose some term in the term set on the right side of a constraint, while synthesis operators correspond to rules that decompose the term on the left of a constraint.

4.3.1 Unification

The unification rule attempts to recognize m as a member of T, by unifying m with some non-variable term $t \in T$, using the most general unifier. Informally, application of this rule represents "replay" of a term known to the attacker. For example, the attacker can replay an encrypted term even if it has not been able to break the encryption. Different successful choices for t result in different branches in the reduction tree. A successful unification may cause one or more variables to be instantiated, in both m and t, and this substitution is applied to every constraint in the set. A successful unification eliminates the current constraint.

The unification rule *(un)* is applied only to constraints $m : T$ where m is not a variable due to the way **P** selects the active constraint. Note that T does not contain any standalone variables since **P** applies rule *(elim)* to $m : T$ before applying any reduction rule, including *(un)*. Unification does not distinguish $\lceil \rceil$ and $[\,]^{\leftrightarrow}$ terms, *i.e.*, $\lceil x \rceil_y$ unifies with $[x']^{\leftrightarrow}_{y'}$ *iff* x unifies with x', and y with y'.

Since σ has already been applied to the constraint set, neither m, nor t contains any variables in the domain of σ, thus the domains of τ and σ are disjoint. If the most general unifier $\text{mgu}(m, t)$ does not exist, the rule is not applicable. Note that if m is a constant, the rule will succeed only if $m \in T$.

4.3.2 Decomposition

Decomposition rules *(pair)*, *(hash)*, *(penc)*, *(senc)*, *(sig)* model the case when term m can be constructed from components which are synthesizable from terms in T. Intuitively, the rules should be read "backwards." For example, rule *(penc)* can be informally understood as "one of the ways the attacker can construct term $[m]^{\rightarrow}_k$ is by constructing terms m and k, and then encrypting m with k." Note that the attacker can construct only its own public-key signature.

4.3.3 Analysis

Analysis rules *(split)* and *(pdec)* attempt to break up terms on the right side of the constraint as far as possible without variable instantiation. If a term is encrypted with a public key which does not belong to the attacker, it cannot be decrypted since our model assumes that private keys are never leaked. Symmetric-key decryption is handled by the *(sdec)* rule.

4.3.4 Key substitution

Application of the key substitution rule *(ksub)* corresponds to the case when the attacker decrypts a term encrypted with a public key, *i.e.*, the right side of the active constraint must contain a $[t]^{\rightarrow}_k$ term. The rule is applicable only if term k in the key position unifies nonidentically with the attacker's public key $\text{pk}(\varepsilon)$ (the case when $k = \text{pk}(\varepsilon)$ is covered by the *(pdec)* rule). The attacker can only decrypt terms encrypted with its own public key since it is assumed that the private key of a key pair is never leaked. If k does not unify with $\text{pk}(\varepsilon)$, this means that term t is not encrypted with the attacker's public key, and the rule does not apply.

The domains of τ and σ are disjoint since σ has already been applied to the constraint set. Note that successful application of rule *(ksub)* enables rule *(pdec)* which can replace $[x]^{\rightarrow}_{\text{pk}(\varepsilon)}$ by x on the right side of the current constraint as well as all those containing terms encrypted with k before the substitution.

4.3.5 Symmetric-key decryption

The symmetric-key decryption rule *(sdec)* can be applied when the right side of the active constraint contains a term encrypted with a symmetric key. This corresponds to the case when the attacker succeeds in decrypting a symmetrically encrypted term by synthesizing the right key.

As far as unification and satisfiability are concerned, the special term $\lceil t \rceil_k$ is indistinguishable from $[t]^{\leftrightarrow}_k$. Its purpose is purely

technical: to "hide" the symmetrically encrypted term $[t]_k^{\leftrightarrow}$ in order to avoid subsequent application of the same rule to the newly added constraint $k : T_k$. The intuition behind this is that decrypting terms encrypted with k is never necessary in order to construct k. The term as a whole may still be necessary. Consider constraint $k : \lceil t \rceil_k, [k]_{[t]_k^{\leftrightarrow}}^{\leftrightarrow}$ where the entire term $\lceil t \rceil_k$ must be used, *without being decrypted itself*, to decrypt another term and extract k. Note that $[t]_k^{\leftrightarrow}$ is replaced in the term set of the original constraint by t and k. Addition of k to the term set is sound if constraint $k : T_k$ is satisfiable, as proved in proposition 5.3.

4.4 Checking satisfiability

Every path in the reduction tree generated by procedure **P** terminates either in a constraint set to which no rule can be applied, or in a *simple* set C that has only variables on the left, *i.e.*, $C = v_1 : T_1, \ldots, v_n : T_n$. A simple constraint set is always satisfiable as long as the attacker has at least one constant in its initial term set. One can check by inspecting the reduction rules that such constants remain in the term set of every constraint. If c is such a constant, $\sigma = [c/v_1, \ldots, c/v_n]$ satisfies all constraints. In the future, we may wish to distinguish different types of constants. We will ensure then that the attacker knows one constant of each type.

We have not performed a detailed analysis of the complexity of the constraint solving algorithm. Rusinowitch and Turuani [13] demonstrated that the problem is NP-complete in a similar setting (free term algebra with arbitrary terms as symmetric encryption keys). The proof of NP-completeness in [13] relies on guessing the right substitution for variables and the right sequence of attacker operators that derives m from T_m for each constraint $m : T_m$. It is likely that while our algorithm has the same worst-case complexity, it is significantly more efficient in practice since in our case substitutions are performed only when they may possibly result in satisfying a constraint (rules *(un)* and *(ksub)*), and generation of the sequences of ϕ operators deriving m from T_m is driven by the structure of the terms.

5. SOUNDNESS AND COMPLETENESS

In this section, we prove that **P** terminates and that it preserves all solutions of C_0 without introducing any new ones.

5.1 Invariant properties of P

Let $\mathbf{C_P}$ be the set of all constraint sequences generated by **P**.

The origination assumption (see section 2.5) implies that, in the initial sequence of constraints, each variable appears for the first time on the left side of a constraint, and not in the right side of that constraint. This *origination property* is an invariant.

