The CPSA Specification: A Reduction System for Searching for Shapes in Cryptographic Protocols

John D. Ramsdell Joshua D. Guttman
Moses D. Liskov Paul D. Rowe
The MITRE Corporation
CPSA Version 2.5.2

July 27, 2015

Contents

Abstract

We describe a term reduction system that enumerates all essentially different executions possible for a cryptographic protocol. We call them the *shapes* of the protocol. Naturally occurring protocols have only finitely many, indeed very few shapes. Authentication and secrecy properties are easy to determine from them, as are attacks and anomalies. Our Cryptographic Protocols Shapes Analyzer (CPSA) program is a direct implementation of the reduction system described within, and the form of the reduction system is partially determined by the implementation.

Chapter 1

Introduction

The Cryptographic Protocol Shapes Analyzer (CPSA) attempts to enumerate all essentially different executions possible for a cryptographic protocol. We call them the *shapes* of the protocol. Naturally occurring protocols have only finitely many, indeed very few shapes. Authentication and secrecy properties are easy to determine from them, as are attacks and anomalies.

The shapes analysis is performed within a pure Dolev-Yao model. The CPSA program reads a sequence of problem descriptions, and prints the steps it used to solve each problem. For each input problem, CPSA is given some initial behavior, and it descovers what shapes are compatible with it. Normally, the initial behavior is from the point of view of one participant. The analysis reveals what the other participants must have done, given the participant's view.

This document specifies the CPSA program using a term reduction system. Chapter 2 describes message algebras as order-sorted quotient term algebras. Chapter 3 presents an implementation-oriented view of strand spaces. Chapter 4 details the model of the adversary. The formal definition of a partial run of a protocol is called a skeleton, and is introduced in Chapter 5.

The term reduction systems used to specify the algorithm is presented in Chapter 6, and the primitive reduction rules are in Chapter 7. The rules used to transform terms called preskeletons into skeletons are in Chapter 8.

The algorithm used to model adversarial behavior is in Chapter 9. The algorithms used to infer what else must have happened given a partial description of a run of a protocol as skeleton is in Chapters 10 and 11. Chapters 13 and 12 find most general descriptions of CPSA answers—the shapes. Finally, Chapter 14 assembles reduction rules into one system that specifies

the CPSA program.

Appendix A describes an extension to Strand Space theory that models passwords and related concepts. Appendix B details the sense in which a protocol role can be viewed as an abstraction of a program. Appendix C describes a formula in the language of order-sorted first-order logic for each problem and its shapes found by CPSA. The formula is called a shape analysis sentence. The formula is modeled by all skeletons that describe full runs of a protocol when CPSA finds all the shapes for the problem.

CPSA's search is based on a high-level algorithm that was claimed to be complete, i.e. every shape can in fact be found in a finite number of steps [5, 8]. Further theoretical work [10] showed classes of executions that are not found by the algorithm, however it also showed that every omitted execution requires an unnatural interpretation of a protocol's roles. Hence the algorithm is complete relative to natural role semantics. See [13, Appendix B] for more on omitted executions.

A CPSA release includes two other documents, the CPSA Design [12] and the CPSA Primer [13]. The design document describes details of the CPSA implementation that would clutter this one. It should be read by anyone interesting in reading and modifying the source code. The CPSA Primer provides an overview of CPSA, and is worth reading before this document is approached.

1.1 Notation

A finite sequence is a function from an initial segment of the natural numbers. The length of a sequence X is $|X|$, and sequence $X = \langle X(0), \ldots, X(n-1) \rangle$ for $n = |X|$. Alternatively, $\langle x_0, x_1, \ldots, x_{n-1} \rangle = x_0 :: x_1 :: \ldots :: x_{n-1} :: \langle\rangle$. If S is a set, then S^* is the set of finite sequences of S, and S^+ is the non-empty finite sequences of S. The concatenation of sequences X_0 and X_1 is $X_0 \frown X_1$. The prefix of sequence X of length n is $X \dagger n$.

Generally, when discussing terms, a lowercase Latin letter is used to denote a term, and an uppercase Latin letter is used to denote a set of terms or a sequence of terms.

Chapter 2

Order-Sorted Message Algebras

CPSA models a message by an equivalence class of terms over a signature. A sort system is used to classify messages. CPSA depends on the sort system to allow it to treat a variable that represents an asymmetric key differently from a variable that represents an arbitrary message. In particular, CPSA uses order-sorted quotient term algebras [6] for message algebras. This formalism enables the use of well-known algorithms for unification and matching in the presences of equations and sorts [2].

This paper makes no attempt to provide a general introduction to order-sorted quotient term algebras. We use a message algebra called the Basic Crypto Algebra (BCA), which is the main algebra used by CPSA.

There are six BCA sorts: **mesg**, the sort of all messages, **skey**, the sort of symmetric keys, **akey**, the sort of asymmetric keys, **name**, the sort of participant names, and **text** and **data** for ordinary values. Sort **mesg** is sometimes written as $\top$ and the other sorts are called *base sorts*. All base sorts are subsorts of **mesg**. The function symbols, or *operations*, used to form terms are given by the signature in Figure 2.1.

Each variable x used to form a term has a unique sort s, written $x\colon s$. Variable set X is an indexed set of sets of variables, $X_s = \{x \mid x\colon s\}$. For BCA, $X_{\mathbf{mesg}}$, $X_{\mathbf{skey}}$, $X_{\mathbf{akey}}$, $X_{\mathbf{name}}$, $X_{\mathbf{text}}$, and $X_{\mathbf{data}}$ partition the set of variables in X. By abuse of notation, at times, we write X for the set of variables in X.

The Basic Crypto Quotient Term Algebra $\mathfrak{A}$ generated by variable set X is displayed in Figure 2.2. The union of the messages in $\mathfrak{A}$ is set of terms generated by X, and $\mathfrak{A}$ partitions the set of terms into a set of equivalence classes induced by the equations. Terms t_0 and t_1 are equivalent, written

Sorts: **name**, **text**, **data**, **skey**, **akey** $<$ **mesg**

Base sorts: **name**, **text**, **data**, **skey**, **akey**
Carried positions: $\bullet$ denotes a carried position.

$\{\!\lvert \bullet \rvert\!\}_{(\cdot)}\colon$ **mesg** $\times$ **mesg** $\to$ **mesg**	Encryption
$\#(\cdot)\colon$ **mesg** $\to$ **mesg**	Hashing
$(\bullet,\bullet)\colon$ **mesg** $\times$ **mesg** $\to$ **mesg**	Pairing
"$\ldots$"$\colon$ **mesg**	Tag constants
$K_{(\cdot)}\colon$ **name** $\to$ **akey**	Public key of name
$(\cdot)^{-1}\colon$ **akey** $\to$ **akey**	Inverse of key
$\mathsf{ltk}(\cdot,\cdot)\colon$ **name** $\times$ **name** $\to$ **skey**	Long term key

Equation: $(x^{-1})^{-1} \approx x$ for $x\colon$ **akey**

Figure 2.1: Basic Crypto Signature and Equation

$$
\begin{aligned}
\mathfrak{A}_{\mathbf{skey}} &= \{\{x\} \mid x \in X_{\mathbf{skey}}\} \cup \{\{\mathsf{ltk}(a,b)\} \mid a \in X_{\mathbf{name}}, b \in X_{\mathbf{name}}\} \\
\mathfrak{A}_{\mathbf{akey}} &= \{\{x^{-2n} \mid n \in \mathbb{N}\} \mid x \in X_{\mathbf{akey}}\} \\
&\quad \cup \{\{x^{-2n-1} \mid n \in \mathbb{N}\} \mid x \in X_{\mathbf{akey}}\} \\
&\quad \cup \{\{K_x^{-2n} \mid n \in \mathbb{N}\} \mid x \in X_{\mathbf{name}}\} \\
&\quad \cup \{\{K_x^{-2n-1} \mid n \in \mathbb{N}\} \mid x \in X_{\mathbf{name}}\} \\
\mathfrak{A}_{\mathbf{name}} &= \{\{x\} \mid x \in X_{\mathbf{name}}\} \\
\mathfrak{A}_{\mathbf{text}} &= \{\{x\} \mid x \in X_{\mathbf{text}}\} \\
\mathfrak{A}_{\mathbf{data}} &= \{\{x\} \mid x \in X_{\mathbf{data}}\} \\
\mathfrak{B} &= \mathfrak{A}_{\mathbf{skey}} \cup \mathfrak{A}_{\mathbf{akey}} \cup \mathfrak{A}_{\mathbf{name}} \cup \mathfrak{A}_{\mathbf{text}} \cup \mathfrak{A}_{\mathbf{data}} \\
\mathfrak{A}^0 &= \mathfrak{B} \cup \{\{x\} \mid x \in X_{\mathbf{mesg}}\} \cup \{\{x\} \mid x \text{ is a tag constant }\} \\
\mathfrak{A}^{n+1} &= \mathfrak{A}^n \cup \{\{(t_0,t_1) \mid t_0 \in T_0, t_1 \in T_1\} \mid T_0 \in \mathfrak{A}^n, T_1 \in \mathfrak{A}^n\} \\
&\quad \cup \{\{\{\!\lvert t_0 \rvert\!\}_{t_1} \mid t_0 \in T_0, t_1 \in T_1\} \mid T_0 \in \mathfrak{A}^n, T_1 \in \mathfrak{A}^n\} \\
&\quad \cup \{\{\#t \mid t \in T\} \mid T \in \mathfrak{A}^n\} \\
\mathfrak{A} = \mathfrak{A}_{\mathbf{mesg}} &= \textstyle\bigcup_{n\in\mathbb{N}} \mathfrak{A}^n
\end{aligned}
$$

Figure 2.2: BCA Messages $\mathfrak{A}$ and Atoms $\mathfrak{B}$

$t_0 \equiv t_1$, iff $t_0 \in T \wedge t_1 \in T$ for some $T \in \mathfrak{A}$. The canonical representative of a message is the t in $\{t' \mid t' \equiv t\}$ with the fewest occurrences of the $(\cdot)^{-1}$ operation.