Theorem (Invariance of Origination) $\forall C \in \mathbf{C_P}, C$ *satisfies the origination property*.

Proof in Appendix A.1.

Consider constraint $m_v : T_v$ in which variable v appears for the first time. By the origination property, T_v does not mention v, or any variable that appears later than v. A constraint sequence is *monotonic*, if, for any constraint $m : T$ s.t. T mentions variable v, $\exists\, T_{\bar{v}} \subseteq T$ such that $T_{\bar{v}}$ does not mention v or any variable that appears later than v, and $T_v \preceq T_{\bar{v}}$. Furthermore, $T_0 \subseteq T$.

Monotonicity captures the fact that the attacker never forgets information. Every message received by the attacker can only add to its knowledge and cannot possibly remove any terms it already knows. If at some point the attacker had access to terms in T_v, then at any later point these terms, possibly transformed in a way that preserves $\mathcal{F}(T_v)$, are still available to the attacker.

Theorem (Invariance of Monotonicity) $\forall C \in \mathbf{C_P}, C$ *is monotonic*.

Proof in Appendix A.2.

PROPOSITION 5.1. *If $m : T \cup v$ is the active constraint, then $T \cong T \cup v$.*

Clearly, $T \preceq T \cup v$, so it suffices to show that $T \cup v \preceq T$. By the origination property, there exists an earlier constraint $m_v : T_v$ where v appears for the first time in m_v. It must be the case that $m_v = v$, since constraints earlier than the active constraint have standalone variables on the left.

Suppose $\sigma \vdash C$. We must show that $\mathcal{F}(\sigma(T \cup v)) \subseteq \mathcal{F}(\sigma T)$. It suffices to show that $\sigma(T \cup v) \subseteq \mathcal{F}(\sigma T)$, since $\mathcal{F}$ is idempotent. Since $\sigma T \subseteq \mathcal{F}(\sigma T)$, we just need $\sigma v \in \mathcal{F}(\sigma T)$.

By monotonicity of the constraint sequence C, $\exists\, T_{\bar{v}} \subseteq T \cup v$ such that $T_{\bar{v}}$ does not mention v and $T_v \preceq T_{\bar{v}}$. In particular, $T_{\bar{v}} \subseteq T$, so $\sigma v \in \mathcal{F}(\sigma T_v) \subseteq \mathcal{F}(\sigma T)$.

PROPOSITION 5.2. *If $T_m = T \cup [t]_k^{\leftrightarrow}$, then $T_m \cong T \cup \lceil t \rceil_k$*

Follows immediately from the fact that $\mathcal{F}$ is closed under ϕ_{open} and ϕ_{hide}.

PROPOSITION 5.3. *Suppose $[t]_k^{\leftrightarrow} \in T$. If $\exists\, \sigma$ s.t. $\sigma \vdash k : T_k$ and $T_k \preceq T$, then $T \cong (T \setminus [t]_k^{\leftrightarrow}) \cup t \cup k$*

Let $T = \hat{T} \cup [t]_k^{\leftrightarrow}$. If $\exists\, \sigma$ s.t. $\sigma \vdash k : T_k$ and $T_k \preceq T$, then $\sigma k \in \mathcal{F}(\sigma T)$ and, since $\mathcal{F}$ is closed, $\mathcal{F}(\sigma T) = \mathcal{F}(\sigma \hat{T} \cup [\sigma t]_{\sigma k}^{\leftrightarrow} \cup \sigma k) = \mathcal{F}^*$. Since $\mathcal{F}^*$ is closed under ϕ_{sdec} and ϕ_{senc}, $\mathcal{F}^* = \mathcal{F}(\sigma \hat{T} \cup [\sigma t]_{\sigma k}^{\leftrightarrow} \cup \sigma k \cup \sigma t) = \mathcal{F}(\sigma \hat{T} \cup \sigma k \cup \sigma t)$. By definition, $T \cong (T \setminus [t]_k^{\leftrightarrow}) \cup t \cup k$.

5.2 Termination

The termination measure of a constraint set is a tuple (N_v, N_s) where N_v is the number of distinct variables and N_s is a special expansion measure. Tuples are ordered lexicographically.

To define the expansion measure, first define the structure size $|m|$ of a term m to be the number of operator applications plus the number of constants and variables in it (the number of nodes in the parse tree). The expansion measure of a constraint set is the sum of the expansion measures of the constraints. The expansion measure of a constraint $m : T$ is $|m| \cdot \chi(T)$ where χ is defined as follows:

$$\begin{array}{ll} \chi(t) = 2 \quad \text{if } t \text{ is a var or constant} & \chi([t]_k^{\rightarrow}) = \chi(t) + 1 \\ \chi(\{t_1, ..., t_n\}) = \chi(t_1) \cdots \chi(t_n) & \chi(\lceil t \rceil_k) = 1 \\ \chi([t_1, t_2]) = \chi(t_1)\chi(t_2) + 1 & \chi(\text{sig}_k(t)) = \chi(t) + 1 \\ \chi([t]_k^{\leftrightarrow}) = \chi(t)\chi(k) + |k| + 1 & \chi(h(t)) = \chi(t) + 1 \end{array}$$

We have to show that each rule reduces the termination measure. Checking the rules one by one, we see that: *(elim)* reduces N_s by eliminating a factor $\chi(v) = 2$; *(un)* either eliminates a variable by substitution and hence reduces N_v, or matches a constant on the left without a substitution, which leaves N_v unchanged but reduces N_s by eliminating the constraint; *(sig)*, *(pair)*, *(hash)*, *(penc)*, and *(senc)* reduce N_s by leaving T alone but decomposing the left side into components with a smaller structure sum; *(split)* and *(pdec)* reduce N_s by decreasing a factor $\chi(t)$; and *(ksub)* instantiates a variable and thus reduces N_v. The most interesting case is *(sdec)*, which replaces $m : T \cup [t]_k^{\leftrightarrow}$ with expansion measure $|m|\chi(T)(\chi(t)\chi(k)+|k|+1)$ by $k : T \cup \lceil t \rceil_k$ and $m : T \cup t \cup k$ with total expansion measure $|k|\chi(T) + |m|\chi(T)\chi(t)\chi(k)$. This measure is smaller because that $|k| < |m|(|k| + 1)$. Hence, ***P** terminates.*

5.3 Soundness

A constraint reduction procedure is *sound* if reduction rules do not introduce new solutions. To prove soundness, it is sufficient to show for every rule r that $\sigma \vdash r(C)$ implies $\sigma \vdash C$. Each reduction rule is sound (proof in Appendix A.3). By induction over the length of the reduction sequence, ***P** is sound.*

5.4 Completeness

The basic idea of completeness is to show that any solution of the initial constraint set is preserved along at least one path in the reduction tree. If σ is a substitution such that $\sigma \vdash C_0$, then **P** is complete if, among all the simple, satisfiable constraint sets produced by **P**, there is at least one set C such that $\sigma \vdash C$.