Keys, names, data, and texts in the algebra are called *atoms* and are members of $\mathfrak{B}$. We write $t\colon B$ iff $\{t' \mid t' \equiv t\} \in \mathfrak{B}$. Note that encryption is defined with an encryption key of sort **mesg**. When the encryption key is of sort **akey** this is meant to model asymmetric encryption: otherwise, this models symmetric encryption. Note that even complex messages such as encryptions can be used as encryption keys in the symmetric sense.

To find the decryption key associated with an encryption, one must exclude the case in which the key is a variable of sort **mesg**, as there is no way to determine if the encryption operation denotes symmetric or asymmetric encryption. Therefore, the decryption key associated with encryption key t is $inv(t)$.

$$inv(t) = \begin{cases} \mathsf{invk}(t) & \text{if } t\colon \textbf{akey}; \\ \text{undefined} & \text{if } t \text{ is a variable of sort } \textbf{mesg}; \\ t & \text{otherwise.} \end{cases}$$

An important property possessed by the algebra is that for all $T \in \mathfrak{A}$, if there are any encryptions in T then all members of T are encryptions. As a result, a message can be identified as representing an encryption and if it is, decomposed into its plaintext and its decryption key. This property is a consequence of the fact that equations relate atoms, not arbitrary messages. A similar property holds for pairs and hashes. A hash is treated as a kind of encryption in which the term that is hashed is the encryption key.

We write $\mathfrak{A}_X$ when it is important to identify the variable set X that generates the algebra. Given two variable sets X and Y, a *substitution* is an order-sorted map $\sigma\colon X \to \mathfrak{A}_Y$ such that $\sigma(x) \neq x$ for only finitely many elements of X. For a substitution σ, the *domain* is the set of variables $Dom(\sigma) = \{x \mid \sigma(x) \neq x\}$ and the *range* is the set $Ran(\sigma) = \{\sigma(x) \mid x \in Dom(\sigma)\}$. Substitution σ_0 is *more general than* σ_1, written $\sigma_0 \trianglelefteq \sigma_1$, if there exists a substitution σ_2 such that $\forall x\, \sigma_1(x) \equiv \sigma_2(\sigma_0(x))$. Given a substitution $\sigma\colon X \to \mathfrak{A}_Y$, the unique homomorphism $\sigma^*\colon \mathfrak{A}_X \to \mathfrak{A}_Y$ induced by σ is also denoted σ.

A *position* p is a finite sequence of natural numbers. The term in t that

occurs at p, written $t \mathbin{@} p$, is:

$$
\begin{array}{l}
t \mathbin{@} \langle\rangle = t; \\
(t_0, t_1) \mathbin{@} i :: p = t_i \mathbin{@} p \text{ for } i \in \{0, 1\}; \\
\{\!|t_0|\!\}_{t_1} \mathbin{@} i :: p = t_i \mathbin{@} p \text{ for } i \in \{0, 1\}; \\
t^{-1} \mathbin{@} 0 :: p = t \mathbin{@} p.
\end{array}
$$

A term t *occurs in* term t' if $t = t' \mathbin{@} p$ for some p. A message T *occurs in* message T' if the canonical representative of T occurs in the canonical representative of T'.

A carried term is one that can be extracted from a message reception assuming plaintext is extractable from encryptions. The positions at which term t is carried in t' is $\mathit{carpos}(t, t')$, where

$$
\mathit{carpos}(t, t') = \begin{cases}
\{\langle\rangle\} & \text{if } t' \equiv t, \text{ else} \\
\{0 :: p \mid p \in \mathit{carpos}(t, t_1)\} & \\
\qquad & \text{if } t' = \{\!|t_0|\!\}_{t_1}, \text{ else} \\
\{i :: p \mid i \in \{0, 1\}, p \in \mathit{carpos}(t, t_i)\} & \\
\qquad & \text{if } t' = (t_0, t_1) \text{ else} \\
\emptyset & \text{otherwise.}
\end{cases}
$$

Term t *carries* t' if $\mathit{carpos}(t', t)$ is not empty, and $t' \sqsubseteq t$ when t' is carried by t. Note that for all terms t_0, t_1, t'_0, t'_1, if $t_0 \equiv t_1$ and $t'_0 \equiv t'_1$, then $\mathit{carpos}(t_0, t'_0) = \mathit{carpos}(t_1, t'_1)$. We write $t' \sqsubseteq_p t$ when $p \in \mathit{carpos}(t', t)$ and $t \mathbin{@} p \equiv t'$.

In what follows, we will often conflate a term with the message of which it is a member, and use lowercase letters to denote both.

Chapter 3

Strand Spaces and Bundles

When using strand space theory, one normally hypothesizes the existence of a single global strand space. This is a very reasonable assumption for theoretical analysis, but from the point view of an implementer, it turns out that it is better to assume there are many local strand spaces and the design specification task is to describe the relations between these local spaces. Our reformulation of strand space notation provides an implementation oriented way of describing the concept of a local strand space, and a direct link between from algorithm specification to the data structures used in the implementation.

A run of a protocol is viewed as an exchange of messages by a finite set of local sessions of the protocol. Each local session is called a *strand*. The behavior of a strand, its *trace*, is a sequence of messaging events. An *event* is either a message transmission or a reception. Outbound message $t \in \mathfrak{A}_X$ is written as $+t$, and inbound message t is written as $-t$. The set of traces over $\mathfrak{A}_X$ is $\mathfrak{C}_X = (\pm\mathfrak{A}_X)^+$. A message *originates* in a trace if it is carried by some event and the first event in which it is carried is outbound. A message is *gained* by a trace if it is carried by some event and the first event in which it is carried is inbound. A message is *acquired* by a trace if it first occurs in a reception event and is also carried by that event.

Abstractly, a strand space is a multiset of traces, but since we wish to name each element, a *strand space* Θ_X over algebra $\mathfrak{A}_X$ is defined to be a sequence of traces in $\mathfrak{C}_X$. A strand s is a member of the domain of Θ_X, and its trace is $\Theta_X(s)$. In a strand space, the elements of the generator set X denote atomic message elements, such as keys, and not composite messages, such as encryptions and pairs. Therefore, the sort of every variable in X is a base sort.

Message events occur at nodes in a strand space. For each strand s, there is a node for every event in $\Theta(s)$. The *nodes* of strand space Θ are $\{(s,i) \mid s \in Dom(\Theta), 0 \leq i < |\Theta(s)|\}$, the event at a node is $evt_\Theta(s,i) = \Theta(s)(i)$, and the message at a node is $msg_\Theta(s,i) = m$ such that $evt_\Theta(s,i) = \pm m$. Just as a position names a subterm within another term, a strand names a trace within a strand space, and a node names an event in a strand space. The relation $\Rightarrow$ defined by $\{(s,i-1) \Rightarrow (s,i) \mid s \in Dom(\Theta), 1 \leq i < |\Theta(s)|\}$ is called the *strand succession relation*.

A *bundle* in strand space Θ is a finite directed acyclic graph $\Upsilon(\Theta, \rightarrow)$, where the vertices are the nodes of Θ, and an edge represents communication ($\rightarrow$) or strand succession ($\Rightarrow$). For communication, if $n_0 \rightarrow n_1$, then there is a message t such that $evt_\Theta(n_0) = +t$ and $evt_\Theta(n_1) = -t$. For each reception node n_1, there is a unique transmission node n_0 with $n_0 \rightarrow n_1$.

Each acyclic graph has a transitive irreflexive relation $\prec$ on its vertices. The relation specifies the causal ordering of nodes in a bundle. An transitive irreflexive binary relation is also called a strict partial order.

An atom *uniquely originates* in a bundle if it originates in the trace of exactly one strand. An atom is *non-originating* in a bundle if it originates on no strand, but each of its variables occurs in some strand's trace.

In a run of a protocol, the behavior of each strand is constrained by a role in a protocol. Adversarial strands are constrained by roles as are non-adversarial strands. A protorole over $\mathfrak{A}_Y$ is $\mathsf{r}_Y(C, N, U)$, where $C \in \mathfrak{C}_Y$, $N \subseteq \mathfrak{B}_Y$, and $U \subseteq \mathfrak{B}_Y$. The trace of the role is C, its non-origination assumptions are N, and its unique origination assumptions are U. A protorole is a *role* if (1) $t \in N$ implies t is not carried in C, and all variables in N occur in C, (2) $t \in U$ implies t originates in C, (3) if variable x occurs in C then x is an atom or it is acquired in C, and (4) the trace of a role may not match the pattern $\langle -t, +t, \ldots \rangle$. This is to ensure that listeners, which are introduced on Page 13, cannot be confused with protocol constrained strands. A *protocol* is a set of roles. Let $Vars(P)$ be the set of variables that occur in the traces of the roles in protocol P.

A bundle $\Upsilon(\Theta_X, \rightarrow)$ is a *run of protocol* P if there is a role mapping $rl\colon \Theta_X \rightarrow P$ that satisfies properties for each $s \in Dom(\Theta_X)$. Assuming $rl(s) = \mathsf{r}_Y(C, N, U)$ and X and Y share no variables, and let $h = |\Theta_X(s)|$, the properties are (1) $h \leq |C|$, (2) there is a homomorphism $\sigma\colon \mathfrak{A}_Y \rightarrow \mathfrak{A}_X$ such that $\sigma \circ C \dagger h = \Theta_X(s)$, (3) $Dom(\sigma)$ is the set of variables that occur in $C \dagger h$, (4) if the variables in $t \in N$ occur in $Dom(\sigma)$, then $\sigma(t)$ is non-originating in $\Upsilon(\Theta_X, \rightarrow)$, and (5) if $t \in U$ originates at index i in C, and $i < h$, then $\sigma(t)$

uniquely originates in $\Upsilon(\Theta_X, \rightarrow)$ at node (s, i). Origination assumptions in bundles specified by roles are called *inherited origination assumptions*.

Chapter 4

Adversary Model

A fixed set of penetrator roles encodes the adversary model associated with a message algebra. For the Basic Crypto Algebra, there are eight roles. Each role makes no origination assumptions, and the trace of each role is given in Figure 4.1. The first line of the figure specifies many traces, one for each base sort, and a trace for each tag.

A strand exhibits non-adversarial behavior when its role is not a penetrator role. A non-adversarial strand is called a *regular* strand as is its role.

The penetrator cannot use a non-originating atom to encrypt or decrypt a message, because every key it uses must be carried in a message. Consider a uniquely originating atom that originates on a regular strand. The penetrator cannot make the atom using a create role, because the atom would originate in more than one trace. Therefore, the penetrator can use a uniquely originating atom to encrypt or decrypt a message only if it is transmitted by a regular strand unprotected by encryption.

Create(z: B)	$\langle +z \rangle$	$\langle +\text{“}\ldots\text{”} \rangle$
Pair(x, y: $\top$)	$\langle -x, -y, +(x, y) \rangle$	$\langle -(x, y), +x, +y \rangle$
Encrypt(x, y: $\top$)	$\langle -x, -y, +\{\!\lvert x \rvert\!\}_y \rangle$	$\langle -\{\!\lvert x \rvert\!\}_y, -\mathit{inv}(y), +x \rangle$
Hash(x: $\top$)	$\langle -x, +\#x \rangle$	

Figure 4.1: Basic Crypto Algebra Penetrator Role Traces

Chapter 5

Skeletons

The details of penetrator behavior are abstracted away when performing protocol analysis. The abstracted description of a bundle is called a realized skeleton, which is defined using a protoskeleton. A *protoskeleton* over $\mathfrak{A}_X$ is $\mathsf{k}_X(rl, P, \Theta_X, \prec, N, U)$, where $rl\colon\ Dom(\Theta_X) \to P$ is a role map, the sets X and $Vars(P)$ are disjoint, Θ_X is a sequence of traces in $\mathfrak{C}_X$, $\prec$ is a relation on the nodes in Θ_X, $N \subseteq \mathfrak{B}_X$ are its non-origination assumptions, and $U \subseteq \mathfrak{B}_X$ are its unique origination assumptions. Unlike a strand space, the sort of a variable in X need not be a base sort.

Assume the strands in bundle $\Upsilon(\Theta_X, \rightarrow)$ have been permuted so that regular strands precede penetrator strands in sequence Θ_X, and rl demonstrates the bundle is a run of protocol P. Let P' be P without penetrator roles. Skeleton $\mathsf{k}_X(rl', P', \Theta'_X, \prec, N, U)$ *realizes* the bundle if rl' and Θ'_X are the truncations of rl and Θ_X respectively that omit penetrator strands from their domains, $\prec$ is the transitive irreflexive relation associated with the bundle without penetrator nodes, N is the set of non-originating atoms with variables that occur in Θ'_X, and U is the set of atoms that uniquely originate and are carried by some regular event.

A protoskeleton $\mathsf{k}_X(rl, P, \Theta_X, \prec, N, U)$ is a *preskeleton* if the following properties hold.

1. Sequence rl demonstrates that the strands in $Dom(\Theta_X)$ satisfy the conditions for being a part of a run of protocol P.

2. Relation $\prec$ is transitive, irreflexive, and includes the strand succession relation ($\Rightarrow$).

3. If $n \prec n'$, then either $n \Rightarrow n'$, $evt_{\Theta_X}(n) = +t$ and $evt_{\Theta_X}(n') = -t'$, or $n \prec n'' \prec n'$ for some n''.

4. Each atom in N is carried by no event, and each variable in the atom occurs in some event.

5. Each atom in U is carried by some event.

6. N includes the non-originating atoms inherited from roles via the role map.

7. U includes the uniquely originating atoms inherited from roles via the role map.

Let $\mathcal{O}_k(t)$ be the set of nodes at which t originates in preskeleton k, and $\mathcal{G}_k(t)$ be the set of nodes at which t is gained in k. Preskeleton $\mathsf{k}_X(rl, P, \Theta_X, \prec, N, U)$ is a *skeleton* if each atom in U originates on at most one strand, and the node of origination precedes each node that gains the atom, i.e. for every $t \in U$, $n_0 \in \mathcal{O}_k(t)$ and $n_1 \in \mathcal{G}_k(t)$ implies $n_0 \prec n_1$.

Let $k_0 = \mathsf{k}_X(rl_0, P, \Theta_0, \prec_0, N_0, U_0)$ and $k_1 = \mathsf{k}_Y(rl_1, P, \Theta_1, \prec_1, N_1, U_1)$ be preskeletons. There is a *preskeleton homomorphism* from k_0 to k_1 if ϕ and σ are maps with the following properties:

1. ϕ maps strands of k_0 into those of k_1, and nodes as $\phi((s, i)) = (\phi(s), i)$, that is ϕ is in $Dom(\Theta_0) \to Dom(\Theta_1)$;

2. $\sigma \colon \mathfrak{A}_X \to \mathfrak{A}_Y$ is a message algebra homomorphism;

3. $n \in nodes(\Theta_0)$ implies $\sigma(evt_{\Theta_0}(n)) = evt_{\Theta_1}(\phi(n))$;

4. $n_0 \prec_0 n_1$ implies $\phi(n_0) \prec_1 \phi(n_1)$;

5. $\sigma(N_0) \subseteq N_1$;

6. $\sigma(U_0) \subseteq U_1$;

7. $t \in U_0$ implies $\phi(\mathcal{O}_{k_0}(t)) \subseteq \mathcal{O}_{k_1}(\sigma(t))$.

A homomorphism is *strandwise injective* if its strand map is injective. Two preskeletons are isomorphic if they are related by strandwise injective homomorphism in both directions. A homomorphism is *nodewise isomorphic* if the strand map ϕ implies a bijection on nodes, and $n_0 \prec_1 n_1$ implies

$\phi^{-1}(n_0) \prec_0 \phi^{-1}(n_1)$. A skeleton is *realized* if there is a nodewise isomorphic homomorphism from it to a skeleton that realizes a bundle, and message component of the homomorphism is injective.

Our formalism requires that every protocol include a listener role of the form: $lsn(x\colon \top) = \mathsf{r}(\langle -x, +x\rangle, \emptyset, \emptyset)$. Instances of this role are sometimes used to make penetrator derived messages visible in skeletons. We say skeleton k *realizes modulo listeners* bundle $\Upsilon(\Theta, \rightarrow)$ if k realizes $\Upsilon(\Theta', \rightarrow')$ and $\Upsilon(\Theta, \rightarrow)$ is the result of removing listener strands, and adjusting the communication ordering $\rightarrow$ appropriately.

The set of bundles denoted by preskeleton k, $[\![k]\!]$, is:

$$[\![k]\!] = \{\Upsilon \mid k \stackrel{\phi,\sigma}{\longmapsto} k' \text{ and } k' \text{ realizes modulo listeners } \Upsilon\}$$

A CPSA algorithm is *complete* if when given a preskeleton k, either the algorithm diverges, or else it terminates and produces a finite set of realized skeletons K, such that $[\![k]\!] = \bigcup_{k' \in K} [\![k']\!]$.

5.1 Blanchet's Simple Example Protocol

The following protocol is a simplified version of the Denning-Sacco key distribution protocol [4] due to Bruno Blanchet.

$$\begin{array}{l} A \rightarrow B : \{\!|\{\!|s|\!\}_{a^{-1}}|\!\}_b \\ B \rightarrow A : \{\!|d|\!\}_s \end{array}$$

Symmetric key s is freshly generated, asymmetric keys a^{-1} and b^{-1} are uncompromised, and the goal of the protocol is to keep data d secret. The protocol was constructed with a known flaw for expository purposes.

This CPSA description of the protocol has an initiator and a responder role.

$$\begin{array}{l} init(a, b\colon \mathsf{A}, s\colon \mathsf{S}, d\colon \mathsf{D}) = \mathsf{r}(\langle +\{\!|\{\!|s|\!\}_{a^{-1}}|\!\}_b, -\{\!|d|\!\}_s\rangle, \emptyset, \emptyset) \\ resp(a, b\colon \mathsf{A}, s\colon \mathsf{S}, d\colon \mathsf{D}) = \mathsf{r}(\langle -\{\!|\{\!|s|\!\}_{a^{-1}}|\!\}_b, +\{\!|d|\!\}_s\rangle, \emptyset, \emptyset) \end{array}$$

where we use A for sort **akey**, S for sort **skey**, and D for sort **data** to save space. The algebra for the initiator role is generated from X, where $X_{\mathsf{A}} = \{a, b\}$, $X_{\mathsf{S}} = \{s\}$, $X_{\mathsf{D}} = \{d\}$, $X_{\mathbf{text}} = \emptyset$, $X_{\mathbf{name}} = \emptyset$, and $X_{\top} = \emptyset$.