Our proof of completeness is quite delicate. In particular, completeness does *not* hold inductively for any constraint set, *i.e.*, it is not always the case that if $\sigma \vdash C$, then there is a rule r such that $\sigma \vdash r(C)$. Consider constraint $c = h(k) : \lceil k \rceil_{h(k)} \cup h(k)$. Applying reduction rule *(hash)* to this constraint, we obtain constraint $c' = k : \lceil k \rceil_{h(k)} \cup h(k)$. This constraint is satisfiable since $k \in \mathcal{F}(\lceil k \rceil_{h(k)} \cup h(k))$ by applying ϕ_{open} on $\lceil k \rceil_{h(k)}$ to obtain $[k]^{\leftrightarrow}_{h(k)}$, and then ϕ_{sdec} on $[k]^{\leftrightarrow}_{h(k)}$ and $h(k)$ to obtain k. Unfortunately, there is no reduction rule in section 4.3 that can be applied to c'. The problem arises because $\lceil x \rceil_y$ can be "opened" by ϕ_{open} and subsequently decrypted when computing $\mathcal{F}$, but there is no corresponding reduction rule that could be applied to the constraint.

Observe, however, that completeness is *not* violated in this example. There exists *another* rule that is applicable to c, namely, *(un)*, which unifies $h(k)$ with $h(k)$ and successfully eliminates the constraint. This observation is the intuition behind our proof of completeness.

Theorem (Completeness) *For any substitution σ such that $\sigma \vdash C_0$, **P** will generate a simple constraint set C such that $\sigma \vdash C$.*

Details of the proof can be found in Appendix A.4. The proof consists of two parts. First, we show that completeness holds inductively for any constraint $m : T$ such that m, T do not contain $\lceil x \rceil_y$ terms. Then, we show that the *first* time a constraint $m : \hat{T}$ such that $\hat{T}$ contains a $\lceil x \rceil_y$ term appears in the reduction sequence, there exists, given any solution $\sigma \vdash m : \hat{T}$, an applicable sequence of reduction rules $r_1, \ldots, r_m$, $r_i = r_i(\sigma)$ such that $\sigma \vdash m' : T' = r_m(\ldots r_1(m : \hat{T}))$ where T' does not contain $\lceil x \rceil_y$ terms.

Informally, the proof can be understood as follows. Procedure **P** may generate "bad" states which are satisfiable but not reducible, thus violating completeness. However, "bad" states cannot appear in an arbitrary place in the reduction tree. We show that they appear only in branches rooted in "bad-root" states of a particular type, namely, those generated by the *(sdec)* rule. For each "bad-root" state we prove that, given any solution, there always exists at least one branch out of the state that preserves this solution and leads to a "good" state. This will guarantee completeness, since any solution of the *initial* constraint set (but not necessarily of every set generated by **P**) is preserved along at least one path in the reduction tree.

6. EXAMPLES

The NSL protocol, as modified, was analyzed using a Prolog program based on the constraint reduction rules in Section 4.3. The program was given several semibundles to look at, with as many as four legitimate strands. On a semibundle with two responder strands, it found an interleaving with a solvable constraint set. One constraint reduction path leading to a solution (and hence an attack) is traced below.

Consider the interleaving of strands for responders b and some A, in which b expects to share n_b with a particular a. The last reply in each strand has been omitted for simplicity, but the secret-reception strand $-n_b$ is added to test secrecy of n_b. The actual analyzed semibundle had more strands, and they were complete.

$-[a, N_A]^{\rightarrow}_{\text{pk}(b)}$	*to b*
$+[N_A, n_b, b]^{\rightarrow}_{\text{pk}(a)}$	*from b to a*
$-[B, N_B]^{\rightarrow}_{\text{pk}(A)}$	*to any A from any B*
$+[N_B, n_a, A]^{\rightarrow}_{\text{pk}(B)}$	*from A to B*
$-n_b$	*secret reception*

The constraint set from this interleaving is:

$$
\begin{array}{lrcl}
1. & [a, N_A]^{\rightarrow}_{\text{pk}(b)} & : & T_0 = \{a, b, \varepsilon, \text{pk}(a), \text{pk}(b)\} \\
2. & [B, N_B]^{\rightarrow}_{\text{pk}(A)} & : & T_1 = T_0 \cup \{[N_A, [n_b, b]]^{\rightarrow}_{\text{pk}(a)}\} \\
3. & n_b & : & T_1 \cup \{[N_B, [n_a, A]]^{\rightarrow}_{\text{pk}(B)}\}
\end{array}
$$

We will follow one path leading to a solution. Note that we are treating concatenation as a binary right-associative operation. First, apply *(penc)* to (1):

$$
\begin{array}{lrcl}
1.1. & \text{pk}(b) & : & T_0 \\
1.2. & [a, N_A] & : & T_0 \\
2. & [B, N_B]^{\rightarrow}_{\text{pk}(A)} & : & T_1 = T_0 \cup \{[N_A, [n_b, b]]^{\rightarrow}_{\text{pk}(a)}\} \\
3. & n_b & : & T_1 \cup \{[N_B, [n_a, A]]^{\rightarrow}_{\text{pk}(B)}\}
\end{array}
$$

Eliminate (1.1) with *(un)* and expand (1.2) with *(pair)*:

$$
\begin{array}{lrcl}
1.2.1. & a & : & T_0 \\
1.2.2. & N_A & : & T_0 \\
2. & [B, N_B]^{\rightarrow}_{\text{pk}(A)} & : & T_1 = T_0 \cup \{[N_A, [n_b, b]]^{\rightarrow}_{\text{pk}(a)}\} \\
3. & n_b & : & T_1 \cup \{[N_B, [n_a, A]]^{\rightarrow}_{\text{pk}(B)}\}
\end{array}
$$