An interesting point of view for analysis is to see if the authentication goals of the initiator are met. To do so, we assume there was full length run

of an initiator strand, and let CPSA determine what else must have happened. Let variable set $Y = a, b\colon \mathsf{A}, s\colon \mathsf{S}, d\colon \mathsf{D}$. The point-of-view skeleton is:

$\mathsf{k}_Y(\langle \mathit{init}(a_0, b_0, s_0, d_0)\rangle,$	Role map
$\{\mathit{init}(a_0, b_0, s_0, d_0), \mathit{resp}(a_1, b_1, s_1, d_1)\},$	Protocol
$\langle\langle +\{\!|\{\!|s|\!\}_{a^{-1}}|\!\}_b, -\{\!|d|\!\}_s\rangle\rangle,$	Traces
$\emptyset,$	Node orderings
$\{a^{-1}, b^{-1}\},$	Non-origination
$\{s\})$	Unique origination

where the variable set that generates the algebra for the initiator and responder roles have been renamed so as to avoid conflicts with the variable set Y used by the skeleton.

The skeleton produced by CPSA for this problem follows. Notice that the two strands agree on the key b used in the outermost encryption of their first message, and indication that the authentication goals of the initiator are met. See Figure 5.1 to see the structure of the shape.

$\mathsf{k}_Y(\langle \mathit{init}(a_0, b_0, s_0, d_0), \mathit{resp}(a_1, b_1, s_1, d_1)\rangle,$	Role map
$\{\mathit{init}(a_0, b_0, s_0, d_0), \mathit{resp}(a_1, b_1, s_1, d_1)\},$	Protocol
$\langle\langle +\{\!|\{\!|s|\!\}_{a^{-1}}|\!\}_b, -\{\!|d|\!\}_s\rangle,$	Traces
$\langle -\{\!|\{\!|s|\!\}_{a^{-1}}|\!\}_b, +\{\!|d|\!\}_s\rangle\rangle,$	
$\{(0,0) \prec (1,0), (1,1) \prec (0,1)\},$	Node orderings
$\{a^{-1}, b^{-1}\},$	Non-origination
$\{s\})$	Unique origination

The homomorphism from the point-of-view skeleton to the shape is

$$(\langle 1\rangle, \{a \mapsto a, b \mapsto b, s \mapsto s, d \mapsto d\}).$$

An analysis of the authentication goals for the responder shows the flaw built into the protocol. To do the analysis, assume there was a full length run of a responder strand, and let CPSA determine what else must have happened. In this case, the point-of-view skeleton is:

$\mathsf{k}_Y(\langle \mathit{resp}(a_0, b_0, s_0, d_0)\rangle,$	Role map
$\{\mathit{init}(a_0, b_0, s_0, d_0), \mathit{resp}(a_1, b_1, s_1, d_1)\},$	Protocol
$\langle\langle -\{\!|\{\!|s|\!\}_{a^{-1}}|\!\}_b, +\{\!|d|\!\}_s\rangle\rangle,$	Traces
$\emptyset,$	Node orderings
$\{a^{-1}, b^{-1}\},$	Non-origination
$\{s\})$	Unique origination

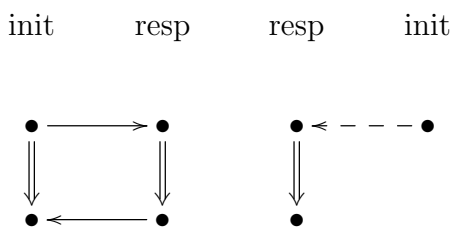

Figure 5.1: Shapes for Blanchet's Protocol

The shape generated by CPSA follows. An early indication of a problem is the variable set for the shape has three asymmetric keys. Let variable set $Z = a, b, b' \colon \mathsf{A}, s \colon \mathsf{S}, d \colon \mathsf{D}$. The shape is:

$\mathsf{k}_Z(\langle resp(a_0, b_0, s_0, d_0), init(a_1, b_1, s_1, d_1)\rangle,$	Role map
$\{init(a_0, b_0, s_0, d_0), resp(a_1, b_1, s_1, d_1)\},$	Protocol
$\langle\langle -\{\!\|\{\!\|s\|\!\}_{a^{-1}}\|\!\}_b, +\{\!\|d\|\!\}_s\rangle,$	Traces
$\langle +\{\!\|\{\!\|s\|\!\}_{a^{-1}}\|\!\}_{b'}\rangle\rangle,$	*Note key is b' not b!*
$\{(1,0) \prec (0,0)\},$	Node orderings
$\{a^{-1}, b^{-1}\},$	Non-origination
$\{s\})$	Unique origination

Notice that the two strands do not agree on the key used in the outermost encryption of their first message—an authentication failure. To see that the authentication failure leads to the failure to protect the secrecy of data d, the protocol is analyzed using the following point-of-view:

$\mathsf{k}_Y(\langle resp(a_0, b_0, s_0, d_0), lsn(x)\rangle,$	Role map
$\{init(a_0, b_0, s_0, d_0), resp(a_1, b_1, s_1, d_1), lsn(x)\},$	Protocol
$\langle\langle -\{\!\|\{\!\|s\|\!\}_{a^{-1}}\|\!\}_b, +\{\!\|d\|\!\}_s\rangle, \langle -d, +d\rangle\rangle,$	Traces
$\{(0,1) \prec (1,0),$	Node orderings
$\{a^{-1}, b^{-1}\},$	Non-origination
$\{s, d\})$	Unique origination

CPSA finds a shape that shows how data d is revealed to the adversary.

5.2 Dolev-Yao Example 1.3

The intended run of the protocol in the Dolev-Yao Example 1.3 is:

$$A \to B : \{|\{|m|\}_b, a|\}_b$$
$$B \to A : \{|\{|m|\}_a, b|\}_a$$

assuming text m is freshly generated, and asymmetric keys a^{-1} and b^{-1} are uncompromised.

The CPSA description of the protocol also has an initiator and a responder role.

$$\mathit{init}(a, b\colon \mathsf{A}, m\colon \mathsf{D}) = \mathsf{r}(\langle +\{|\{|m|\}_b, a|\}_b, -\{|\{|m|\}_a, b|\}_a\rangle, \emptyset, \emptyset)$$
$$\mathit{resp}(a, b\colon \mathsf{A}, m\colon \top) = \mathsf{r}(\langle -\{|\{|m|\}_b, a|\}_b, +\{|\{|m|\}_a, b|\}_a\rangle, \emptyset, \emptyset)$$

An interesting point of view for analysis is to see if m is kept secret after the initiator sends its message. Let variable set $Z = a, b\colon \mathsf{A}, m\colon \mathsf{D}$. The initial scenario preskeleton is:

$\mathsf{k}_Z(\langle \mathit{init}(a_0, b_0, m_0), \mathit{lsn}(x)\rangle,$	Role map
$\{\mathit{init}(a_0, b_0, m_0), \mathit{resp}(a_1, b_1, m_1), \mathit{lsn}(x)\},$	Protocol
$\langle\langle +\{|\{|m|\}_b, a|\}_b\rangle, \langle -m\rangle\rangle,$	Traces
$\emptyset,$	Node orderings
$\{a^{-1}, b^{-1}\},$	Non-origination
$\{m\})$	Unique origination

where the variable set that generates the algebra for the initiator role has been renamed so as to avoid conflicts with the variable set Z used by the preskeleton.

CPSA determines m is not kept secret by producing the shape in Figure 5.2. The added strands in the shape are instances of responder roles. The strands in the shape are:

$$\langle +\{|\{|m|\}_b, a|\}_b\rangle$$
$$\langle -m\rangle$$
$$\langle -\{|\{|m|\}_b, a'|\}_b, +\{|\{|m|\}_{a'}, b|\}_{a'}\rangle$$
$$\langle -\{|\{|\{|m|\}_b, a|\}_b, a''|\}_b, +\{|\{|\{|m|\}_b, a|\}_{a''}, b|\}_{a''}\rangle$$

The non-origination and unique origination assumptions are as they are in the initial scenario preskeleton. An interesting exercise left for the reader is to produce a bundle that is realized by the shape.

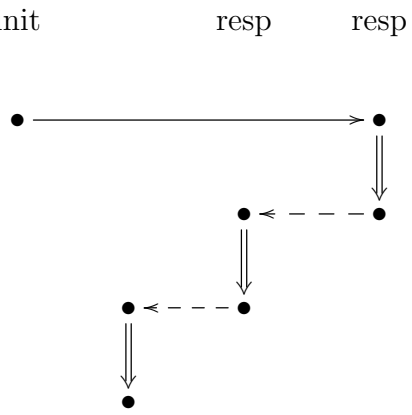

Figure 5.2: Dolev-Yao Example 1.3 Shape

5.3 Exercise

Consider the following roles.

$$\begin{aligned} \mathit{init}(a, b\colon \mathsf{A}) &= \mathsf{r}(\langle +(a,b), -(b,a)\rangle, \emptyset, \emptyset) \\ \mathit{resp}(a, b\colon \mathsf{A}) &= \mathsf{r}(\langle -(a,b), +(b,a)\rangle, \emptyset, \emptyset) \end{aligned}$$

Let $X = x, y\colon \mathsf{A}$ and $k = \mathsf{k}_X(\langle \mathit{init}(a,b), \mathit{resp}(a,b), \mathit{resp}(a,b)\rangle,$
$\{\mathit{init}(a,b), \mathit{resp}(a,b)\},$
$\langle\langle +(x,y), -(y,x)\rangle,$
$\langle -(x,y), +(y,x)\rangle,$
$\langle -(x,y), +(y,x)\rangle\rangle,$
Node ordering in Figure 5.3,
$\emptyset,$
$\emptyset)$

What is $[\![k]\!]$?
One member is shown in Figure 5.4.

5.4 External Syntax and Instances

The external syntax used by CPSA is a little different than what has been described here. In the external syntax, the trace and the role associated with a strand is specified by an *instance*. An instance is of the form $\mathsf{i}(r, h, \sigma)$,

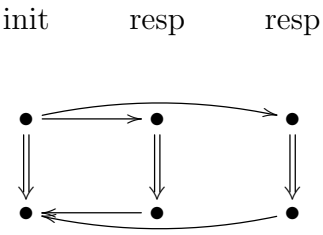

Figure 5.3: Exercise Skeleton

init	$\langle +(x,y), -(y,x) \rangle$
resp	$\langle -(x,y), +(y,x) \rangle$
resp	$\langle -(x,y), +(y,x) \rangle$
pair	$\langle -(y,x), -(y,x), +((y,x),(y,x)) \rangle$
sep	$\langle -((y,x),(y,x)), +(y,x) \rangle$

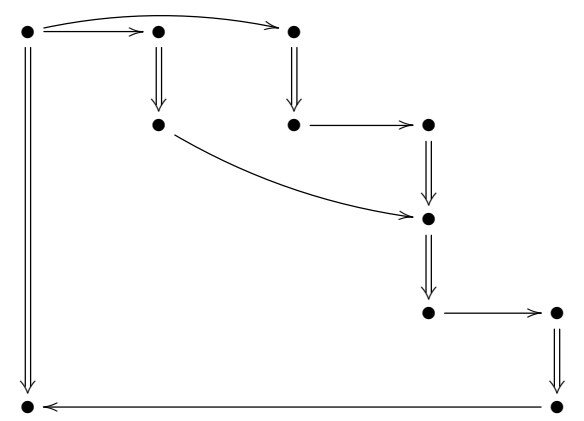

Figure 5.4: A Bundle Realized by the Example Skeleton

where r is a role, h specifies the length of a trace instantiated from the role, and σ specifies how to instantiate the variables in the role to obtain the trace. Thus when $r = \mathsf{r}_Y(C, N, U)$, the trace associated with $\mathsf{i}(r, h, \sigma)$ is $\sigma \circ C \dagger h$. An instance is well-formed if $1 \leq h \leq |C|$, and $Dom(\sigma)$ is the set of variables that occur in $C \dagger h$.

In the external syntax, the role map and sequence of traces are replaced by a sequence of instances. So for preskeleton $\mathsf{k}_X(rl, P, \Theta_X, \prec, N, U)$, the external syntax is $\mathsf{k}_X(P, I, \prec, N, U)$, where for each $s \in Dom(\Theta_x)$, $I(s) = \mathsf{i}(r, h, \sigma)$, $r = rl(s)$, and the trace of $\mathsf{i}(r, h, \sigma)$ is $\Theta_X(s)$.

Chapter 6

Algorithms as Reduction Systems

Algorithms in this paper are specified as abstract reduction systems [1, Chapter 2]. A reduction system is a pair $(A, \rightarrow)$, where reduction $\rightarrow$ is a binary relation $\rightarrow \subseteq A \times A$. Element $x \in A$ is a *normal form* if there is no y such that $x \rightarrow y$. The transitive closure of $\rightarrow$ is $\rightarrow^+$. The reflexive transitive closure of $\rightarrow$ is $\rightarrow^*$. A reduction is confluent if $x \rightarrow^* y_0$ and $x \rightarrow^* y_1$ implies there is a z such that $y_0 \rightarrow^* z$ and $y_1 \rightarrow^* z$. A reduction is terminating if there are no infinite descending chains. A reduction is convergent if it is confluent and terminating.

Let $\mathfrak{K}$ be the set of preskeletons. Algorithms are specified as reduction systems of the form $(\mathfrak{K}, \rightarrow)$, which are then used to specify a related setwise reduction system of the form $(2^{\mathfrak{K}}, \twoheadrightarrow)$. Setwise reduction systems are the ones with the interesting normal forms and confluence properties. In a setwise reduction system, reduction rewrites one element of a set to a set of elements.

Definition 6.1 (Setwise Reduction System)**.** The *setwise reduction system* of binary relation $\rightsquigarrow \subseteq \mathfrak{K} \times 2^{\mathfrak{K}}$ is a reduction system $(2^{\mathfrak{K}}, \twoheadrightarrow)$, where for each $K_0 \in 2^{\mathfrak{K}}$, $K_0 \twoheadrightarrow K_1$ if for some $k_0 \in K_0$, $k_0 \rightsquigarrow K_2$, $K_1 = K_2 \cup (K_0 \setminus \{k_0\})$, and $K_1 \neq K_0$.

The CPSA algorithm will be specified as a setwise term reduction system, where the initial problem is given a singleton in $2^{\mathfrak{K}}$, and the answers computed by an implementation of the algorithm are a normal form of the setwise reduction relation $\twoheadrightarrow_k$ defined in Chapter 14.

In what follows the relation $k \rightsquigarrow K$ is defined in terms of $\rightarrow \subseteq \mathfrak{K} \times \mathfrak{K}$ by specifying $\{k\} \twoheadrightarrow K$ using $\rightarrow$, so the $\rightsquigarrow$ relation is not explicitly defined.

We regard sets of preskeletons as factored by isomorphism, where each set has at most one representative of the equivalence class of isomorphic preskeletons. The definition of isomorphic preskeletons is given on Page 12.

The CPSA Design [12] describes an extension of a message algebra signature that models the data structures used in the CPSA program. The terms over the extended signature include ones that model preskeletons. Sets of terms of sort preskeleton are the domain of our setwise reduction systems.

Chapter 7

Primitive Preskeleton Operators

The are four primitive operators on preskeletons used by CPSA to solve authentication tests. Each operator is a partial map from preskeletons to preskeletons.

Definition 7.1 (Substitution Operator)**.** For order-sorted substitution $\sigma\colon X \to \mathfrak{A}_Y$, the operator $\mathbb{S}_\sigma$ is:

$$\begin{aligned}&\mathbb{S}_\sigma(\mathsf{k}_X(rl, P, \Theta_X, \prec, N, U)) = \\ &\quad \mathsf{k}_Y(rl, P, s \mapsto \sigma \circ \Theta_X(s), \prec, \sigma(N), \sigma(U))\end{aligned}$$

For $k' = \mathbb{S}_\sigma(k)$, there is a homomorphism from k to k' only if for all $t \in U_k$, $\mathcal{O}_k(t) \subseteq \mathcal{O}_{k'}(\sigma(t))$. The structure preserving maps associated with the homomorphism are ϕ_{id} and σ.

Definition 7.2 (Compression Operator)**.** For distinct strands s and s', operator $\mathbb{C}_{s,s'}$ compresses strand s into s'.

$$\begin{aligned}&\mathbb{C}_{s,s'}(\mathsf{k}_X(rl, P, \Theta_X, \prec, N, U)) = \\ &\quad \mathsf{k}_X(rl \circ \phi'_s, P, \Theta_X \circ \phi'_s, \prec', N, U)\end{aligned}$$

where

$$\phi'_s(j) = \begin{cases} j+1 & \text{if } j \geq s \\ j & \text{otherwise,} \end{cases}$$

relation $\prec'$ is the transitive closure of $\phi_{s,s'}(\prec)$, and

$$\phi_{s,s'}(j) = \begin{cases} \phi_s(s') & \text{if } j = s \\ \phi_s(j) & \text{otherwise} \end{cases}$$
$$\phi_s(j) = \begin{cases} j-1 & \text{if } j > s \\ j & \text{otherwise.} \end{cases}$$

The compression operator is only used when $\Theta_X(s)$ is a prefix of $\Theta_X(s')$, and when there is a homomorphism from k to $\mathbb{C}_{s,s'}(k)$. The structure preserving maps associated with the homomorphism are $\phi_{s,s'}$ and σ_{id}. Note that the compression operator is defined only when relation $\prec'$ is asymmetric, and that $\phi_{s,s'} \circ \phi'_s = \phi_{\text{id}}$.

Definition 7.3 (Ordering Enrichment Operator)**.** Operator $\mathbb{O}$ applied to k enriches $\prec_k$ by adding all elements implied by unique origination. That is $\mathbb{O}(k) = \mathsf{k}_X(rl, P, \Theta_X, \prec', N, U))$, where $k = \mathsf{k}_X(rl, P, \Theta_X, \prec, N, U)$ and $\prec' = (\prec \cup \{(n_0, n_1) \mid n_0 \in \mathcal{O}(k,t) \wedge n_1 \in \mathcal{G}(k,t)\})^*$.

The ordering enrichment operator is total and idempotent. The structure preserving maps associated with the operator's homomorphism are ϕ_{id} and σ_{id}, i.e. the homomorphism is an embedding.

Definition 7.4 (Augmentation Operator)**.** For node n, role r, and trace C, operator $\mathbb{A}_{n,r,C}$ is:

$$\mathbb{A}_{n,r,C}(\mathsf{k}_X(rl, P, \Theta_X, \prec, N, U)) = \\ \mathsf{k}_{X'}(rl \frown r, P, \Theta_X(s) \frown C, \prec', N', U')$$

where X' is X extended to include the variables in C, $\prec'$ is the minimal extension of $\prec$ such that $(|\Theta_X| + 1, |C|) \prec' n$, N' is N extended with non-origination assumptions inherited from r by C, and likewise for U'.

The structure preserving maps associated with the augmentation operator's homomorphism are ϕ_{id} and σ_{id}, i.e. the homomorphism is an embedding.