Eliminate (1.2.1) with *(un)* and skip (1.2.2) because it has a variable on the left. Apply *(un)* to (2) with the substitutions $B \mapsto N_A$, $N_B \mapsto [n_b, b]$ and $A \mapsto a$, eliminating (2).

$$
\begin{array}{lrcl}
1.2.2. & N_A & : & T_0 \\
3. & n_b & : & T_1 \cup \{[n_b, b], [n_a, a]]^{\rightarrow}_{\text{pk}(N_A)}\}
\end{array}
$$

Finally, apply *(ksub)* to (3) with $N_A \mapsto \varepsilon$. It should be clear after this that n_b will be exposed and the solution can be finished up easily. Installing the substitutions into the original semibundle yields the attack.

The attack requires two somewhat implausible but not impossible type confusions: ε in the first message is occupying a nonce field, and $[n_b, b]$ in the first message of the a strand is also occupying a nonce field. This could work if agent names are the same length as nonces and the protocol could handle two sizes of nonce (single or double). The point is not that this is a realistic attack, but that it illustrates the power of the analysis technique to find surprising results, in this case by permitting a type flaw. Protocols can also be encoded in such a way as to reflect type protection, if the implementation is believed to work that way.

To illustrate how the algorithm handles constructed keys, we present partial analysis of a toy, faulty mutual authentication protocol inspired by Gong's mutual authentication protocol [8] (we did not discover any bugs in the original protocol).

$$
\begin{array}{l}
A \rightarrow B : A, N_A \\
B \rightarrow S : A, B, N_A, N_B \\
B \leftarrow S : N_S, [h([N_S, N_A, B, K_A])]^{\leftrightarrow}_{h([N_S, N_B, A])} \\
A \leftarrow B : N_S, h(h([N_S, N_A, B, K_A])) \\
A \rightarrow B : h(h([N_S, N_A, B, K_A]), N_S)
\end{array}
$$

The goal of the protocol is to preserve the secrecy of the key $K_{AB} = h([N_S, N_A, B, K_A])$ shared between A and B.

One of the possible interleavings of the protocol and the secret reception strand $A \leftarrow K_{AB}$ gives rise to the following constraint:

$$K_{AB} \quad : \quad \{A, B, N_A, N_B, N_S, [K_{AB}]^{\leftrightarrow}_{h([N_S,N_B,A])}\}$$

Rule *(sdec)* transforms this constraint into:

$$\begin{aligned} h([N_S, N_B, A]) \quad &: \quad \{A, B, N_A, N_B, N_S, \\ &\qquad \lceil K_{AB} \rceil_{h([N_S,N_B,A])}\} \\ K_{AB} \quad &: \quad \{A, \ldots, N_S, K_{AB}, h([N_S, N_B, A])\} \end{aligned}$$

Rules *(hash)* and *(un)* dispose of the first constraint (this corresponds to the fact that $h([N_S, N_B, A])$ can be constructed by the attacker who knows A, N_B, and N_S) and *(un)* disposes of the second constraint. Therefore, the constraint set is satisfiable, proving that the secret reception semibundle is reachable.

7. CONCLUSION

By using the strand space model only for honest processes and the term set closure characterization of the attacker, our model achieves a clean transition from the process model to the constraint solving problem. Models that put individual attacker actions (such as in penetrator strands) in the process side must mix the two, because they cannot predict which actions the attacker will perform. In terms of attacker capabilities, the term set closure characterization of the attacker results in the same (infinite) set of synthesizable messages as penetrator strands.

Using the free term algebra simplifies the model, enabling us to handle constructed keys even in cases of self-encryption. However, without cryptographic reduction rules, we cannot handle protocols where both keys in a public-private key pair are used explicitly. A free algebra also fails to represent the properties of encryption operations with associative and commutative characteristics, such as `xor` and Diffie-Hellman exponentiation. We are currently investigating how the constraint solving algorithm presented in this paper may be extended to support such operations.

The finite semibundle node-merge generation and the constraint reduction rules lend themselves well to implementation in Prolog, with its built-in depth-first search strategy and unification. We have implemented the decision procedure in XSB (SUNY Stony Brook) Prolog, and it often runs in a small fraction of a second on the examples we have tried, even when the vulnerability search fails. The program is less than three pages. The approach can be extended in a natural way to unbounded process analysis by iteratively adding strands to the initial finite set, though there is no guarantee of termination if this is done.

8. REFERENCES

[1] AMADIO, R., AND LUGIEZ, D. On the reachability problem in cryptographic protocols. In *CONCUR* (2000), vol. 1877 of *LNCS*, Springer, pp. 380–394.

[2] AMADIO, R., LUGIEZ, D., AND VANACKÈRE, V. On the symbolic reduction of processes with cryptographic functions. Tech. Rep. 4147, INRIA, March 2001.

[3] BOREALE, M. Symbolic analysis of cryptographic protocols in the spi-calculus. In *ICALP* (2001). To appear.

[4] CERVESATO, I., DURGIN, N., LINCOLN, P., MITCHELL, J., AND SCEDROV, A. Relating strands and multiset rewriting for security protocol analysis. In *13th IEEE Computer Security Foundations Workshop* (2000), pp. 35–51.

[5] DURGIN, N., LINCOLN, P., MITCHELL, J., AND SCEDROV, A. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols* (1999), FLOC.

[6] FIORE, M., AND ABADI, M. Computing symbolic models for verifying cryptographic protocols. In *14th IEEE Computer Security Foundations Workshop* (2001), pp. 160–173.

[7] FREIER, A., KARLTON, P., AND KOCHER, P. The SSL protocol. Version 3.0. `http://home.netscape.com/eng/ssl3/`.

[8] GONG, L. Using one-way functions for authentication. *Computer Communication Review 19*, 5 (1989), 8–11.

[9] HEINTZE, N., AND TYGAR, J. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering 22*, 1 (1996), 16–30.

[10] HUIMA, A. Efficient infinite-state analysis of security protocols. In *Workshop on Formal Methods and Security Protocols* (1999), FLOC.

[11] LOWE, G. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS* (1996), vol. 1055 of *LNCS*, Springer, pp. 147–166.

[12] PAULSON, L. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security 6*, 1 (1998), 85–128.

[13] RUSINOWITCH, M., AND TURUANI, M. Protocol insecurity with finite number of sessions is NP-complete. In *14th IEEE Computer Security Foundations Workshop* (2001), pp. 174–190.

[14] SONG, D. Athena: a new efficient automatic checker for security protocol analysis. In *12th IEEE Computer Security Foundations Workshop* (1999), pp. 192–202.

[15] THAYER, F., HERZOG, J., AND GUTTMAN, J. Honest ideals on strand spaces. In *11th IEEE Computer Security Foundations Workshop* (1998), pp. 66–78.

[16] THAYER, F., HERZOG, J., AND GUTTMAN, J. Strand spaces: Why is a security protocol correct? In *IEEE Symposium on Security and Privacy* (1998), pp. 160–171.

APPENDIX

A. PROOFS

A.1 Invariance of origination

The origination property for a constraint sequence states that each variable appears for the first time on the left side of a constraint, and not in the right side of that constraint. This property can only be violated by a reduction rule which, given a constraint $m : T$, changes it to $m' : T'$ such that T' contains a variable v' but there is no preceding constraint that has v' only on the left side.

With the exception of rules *(un)* and *(ksub)*, constraint reduction rules do not instantiate existing variables or introduce new variables or change the order of constraints. Therefore, they cannot violate the origination property. Rules *(un)* and *(ksub)* may introduce a new variable on the right side as a result of applying substitution τ. Consider constraint $m : T$ such that T mentions variable v, and substitution τ such that τv mentions some other variable v'. If the origination property is true for the constraint set to which the rule is applied, then there exists another constraint $m(v) : T_v$ that precedes $m : T$ in the chronologically ordered constraint list such that v is mentioned in $m(v)$ but not in T_v. Consider two cases.

$v' \in T_v$ By the origination property for the constraint set before rule application, there exists $m(v') : T_{v'}$ preceding $m(v) : T_v$ such that v' is mentioned in $m(v')$ but not in $T_{v'}$. But this

constraint must also precede $m : T$ and, therefore, $m' : T'$, since neither *(un)*, nor *(ksub)* changes the relative order of constraints. Application of τ does not substitute v', otherwise v' would not appear in τT. Therefore, v' is mentioned in $\tau m(v')$ but not in $\tau T_{v'}$, thus there exists a constraint preceding $m' : T'$, namely, $\tau m(v') : \tau T_{v'}$ that mentions v' only on the left side.

$v' \notin T_v$ Then $\tau m(v)$ mentions v', but τT_v does not. Therefore, there exists a constraint preceding $m' : T'$, namely, $\tau m(v) : \tau T_v$ such that v' is mentioned only on the left side.

By induction over the length of the reduction sequence, the origination property is true for all constraints generated by **P**.

A.2 Invariance of monotonicity

The initial constraint set is monotonic by simple term set inclusion. To prove that all constraint sets produced by **P** are monotonic, it is sufficient to show that monotonicity is invariant with respect to every reduction rule from section 4.3.

Rule *(elim)* eliminates one of v_i from T but does not affect $T_{\bar{v}_i}$.

Rules *(split)* and *(pdec)* do not introduce new variables and do not affect $\mathcal{F}(\sigma T_{\bar{v}_i})$ for any $v_i, \sigma \vdash C$. For example, if $[t]^{\rightarrow}_{\mathrm{pk}(\varepsilon)} \in T_{\bar{v}_i}$, then $T_{\bar{v}_i} \cong (T_{\bar{v}_i} \setminus [t]^{\rightarrow}_{\mathrm{pk}(\varepsilon)}) \cup t$.

Rules *(pair)*, *(hash)*, *(penc)*, *(senc)*, *(sig)* do not affect T at all.

Rule *(ksub)* does not introduce any new variables or terms to T and, therefore, cannot change $T_{\bar{v}_i}$.

Rule *(sdec)* does not introduce new variables. Suppose $[t]^{\leftrightarrow}_k \in T_{\bar{v}_i}$. Since $\mathcal{F}$ is closed under ϕ_{open} and ϕ_{sdec}, $\mathcal{F}(\sigma T_{\bar{v}_i})$ for any $\sigma \vdash k : T_k, C$ is not affected if $[t]^{\leftrightarrow}_k$ is replaced by $\lceil t \rceil_k$ or $t \cup k$. This follows from propositions 5.2 and 5.3.

Finally, consider rule *(un)*. Suppose τ includes substitution $v_j \mapsto t(v_i)$ for some variables v_i, v_j where $t(v_i)$ is an arbitrary term mentioning v_i. Even though τ may introduce v_i into some terms of T that did not mention v_i before, we'll prove that either these terms are not in $T_{\bar{v}_i}$, or τ, when applied to the entire constraint set, replaces v_j with $t(v_i)$ in some constraint c preceding $m(v_i) : T_i$ and makes *that* constraint the first constraint mentioning v_i. We'll also show that T must contain a superset of T_c which is not affected by τ, thus preserving monotonicity. Consider two cases.

$j \geq i$ By definition, $T_{\bar{v}_i}$ does not mention v_j, and τ does not introduce v_i to $T_{\bar{v}_i}$.

$j < i$ By the origination property, $\exists$ constraint $c = m(v_j) : T_j$ preceding $m(v_i) : T_i$ such that T_j does not mention v_i or v_j. By the induction hypothesis, the constraint set to which rule *(un)* is applied is monotonic. Therefore, $\exists\, T_{\bar{v}_j} \subseteq T$ such that $T_j \preceq T_{\bar{v}_j}$. Observe that $T_{\bar{v}_j} \subseteq T_{\bar{v}_i}$ since $T_{\bar{v}_j}$ contains all terms of T that do not mention $v_j, \ldots, v_i, \ldots, v_k$, while $T_{\bar{v}_i}$ contains all terms of T that do not mention $v_i, \ldots, v_k$ only. Substitution τ does not affect any terms in $T_{\bar{v}_j}$ since they do not mention v_j. Therefore, $T_{\bar{v}_j} \subseteq \tau T$.