Chapter 8

Reductions on Preskeletons

This chapter describes the algorithm used to transform a preskeleton into a skeleton as a setwise term reduction system $(\mathfrak{K}, \twoheadrightarrow)$. Recall that the relation $k \rightsquigarrow K$ in Chapter 6 is defined in terms of $\rightarrow \subseteq \mathfrak{K} \times \mathfrak{K}$ by specifying $\{k\} \twoheadrightarrow K$ using $\rightarrow$. Additionally, when $k = \mathsf{k}(rl, P, \Theta, \prec, N, U)$, U_k is used to name U, and so forth for other components of k.

If a preskeleton k is not a skeleton, then it is either because some $t \in U_k$ actually originates at more than one node, or because for some $t \in U_k$, there is a node $n_1 \in \mathcal{G}_k(t)$, and a node $n_0 \in \mathcal{O}_k(t)$ such that $n_0 \not\prec_k n_1$. A preskeleton in which an atom assumed to be uniquely originating originates more than once is simply expunged. The second obstruction is resolved by enriching node orderings. A *hulled preskeleton* is a preskeleton in which every uniquely originating atom originates at most once, but it may lack some node orderings needed to be a skeleton.

Skeletons may contain "effectively equivalent" strands. After converting preskeletons to skeletons, a preskeleton reduction system may remove effectively equivalent strands using a process called thinning. A skeleton without any effectively equivalent strands is called a *thinned skeleton*.

Definition 8.1 (Hulling Reduction)**.** A non-hulled preskeleton k is expunged. The setwise hulling reduction rule is $\{k\} \xrightarrow{\mathbb{H}} \{\}$ when k is not hulled.

Definition 8.2 (Order Enrichment)**.** Suppose hulled preskeleton k_0 is not a skeleton. Hulled preskeleton k_0 reduces to skeleton k_1 by order enrichment, written $k_0 \xrightarrow{\mathbb{O}} k_1$, iff k_1 is the result of adding node orderings implied by origination. That is, $\prec_{k_1} = (\prec_{k_0} \cup \{(n_0, n_1) \mid n_0 \in \mathcal{O}(k_0, t) \wedge n_1 \in \mathcal{G}(k_0, t)\})^*$.

There is a homomorphism from k_0 to k_1 that is an embedding. For the setwise order enrichment reduction, $\{k_0\} \xrightarrow{\mathbb{O}}\!\!\!\!\rightarrow \{k_1 \mid k_0 \xrightarrow{\mathbb{O}} k_1\}$ when k_0 is a hulled preskeleton that is not a skeleton.

The thinning operator $\mathbb{T}_s$ removes an effectively equivalent strand.

Definition 8.3 (Thinning)**.** Two strands s and s' in a skeleton k are *effectively equivalent* if k with s removed and k with s' removed are isomorphic. (Page 12 defines isomorphic skeletons.) Skeleton k_0 reduces to skeleton k_1 by thinning, written $k_0 \xrightarrow{\mathbb{T}_s} k_1$, if and only if there exists a strand s' that is effectively equivalent to s in k_0 and k_1 is isomorphic to both k_0 with s removed and k_0 with s' removed. In detail, $k_0 \xrightarrow{\mathbb{T}_s} k_1$ if and only if there exists a strand s' such that

1. strands s and s' are not in the image of k_0's strand map from the point-of-view skeleton;
2. $|\Theta_{k_0}(s)| = |\Theta_{k_0}(s')|$;
3. there is a matcher σ such that $\sigma \circ \Theta_{k_0}(s) \equiv \Theta_{k_0}(s')$;
4. for all variables in k_0 with s removed, matcher σ is the identity;
5. matcher σ is a bijection, also called a renaming;
6. $k_0 \xrightarrow{\mathbb{S}_\sigma} k \xrightarrow{\mathbb{C}_{s,s'}} k_1$;
7. $k_0 \xrightarrow{\mathbb{S}_\sigma} k \xrightarrow{\mathbb{C}_{s',s}} k_2$;
8. k_1 is isomorphic to k_2.

For the setwise thinning reduction, $\{k_0\} \xrightarrow{\mathbb{T}_s}\!\!\!\!\rightarrow \{k_1 \mid k_0 \xrightarrow{\mathbb{T}_s} k_1\}$, when there is a k_1 such that $k_0 \xrightarrow{\mathbb{T}_s} k_1$.

Thinning in CPSA is also implemented for the case of sets of multiple strands rather than pairs of individual strands. The correctness of thinning has yet to be demonstrated.

8.1 Reduction Systems

Notice that a setwise hulling reduction may produce the empty set, but a setwise order enrichment and thinning reduction never does.

There are two preskeleton reduction systems, one with thinning, and one without. For the one without, let reduction $\twoheadrightarrow = \bigcup_{s,s'} \xrightarrow[]{\mathbb{H}}\!\!\!\!\!\twoheadrightarrow \cup \xrightarrow[]{\mathbb{O}}\!\!\!\!\!\twoheadrightarrow$.

Definition 8.4 (Preskeleton Reduction System)**.** Preskeleton k_0 reduces to skeleton k_1, written $k_0 \xrightarrow{skel} k_1$, if $\{k_0\} \twoheadrightarrow^* K$, $k_1 \in K$, and K is a normal form of $\twoheadrightarrow$.

For each skeleton k, $k \xrightarrow{skel} k$.

When using thinning, let reduction $\twoheadrightarrow = \bigcup_{s,s'} \xrightarrow[]{\mathbb{H}}\!\!\!\!\!\twoheadrightarrow \cup \xrightarrow[]{\mathbb{O}}\!\!\!\!\!\twoheadrightarrow \cup \bigcup_{s} \xrightarrow[]{\mathbb{T}_s}\!\!\!\!\!\twoheadrightarrow$.

Definition 8.5 (Preskeleton Reduction System with Thinning)**.** Preskeleton k_0 reduces to thinned skeleton k_1, written $k_0 \xrightarrow{tskel} k_1$, if $\{k_0\} \twoheadrightarrow^* K$, $k_1 \in K$, and K is a normal form of $\twoheadrightarrow$.

For each thinned skeleton k, $k \xrightarrow{tskel} k$.

Chapter 9

Penetrator Derivable

For each algebra, the powers of the adversary are defined by a set of roles. For the Basic Crypto Signature in Figure 2.1, the traces of the penetrator roles are in Figure 9.1. Penetrator roles make no origination assumptions.

The context in which penetrator strands appear determine the messages the adversary can derive. The context includes previously sent messages and atoms it is forbidden to originate. An atom that is assumed to be non-originating must be avoided as is a uniquely originating atom that is assumed to originate on a regular strand.

The ternary relation $T_p : T_a \vdash t$ states that message t is penetrator derivable from previously sent messages T_p while avoiding atoms T_a. The relation is defined by a set of inference rules. Most of the rules are justified by a penetrator role that when instantiated, derives a message in the conclusion of the rule.

The first rule states that no additional penetrator behavior is required to

$$
\begin{array}{l}
base(t) = \langle +t \rangle, \text{ where } t \text{ is an atom} \\
tag(t) = \langle +t \rangle, \text{ where } t \text{ is a tag} \\
cat(t_0, t_1) = \langle -t_0, -t_1, +(t_0, t_1) \rangle \\
sep(t_0, t_1) = \langle -(t_0, t_1), +t_0, +t_1 \rangle \\
enc(t_0, t_1) = \langle -t_0, -t_1, +\{\!|t_0|\!\}_{t_1} \rangle \\
dec(t_0, t_1) = \langle -\{\!|t_0|\!\}_{t_1}, -t_2, +t_0 \rangle, \text{ where } t_2 = inv(t_1) \\
hash(t) = \langle -t, +\#t \rangle
\end{array}
$$

Figure 9.1: Penetrator Traces

derive t if it has been previously sent.

$$\frac{t \in T_p}{T_p : T_a \vdash t}$$

A uniquely originating atom need not be avoided if it has been sent.

$$\frac{T_p : T_a \vdash t}{\{t_0\} \cup T_p : \{t_0\} \cup T_a \vdash t} \tag{9.1}$$

There are two decomposition steps available to the penetrator.

$$\frac{\{t_0, t_1\} \cup T_p : T_a \vdash t}{\{(t_0, t_1)\} \cup T_p : T_a \vdash t} \quad [\text{by } sep(t_0, t_1)] \tag{9.2}$$

$$\frac{T_p : T_a \vdash inv(t_1) \qquad \{t_0, \{\!|t_0|\!\}_{t_1}\} \cup T_p : T_a \vdash t}{\{\{\!|t_0|\!\}_{t_1}\} \cup T_p : T_a \vdash t} \quad [\text{by } dec(t_0, t_1)] \tag{9.3}$$

There are three constructive steps.

$$\frac{T_p : T_a \vdash t_0 \qquad T_p : T_a \vdash t_1}{T_p : T_a \vdash (t_0, t_1)} \quad [\text{by } cat(t_0, t_1)]$$

$$\frac{T_p : T_a \vdash t_0 \qquad T_p : T_a \vdash t_1}{T_p : T_a \vdash \{\!|t_0|\!\}_{t_1}} \quad [\text{by } enc(t_0, t_1)]$$

$$\frac{T_p : T_a \vdash t}{T_p : T_a \vdash \#t} \quad [\text{by } hash(t)]$$

There are three rules for indivisible messages.

$$T_p : T_a \vdash C_i \quad [\text{by } tag(C_i)]$$

$$\frac{t \notin T_a \qquad t \text{ an atom}}{T_p : T_a \vdash t} \quad [\text{by } base(t)]$$

A non-base sorted variable is derivable in a bundle that instantiates it with any message other than an element of $X_\top$.

$$\frac{t \in X_\top}{T_p : T_a \vdash t}$$

Definition 9.1 (Outbound predecessors)**.** The *outbound predecessors* of skeleton k at n is $outpred(k, n) = \{msg_k(n_0) \mid n_0 \prec_k n, n_0 \text{ is transmitting}\}$.