After τ is applied to the constraint set, $c' = \tau c = \tau m(v_j) : \tau T_j = m(t(v_i)) : T_j$. Note that $m(t(v_i))$ mentions v_i. Therefore, c' and not $m_i : T_i$ is now the first constraint that mentions v_i. But $\exists\, T_{\bar{v}_j} \subseteq \tau T$ such that $T_{\bar{v}_j}$ does not mention $v_j, \ldots, v_i, \ldots, v_k$ and $\tau T_j = T_j \preceq T_{\bar{v}_j}$.

By induction over the length of the reduction sequence, all constraint sets generated by procedure **P** are monotonic.

A.3 Soundness

For rule *(elim)*, soundness follows from proposition 5.1.

Rules *(split)* and *(pdec)* are sound because $\mathcal{F}$ is closed under ϕ_{pair} and ϕ_{pdec}. For example, in case of rule *(pdec)*, $\mathcal{F}(\sigma T \cup \sigma t) = \mathcal{F}(\sigma T \cup \sigma t \cup \sigma[t]^{\rightarrow}_{\mathrm{pk}(\varepsilon)}) = \mathcal{F}(\sigma T \cup \sigma[t]^{\rightarrow}_{\mathrm{pk}(\varepsilon)})$. Therefore, if $\sigma m \in \mathcal{F}(\sigma t \cup \sigma T)$, then $\sigma m \in \mathcal{F}(\sigma[t]^{\rightarrow}_{\mathrm{pk}(\varepsilon)} \cup \sigma T)$.

For rule *(un)*, consider that if $\sigma \vdash \tau C_<, \tau C_>$, then $\sigma \cup \tau \vdash C_<, C_>$. Also, $\sigma \cup \tau \vdash m : t \cup \hat{T}$ because $\tau = \mathrm{mgu}(m, t)$, thus $\tau m = \tau t$ by definition of the most general unifier. Therefore, $\sigma \cup \tau \vdash C$.

For rules *(pair)*, *(hash)*, *(penc)*, *(senc)*, and *(sig)*, soundness follows from the fact that $\mathcal{F}$ is closed under the corresponding ϕ operator. For example, consider rule *(penc)*. If $\mathit{penc}(C) = C_<, k : T, m : T, C_>$ is satisfiable, then $\exists\, \sigma$ s.t. $\sigma \vdash C_<, C_>$ and $\sigma k \in \mathcal{F}(\sigma T), \sigma m \in \mathcal{F}(\sigma T)$. Since $\mathcal{F}$ is closed under ϕ_{penc}, $[\sigma m]^{\rightarrow}_{\sigma k} \in \mathcal{F}(\sigma T)$. Therefore, $\sigma \vdash C_<, [m]^{\rightarrow}_k : T, C_> = C$.

For rule *(ksub)*, if $\sigma \vdash \tau C_<, \tau m : \tau[t]^{\rightarrow}_k \cup \tau T, \tau C_m$, then $\sigma \cup \tau \vdash C_<, m : [t]^{\rightarrow}_k \cup T, C_> = C$.

Finally, consider $\sigma \vdash \mathit{sdec}(C) = C_<, k : T_k, m : (T \setminus [t]^{\leftrightarrow}_k) \cup t \cup k, C_>$ where $T_k = \lceil t \rceil_k \cup T$. Clearly, $\sigma \vdash C_<, C_>$. By proposition 5.2, $T_k \cong T$, thus $\sigma \vdash k : T$. Given $\sigma \vdash k : T$, $T \cong (T \setminus [t]^{\leftrightarrow}_k) \cup t \cup k$ by proposition 5.3. Therefore, $\sigma \vdash m : T$. It follows that $\sigma \vdash C_<, m : T, C_> = C$.

A.4 Completeness

This proof is long because it requires consideration of many cases. Because of space limitations, and in the interests of readability, what follows is a fairly detailed proof sketch.

A.4.1 Completeness without encryption hiding

Suppose $m : T_m$ is the active constraint and m, T_m contain no terms of $\lceil x \rceil_y$ form. For any $\sigma \vdash C = C_<, m : T_m, C_>$, we show that there is a rule r_σ such that (i) r_σ is applicable to C, and (ii) $\sigma \vdash r_\sigma(C)$. Note that the solution does not have to be preserved in *every* possible reduction. As long as in every state there is at least one rule that preserves the solution, completeness will hold. The proof that an applicable rule can always be found relies on the existence of a normal derivation for any term that can be constructed by the attacker.

A.4.1.1 Normal derivation.

For any ground term t and set of ground terms T, where neither t, nor T contain any occurrences of terms of $\lceil x \rceil_y$ form, we prove that if $t \in \mathcal{F}(T)$, then there exists a *normal derivation* of t from T which either ends with an operator from Φ_{synth}, starts with an operator from Φ_{analz}, or starts with a sequence of operators from Φ_{synth}, followed by ϕ_{sdec} which is applied to a term from T.

PROPOSITION A.1. *If $t \in \mathcal{F}(T)$ and neither t, nor T contain any occurrences of $\lceil x \rceil_y$ terms, then there exists a normal sequence $\phi_1, \ldots, \phi_n$ such that $t \in \phi_n(\ldots \phi_1(T))$. A sequence is* normal *iff one of the following conditions holds:*
- $\phi_n \in \Phi_{synth}$, or
- $\phi_1 \in \Phi_{analz}$, or
- $\phi_i = \phi_{sdec}$ for some i, $\phi_1, \ldots, \phi_{i-1} \in \Phi_{synth}$, and ϕ_i is applied to term $[x]^{\leftrightarrow}_y \in T$ for some x, y.

Suppose $t \in \mathcal{F}(T)$. Since $\mathcal{F}$ is defined as a closure of term set operators ϕ (see section 2.3), this means that either $t \in T$, or $t \in \phi_r(\ldots \phi_1(T))$ where each ϕ_i is one of the term set operators defining $\mathcal{F}$. For notational convenience, let $T_0 = T$, and let T_i stand for $\phi_i(\ldots \phi_1(T_0))$ for any i.