Definition 9.2 (Avoidance Set)**.** The *avoidance set* of skeleton k is $avoid(k) = N_k \cup \{t \mid t \in U_k \wedge |\mathcal{O}_k(t)| = 1\}$.

An atom in $avoid(k)$ is not available to the penetrator, except if it is exposed by a messages transmission. Clearly, only uniquely originating atoms can be exposed.

Definition 9.3 (Derivable Before)**.** A message t is *derivable before* reception node n in skeleton k, written $der(k, n, t)$, if $T_p : T_a \vdash t$ where $T_p = outpred(k, n)$ and $T_a = avoid(k)$.

Definition 9.4 (Realized Node)**.** A reception node n is *realized* in skeleton k if $msg_k(n)$ is derivable before n in k.

Notice that one can read off penetrator behavior from the proof tree used to demonstrate that $msg_k(n)$ is derivable before n in k. For example, if a decryption step is required by the proof, an instance of the penetrator's decryption role is indicated. In a bundle, for a non-base sorted variable, there is a substitution that maps the variable to a message that is not a non-base sorted variable. The substitution determines the penetrator behavior associated with the variable.

Conjecture 9.1 (Realized Skeleton)**.** A skeleton is realized if and only if all of its reception nodes are realized.

Partial Proof. Given a skeleton k in which all of its reception nodes are realized, the combination of the regular behavior in the skeleton, the penetrator behavior specified by the proof trees used to demonstrate each node is realized, and a substitution for non-base sorted variables determines a bundle. The skeleton of the bundle may have more non-originating atoms than is in N_k, however since the extra non-originating atoms are derivable by the bundle that realizes k, the proof trees for those atoms specify any additional penetrator behavior required.

The "only if" part of this proof has yet to be completed. □

9.1 Implementation

The derivable before a node predicate is implemented using auxiliary functions.

Definition 9.5 (Buildable)**.** Message t is *buildable* from previously sent messages T_p while avoiding T_a, written $bld(t, T_p, T_a)$, if $T_p : T_a \vdash t$ without the use of Inference Rules 9.1, 9.2, and 9.3.

Consider the following reduction system based on Inference Rules 9.1, 9.2, and 9.3.

$$\begin{array}{rll} \{t\} \cup T_p : T_a \to T_p : T_a \setminus \{t\} & & \text{if } t \text{ is an atom or in } X_\top \\ \{(t_0, t_1)\} \cup T_p : T_a \to \{t_0, t_1, (t_0, t_1)\} \cup T_p : T_a & & \text{if } t_0, t_1 \notin T_p \\ \{\{|t_0|\}_{t_1}\} \cup T_p : T_a \to \{t_0, \{|t_0|\}_{t_1}\} \cup T_p : T_a & & \text{if } t_0 \notin T_p \text{ and} \\ & & bld(inv(t_1), T_p, T_a) \end{array}$$

Definition 9.6 (Decompose)**.** Previously sent messages T_p and avoidance set T_a *decompose* to T'_p, T'_a, written $decompose(T_p, T_a) = (T'_p, T'_a)$, if $T_p : T_a \to^* T'_p : T'_a$ and (T'_p, T'_a) is a normal form of reduction $\to$.

The penetrator derivable predicate $T_p : T_a \vdash t$ is implemented as

$$\begin{array}{l} T_p : T_a \vdash t = \\ \quad \textbf{let } T'_p, T'_a = decompose(T_p, T_a) \textbf{ in} \\ \quad bld(t, T'_p, T'_a) \end{array}$$

The decomposition at a node function is

$$\begin{array}{l} dcmp(k, n) = \\ \quad decompose(outpred(k, n), avoid(k)) \end{array}$$

The derivable before a node predicate is implemented as

$$\begin{array}{l} der(k, n, t) = \\ \quad \textbf{let } T_p, T_a = dcmp(k, n) \textbf{ in} \\ \quad bld(t, T_p, T_a) \end{array}$$

Chapter 10

Carried Only Within

A set of encryptions T_e protects critical message t in message t' if t is carried by t' only within a member of T_e. The definition of the carried only within (COW) relation to follow makes this concept precise. The concept is used when solving authentication tests (Chapter 11).

Definition 10.1 (Ancestors)**.** Let $t' = t \,@\, p$. The *ancestors* of t' in t at p is the set $anc(t, p) = \{t \,@\, p' \mid p' \text{ a proper prefix of } p\}$.

Definition 10.2 (Carried Only Within)**.** Message t is *carried only within* set T_e in t' if for all carried positions p of t in t', there exists an ancestor $t_a \in anc(t', p)$ and $t_e \in T_e$ such that $t_a \equiv t_e$.

The function defines $carpos(t, t')$ is defined on Page 6, the set of positions at which t' carries t. The interface to each algebra exports $unify_0$, where

$$unify_0(t, t', \sigma) = \{\sigma' \circ \sigma \mid \sigma' \in unify(\sigma(t), \sigma(t'))\}.$$

The details of a reduction on skeletons called a augmentation will be described in Section 11.2. In simplified form, for an augmentation, given t, T_e, and t', one must find all most general unifiers σ such that $\sigma(t)$ is carried only within set $\sigma(T_e)$ in $\sigma(t')$.

A carried only within solution cannot be directly computed using Definition 10.2. Given terms t_a and t_e, the $unify_0$ function finds substitutions σ such $\sigma(t_a) \equiv \sigma(t_e)$, however, the carried positions $carpos(\sigma(t), \sigma(t'))$, are used before the $unify_0$ function computes the substitution σ. Figure 10.1 displays the iterative procedure that breaks the cyclic dependencies. Each step of the iteration improves an approximation of a solution to the problem.

$$
\begin{array}{l}
cows(t, T, t') = \\
\quad cows_0(t, T, t', \sigma_{\mathrm{id}}) \quad \text{— } \sigma_{\mathrm{id}} \text{ is the identity subst} \\
\\
cows_0(t, T, t', \sigma) = \\
\quad \textbf{if } \sigma(t) \text{ is COW } \sigma(T) \text{ at } \sigma(t') \textbf{ then} \\
\qquad \{\sigma\} \\
\quad \textbf{else} \\
\qquad \textbf{let } S = fold(t, T, t', \sigma) \textbf{ in} \\
\qquad \bigcup_{\sigma' \in S} cows_0(t, T, t', \sigma') \\
\\
fold(t, T, t', \sigma) = \\
\quad \{\sigma' \circ \sigma \mid \sigma' \in fold_0(\sigma(T), \sigma(t'), \{\sigma_{\mathrm{id}}\}, carpos(\sigma(t), \sigma(t')))\} \\
\\
fold_0(T, t', S, \{\}) = S \\
fold_0(T, t', S, \langle p \rangle \frown P) = \\
\quad fold_0(T, t', solve(anc(t', p), T, S), P) \\
\\
solve(T, T', S) = \\
\quad \{\sigma' \mid t \in T, t' \in T', \sigma \in S, \sigma' \in unify_0(t, t', \sigma)\}
\end{array}
$$

Figure 10.1: The *cows* Function

$$
\begin{aligned}
&\mathit{fold}(t, T, t', \sigma) = \\
&\quad \mathit{fold}_0(t, T, t', \sigma, \{\sigma_{\mathrm{id}}\}, \mathit{carpos}(\sigma(t), \sigma(t'))) \\
&\\
&\mathit{fold}_0(t, T, t', \sigma, S, \{\}) = \{\sigma' \circ \sigma | \sigma' \in S\} \\
&\mathit{fold}_0(t, T, t', \sigma, S, \langle p \rangle \frown P) = \\
&\quad \mathit{fold}_0(t, T, t', \sigma, \mathit{solve}(\mathit{anc}(\sigma(t'), p), \sigma(T), S), P)
\end{aligned}
$$

Figure 10.2: The Alternate *fold* Function

The correctness of this function is shown in [9], although for a version of the algorithm with the alternate definition for the function *fold* in Figure 10.2.

Chapter 11

Solving Authentication Tests

Definition 11.1 (Protectors)**.** Let *deriv* be a boolean valued function that determines if a message is derivable. The encryptions that protect t_c in t is $\mathit{protectors}(\mathit{deriv}, t_c, t) = \mathit{prot}(t)$ where

$$\mathit{prot}(t) = \begin{cases} \text{undefined} & \text{if } t \equiv t_c, \text{ else} \\ \{\} & \text{if } t = \{\!|t_0|\!\}_{t_1} \text{ and } t_c \text{ is not carried by } t_0, \text{ else} \\ \{\{\!|t_0|\!\}_{t_1}\} & \text{if } t = \{\!|t_0|\!\}_{t_1} \text{ and } \neg\, \mathit{deriv}(\mathit{inv}(t_1)), \text{ else} \\ \mathit{prot}(t_0) & \text{if } t = \{\!|t_0|\!\}_{t_1}, \text{ else} \\ \bigcup_{i<n} \mathit{prot}(t_i) & \text{if } t = f(t_0, \ldots, t_{n-1}) \text{ and } t \text{ is not an atom, else} \\ \{\} & \text{otherwise.} \end{cases}$$

Definition 11.2 (Escape Set)**.** The *escape set* for message t_c at n in skeleton k is the set of encryptions $\mathit{esc}(k, n, t_c)$ where

$$\mathit{esc}(k, n, t_c) = \{t_e \mid t_e \in \mathit{protectors}(\lambda t.\, \mathit{der}(k, n, t), t_c, t_o), t_o \in \mathit{outpred}(k, n)\}$$

and $\mathit{der}(k, n, t)$ is true when t is derivable before n in k (See Definition 9.3).