Step 1. First, we observe that t with no hidden terms can be derived without ϕ_{open} and ϕ_{hide} operators, since any operator application using a hidden encryption could be replaced by one using the corresponding ordinary encryption.

Step 2. Following Step 1, we obtain a sequence $\phi_1, \ldots, \phi_n$ s.t. $\forall i\ \phi_i \in \Phi_{\mathrm{synth}}$ or Φ_{analz}, and $t \in \phi_n(\ldots \phi_1(T_0))$. If $t \in T_0$, the

proposition holds. If $t \notin T_0$, the proposition can only be violated if, for some k, $\phi_k \in \Phi_{\text{analz}}$ and $\phi_j \in \Phi_{\text{synth}}$ for $j < k$.

Since $\phi_k \in \Phi_{\text{analz}}$, $\phi_k = \phi_{\text{split}}$, ϕ_{pdec}, or ϕ_{sdec}. First, consider the case when $\phi_k = \phi_{\text{split}}$ or ϕ_{pdec}, and let $\tilde{t} \in T_{k-1}$ be the term to which ϕ_k is applied. If $\tilde{t} \in T_0$, then ϕ_k could be moved up to the ϕ_1 position. Otherwise $\tilde{t}$ was created by one of the Φ_{synth} operations and the Φ_{analz} operation is redundant and can be removed.

Now, consider the case of $\phi_k = \phi_{\text{sdec}}$, applied to terms $[x]^{\leftrightarrow}_y, y \in T_{k-1}$ for some x, y. There are two possibilities for term $[x]^{\leftrightarrow}_y$: either $[x]^{\leftrightarrow}_y \in T_0$ (in this case the proof is complete), or $[x]^{\leftrightarrow}_y \in T_i \setminus T_{i-1}$ for some i. In this case, also, $[x]^{\leftrightarrow}_y$ was created from its components by one of the Φ_{synth} operations and the ϕ_{sdec} operation is redundant and can be removed.

A.4.1.2 Finding an applicable rule.

Consider the active constraint $m : T_m$ and a satisfying substitution σ. Given a normal derivation of σm from σT_m, we must find a reduction rule applicable to T_m that is compatible with σ.

By definition of the active constraint (see section 4.1), m is not a variable and T_m does not contain any standalone variables after application of the *(elim)* rule. Suppose $\sigma \vdash m : T_m$, *i.e.*, $\sigma m \in \mathcal{F}(\sigma T_m)$. Then, by proposition A.1, either $\sigma m \in \sigma T_m$, or there exists a normal derivation $\phi_1, \ldots, \phi_n$ s.t. $\sigma m \in \phi_n(\ldots \phi_1(\sigma T_m))$ and $\phi_n \in \Phi_{\text{synth}}$, or $\phi_1 \in \Phi_{\text{analz}}$, or $[x]^{\leftrightarrow}_y \in \sigma T_m$ for some x, y and $\phi_i = \phi_{\text{sdec}}$ applied to $[x]^{\leftrightarrow}_y$.

If $\sigma m \in \sigma T_m$, then the unification rule *(un)* can be applied to the $m : T_m$ constraint, and since the rule applies the most general unifier, it will be consistent with σ. Otherwise, an applicable reduction rule can be found by pattern matching given the normal derivation which satisfies one of the three conditions given above.

First, consider the case of $\phi_1 \in \Phi_{\text{analz}}$. The reduction rule corresponding to ϕ_1 will be applicable to σT_m. It must be the case that σT_m contains a "target" term t such that ϕ_1 operates on σt. There are no standalone variables in T_m, so t has the necessary top-level structure. The case of $\phi_n \in \Phi_{\text{synth}}$ is handled similarly. Since σm is the result of applying ϕ_n, σm must have the corresponding structure. Since m is not a standalone variable, m must have the same top-level structure as σm, and the corresponding decomposition rule is applicable to $m : T_m$.

Finally, consider the case when $\phi_i = \phi_{\text{sdec}}$ and σT_m contains a $[x]^{\leftrightarrow}_y$ term. Since there are no standalone variables in T_m, T_m must also contain a $[x']^{\leftrightarrow}_{y'}$ term, and the *(sdec)* rule can be applied to $m : T_m$.

A.4.1.3 Preserving the solution.

The proof that the applicable rule preserves the solution of the constraint set proceeds on cases of ϕ_1 if $\phi_1 \in \Phi_{\text{analz}}$, and ϕ_n if $\phi_n \in \Phi_{\text{synth}}$. For brevity, we omit the details and explain the proof for the case of $\phi_i = \phi_{\text{sdec}}$.

By the applicability argument, $[x]^{\leftrightarrow}_y \in T_m$ for some terms x and y and rule *(sdec)* is applicable to C. It will reduce C to $C' = C_<, y : (T_m \setminus [x]^{\leftrightarrow}_y) \cup \lceil x \rceil_y, m : (T_m \setminus [x]^{\leftrightarrow}_y) \cup x \cup y, C_>$.

By proposition 5.2, $T_m \cong (T_m \setminus [x]^{\leftrightarrow}_y) \cup \lceil x \rceil_y$. Observe that $\sigma y \in T_{i-1}$ where $T_{i-1} = \phi_{i-1}(\ldots \sigma T_m)$ (otherwise, ϕ_{sdec} would not be applicable to T_{i-1}). Therefore, $\sigma \vdash y : (T_m \setminus [x]^{\leftrightarrow}_y) \cup \lceil x \rceil_y$. In this case, according to proposition 5.3, $T_m \cong (T_m \setminus [x]^{\leftrightarrow}_y) \cup x \cup y$. This implies that if $\sigma \vdash m : T_m$, then $\sigma \vdash m : (T_m \setminus [x]^{\leftrightarrow}_y) \cup x \cup y$. Therefore, $\sigma \vdash C_<, y : (T_m \setminus [x]^{\leftrightarrow}_y) \cup \lceil x \rceil_y, m : (T_m \setminus [x]^{\leftrightarrow}_y) \cup x \cup y, C_> = C'$.

A.4.2 Completeness with encryption hiding

The initial constraint set contains no $\lceil x \rceil_y$ terms. The first time an $\lceil x \rceil_y$ term can appear in the reduction sequence is as a result of *(sdec)* application, which generates a constraint of the form $k : T \cup \lceil t \rceil_k$. We will show that such a constraint can be solved without applying *(sdec)* to the hidden term.

PROPOSITION A.2. *Suppose $k : T_k$ is satisfiable and T_k contains terms of the form $[t]^{\leftrightarrow}_k$. Then $\forall t\ k : T_k$ is satisfiable without decrypting $[t]^{\leftrightarrow}_k$.*

Assume the statement of the proposition is not true. Then *every* construction of k using terms from T_k must involve an application of ϕ_{sdec} on $[t]^{\leftrightarrow}_k$ for some t. Below, we annotate each application of ϕ_{sdec} with the encrypted term on which it operates, so that if ϕ_{sdec} is applied to $[x]^{\leftrightarrow}_y$, we write $\phi_{\text{sdec}}\langle [x]^{\leftrightarrow}_y \rangle$.

If the assumption is true, then for any solution $\sigma \vdash k : T_k$ and *any* sequence $\phi_1, \ldots, \phi_r$ such that $\sigma k \in \phi_r(\ldots \phi_1(\sigma T_k))$, $\exists\, i \in 1..r$ such that $\phi_i = \phi_{\text{sdec}}\langle [\sigma t]^{\leftrightarrow}_{\sigma k} \rangle$ for some t. Consider the shortest such sequence.

Let $T_i = \phi_{i-1}(\ldots \phi_1(\sigma T_k))$. Since $\phi_{\text{sdec}}\langle [\sigma t]^{\leftrightarrow}_{\sigma k} \rangle$ can be applied to T_i, it must be the case that $[\sigma t]^{\leftrightarrow}_{\sigma k} \in T_i$ and $\sigma k \in T_i$. This implies that $\sigma k \in \phi_{i-1}(\ldots \phi_1(\sigma T_k))$ where $\forall j \in 1..i-1\ \phi_j \neq \phi_{\text{sdec}}\langle [\sigma t]^{\leftrightarrow}_{\sigma k} \rangle$. This contradicts the shortest-sequence assumption and completes the proof of the proposition.

Now consider constraint $k : T \cup \lceil t \rceil_k$ created by *(sdec)* application. Since $\mathcal{F}$ is closed under ϕ_{open} and ϕ_{hide}, $\sigma \vdash k : T \cup \lceil t \rceil_k$ *iff* $\sigma \vdash k : T \cup [t]^{\leftrightarrow}_k$. By proposition A.2, $\exists\, \phi_1, \ldots, \phi_k$ s.t. $\sigma k \in \phi_k(\ldots \phi_1(\sigma T \cup \sigma[t]^{\leftrightarrow}_k))$ and $\forall i\ \phi_i \neq \phi_{\text{sdec}}\langle \sigma[t]^{\leftrightarrow}_k \rangle$. Note that $\sigma T \cup \sigma[t]^{\leftrightarrow}_k$ contains no terms of $\lceil x \rceil_y$ form. By the same argument as was used in normal derivation construction, we can show that $\forall i\ \phi_i \neq \phi_{\text{open}}$ or ϕ_{hide}. This implies $\forall i$ that if ϕ_i operates on the $\sigma[t]^{\leftrightarrow}_k$ term, ϕ_i is also applicable to the $\sigma\lceil t \rceil_k$ term because the only operators that distinguish between $\sigma[t]^{\leftrightarrow}_k$ and $\sigma\lceil t \rceil_k$ are $\phi_{\text{sdec}}\langle [\sigma t]^{\leftrightarrow}_{\sigma k} \rangle$ and $\phi_{\text{hide}}\langle [\sigma t]^{\leftrightarrow}_{\sigma k} \rangle$, and the sequence does not contain any such operators.

Given that σk may not contain $\sigma[t]^{\leftrightarrow}_k$ as a subterm, we conclude that $\sigma k \in \phi_k(\ldots \phi_1(\sigma T \cup \sigma\lceil t \rceil_k))$ *iff* $\sigma k \in \phi_k(\ldots \phi_1(\sigma T \cup \sigma[t]^{\leftrightarrow}_k))$. Moreover, each ϕ_i has the corresponding reduction rule $r_i = r(\phi_i)$ since that are no ϕ_{open} or ϕ_{hide} in the sequence. Because $\forall i\ \phi_i \neq \phi_{\text{sdec}}\langle \sigma[t]^{\leftrightarrow}_k \rangle$, none of r_i are *(sdec)* applied to $C_<, m' : T' \cup [t']^{\leftrightarrow}_k, C_>$. By induction over the length of the derivation, every reduction rule r_i is applicable to its respective constraint, and $\forall i \in 1..k$, if $\sigma \vdash r_{i-1}(\ldots r_1(C))$, then $\sigma \vdash r_i(r_{i-1}(\ldots C))$ where $C = C_<, k : T \cup \lceil t \rceil_k, m : T \cup t \cup k, C_>$ is the constraint set after the first application of rule *(sdec)*. Solution σ is preserved along the reduction sequence $r_k(\ldots r_1(C))$ by the same inductive argument as in the case of term sets without encryption hiding. Since constraint $k : T \cup \lceil t \rceil_k$ is satisfiable, by the end of the reduction sequence all constraints derived from it will be disposed of (either eliminated by unification, or reduced to $v : T$ where v is variable), and no unifications involve substituting $\lceil t \rceil_k$ for a variable since it appears as a standalone term in $T \cup \lceil t \rceil_k$. Therefore, $C^* = r_k(\ldots r_1(C))$ contains no terms of the form $\lceil x \rceil_y$, and the inductive completeness argument is true for the reduction sequences rooted in C^*.

To summarize, whenever rule *(sdec)* introduces a constraint containing $\lceil x \rceil_y$ to the constraint set, for any solution σ there exists a sequence of reduction rules that preserves σ. None of the rules in the sequence require $[x]^{\leftrightarrow}_y$ terms. Therefore, the sequence is applicable to constraint $k : (T_m \setminus [x]^{\leftrightarrow}_y) \cup \lceil t \rceil_k$ and leads to a constraint set in which there are no $\lceil x \rceil_y$ terms.