The *der* function is implemented as $\mathit{der}(k, n, t) = \mathit{bld}(t, T_p, T_a)$ where $(T_p, T_a) = \mathit{dcmp}(k, n)$, so that T_p and T_a need not be recomputed.

Definition 11.3 (Critical Position)**.** Position p is a *critical position* of $t = \mathit{msg}_k(n)$ if

1. p is a carried position in t,
2. either $t \mathbin{@} p \in U_k$, $t \mathbin{@} p = \#t_1$, or $t \mathbin{@} p = \{\!|t_0|\!\}_{t_1}$ and t_1 is not derivable before n in k,

3. $esc(k, n, t \mathbin{@} p)$ is defined, and

4. $anc(t, p) \cap esc(k, n, t \mathbin{@} p) = \emptyset$.

The message at a critical position is called an *authentication test*. It is a *nonce test* if the message is an atom, otherwise it is a *encryption test*. (A hash is treated as a kind of encryption in which the term that is hashed is the encryption key.) Observe that every critical message at a node in a skeleton is not derivable at the node.

Conjecture 11.1. A reception node is unrealized iff it has a critical position.

Definition 11.4 (Target Messages)**.** Let T_e be a set of messages, and t_c be a message. The set of *target messages* is

$$targ(t_c, T_e) = \{t_c\} \cup \{t_t \mid t_e \in T_e, p \in carpos(t_c, t_e), t_t \in anc(t_e, p)\} \setminus T_e.$$

Definition 11.5 (Critical Position Solved)**.** Suppose p is a critical position at n_0 in k_0 and $k_0 \stackrel{\phi,\sigma}{\longmapsto} k_1$. Let $t_0 = msg_k(n) \mathbin{@} p$, $t_1 = \sigma(t_0)$, $T_0 = esc(k_0, n_0, t_0)$, $T_1 = \sigma(T_0)$, $n_1 = \phi(n_0)$, and $t = msg_{k_1}(n_1)$. Critical position p is *solved* in k_1 after k_0 at n_0 if:

1. $anc(t, p) \cap T_1 \neq \emptyset$, or

2. for some $t_p \in outpred(k_1, n_1)$, t_1 is not carried only within T_1 in t_p, or

3. $targ(t_1, esc(k_1, n_1, t_1)) \setminus \sigma(targ(t_0, T_0)) \neq \emptyset$ and there are variables in k's protocol that are not atoms, or

4. the decryption key of a member of T_1 is derivable before n_1 in k_1, or

5. t_1 is an encryption and its encryption key is derivable before n_1 in k_1.

11.1 Test Solving Steps

A step used to solve a test is a contraction (Definition 11.6), a regular augmentation (Definition 11.7), a displacement (Definition 11.8), or a listener augmentation (Definition 11.9).

Definition 11.6 (Contraction)**.** Let p be a critical position at n in k, $t = msg_k(n)$, and $T_e = esc(k, n, t@p)$. Suppose there is a substitution σ such that for some $t_a \in anc(t, p)$, $t_e \in T_e$, $\sigma(t_a) = \sigma(t_e)$. Skeleton k_1 is a *contraction* if $k \xrightarrow{\mathbb{S}_\sigma} k_0 \xrightarrow{tskel} k_1$.

CPSA computes a set of substitutions for each critical position, and then removes some substitutions to form a complete set of most general unifiers. Only most general unifiers are used for contractions.

Definition 11.7 (Regular Augmentation)**.** Suppose substitution σ, non-listener role r, and trace C are selected as described in Section 11.2. Skeleton k_2 is a *regular augmentation* if $k \xrightarrow{\mathbb{S}_\sigma} k_0 \xrightarrow{\mathbb{A}_{n,r,C}} k_1 \xrightarrow{tskel} k_2$.

Definition 11.8 (Displacement)**.** Let substitution σ, non-listener role r, and trace C be selected as described in Section 11.2, and there be a preskeleton k_1 such that $k \xrightarrow{\mathbb{S}_\sigma} k_0 \xrightarrow{\mathbb{A}_{n,r,C}} k_1$. Suppose there are strands s and s', where one of them is the newly created strand, and a most general unifier σ' such that $\sigma'(\Theta_{k_1}(s)(j)) \equiv \sigma'(\Theta_{k_1}(s')(j))$ for $0 \leq j < |\Theta_{k_1}(s)|$. Skeleton k_4 is a *displacement* if $k_1 \xrightarrow{\mathbb{S}_{\sigma'}} k_2 \xrightarrow{\mathbb{C}_{s,s'}} k_3 \xrightarrow{tskel} k_4$.

Definition 11.9 (Listener Augmentation)**.** Let p be a critical position at n in k, $t_c = msg_k(n) @ p$, and $T_e = esc(k, n, t_c)$. For each $\{|t_0|\}_{t_1} \in T_e$, skeleton k_1 is a *listener augmentation* if $k \xrightarrow{\mathbb{A}_{n,lsn,C}} k_0 \xrightarrow{tskel} k_1$ and C listens for $inv(t_1)$, i.e. $C = \langle - inv(t_1), + inv(t_1) \rangle$. If $t_c = \{|t_0|\}_{t_1}$, then skeleton k_1 is a *listener augmentation* if $k \xrightarrow{\mathbb{A}_{n,lsn,C}} k_0 \xrightarrow{tskel} k_1$ and $C = \langle -t_1, +t_1 \rangle$.

For regular augmentation and displacement, CPSA removes solutions that lead to skeletons that are less general than other solutions, that is, when there is a homomorphism from a solution to the omitted solution.

Definition 11.10 (Cohort Member)**.** For unrealized node n in a skeleton k_0, and a position p at n, $k_0 \xrightarrow{n,p} k_1$ asserts that k_1 is a member of the cohort of k_0, where k_1 is derived using contraction, regular augmentation, displacement, or listener augmentation, and p is solved in k_1 after k_0 at n. For the setwise cohort member reduction, $\{k_0\} \xrightarrow{n,p} \{k_1 \mid k_0 \xrightarrow{n,p} k_1\}$, when n is unrealized in k_0, and p is a critical position at n.

Conjecture 11.2 (Critical Message Solved)**.** If $k_0 \xrightarrow{n_0,p_0} k_1 \xrightarrow{n_1,p_1} \ldots \xrightarrow{n_{\ell-1},p_{\ell-1}} k_\ell$ is a sequence of cohort member reductions, then for positive ℓ, p_0 is solved in k_ℓ after k_0 at n_0.

11.2 Augmentation

Let t_c be the critical message that demonstrates n is a test node in skeleton k. For each triple (σ, r, C) that satisfies some properties, there is a potential regular augmentation with $\xrightarrow{tskel} \circ \xrightarrow{\mathbb{A}_{n,r,C}} \circ \xrightarrow{\mathbb{S}_\sigma}$. When successful, the message t in the last event of the added strand is outbound, carries $\sigma(t_c)$, but $\sigma(t_c)$ is not carried only within escape set $\sigma(T_e)$ in t, where $T_e = esc(k, n, t_c)$, the escape set. Moreover, for every other message t in the strand, $\sigma(t_c)$ is carried only within escape set $\sigma(T_e)$ in t. The last event in the strand is called a *transforming event*, as this event no longer protects the critical message, but events that precede it do.

CPSA computes the parameters for a set of augmentation steps as follows. Suppose skeleton $k = \mathsf{k}_X(\_, P, \_, \_, \_, \_)$. First, compute the target messages, $T_t = targ(t_c, T_e)$. Next, for each non-listener role $\mathsf{r}(C_r, N_r, U_r) \in P$ and each index h where $C_r(h) = +t$, a transmission, do the following.

Create fresh variables: Let σ_r be a sort preserving variable renaming, where the domain is the variables that occur in $C_r \dagger h$, and every variable in the range does not occur in X or in P.

Insert critical message: For each message t' carried by t, and each $t_t \in targ(t_c, T_e)$, consider most general unifiers σ' where, $\sigma'(t') = \sigma'(t_t)$ and $\sigma_r \trianglelefteq \sigma'$. (In other words, $\sigma' = \sigma_0 \circ \sigma_r$ for some σ_0.)

Ensure previous events do not transform: For each σ', find most general unifiers σ such that for $0 \leq i < h$, $\sigma(t_c)$ is carried only within $\sigma(T_e)$ at $\sigma(C(i))$ and $\sigma' \trianglelefteq \sigma$. The function *cowt*, presented in Figure 11.1, performs the explorations, producing the substitutions $S' = cowt(t_c, T_e, C_r \dagger h, S)$. Function *fold* is defined in Figure 10.1. Let $S_{r,h}$ be the set S' with non-most general unifiers removed.

Ensure last event transforms: For each $\sigma \in S_{r,h}$, if $\sigma(t_c)$ is not carried only with $\sigma(T_e)$ at $\sigma(C(h))$, try augmenting with parameters n, r, $\sigma \circ C \dagger h$, and σ.

For target terms to be the reasonable set for insertion of the critical message, one must require that variables of sort message are acquired. This fact needs to be explained and noted as another reason for the acquired variable constraint.

$$
\begin{array}{l}
\mathit{cowt}(t, T, C, S) = \\
\quad \bigcup_{\sigma \in S} \mathit{cowt}_0(t, T, C, \sigma) \\
\\
\mathit{cowt}_0(t, T, C, \sigma) = \\
\quad \textbf{if } \forall t \pm t \in C \rightarrow \sigma(t) \text{ is COW } \sigma(T) \text{ at } \sigma(t') \textbf{ then} \\
\qquad \{\sigma\} \\
\quad \textbf{else} \\
\qquad \mathit{cowt}(t, T, C, \mathit{foldn}(t, T, C, \{\sigma\})) \\
\\
\mathit{foldn}(t, T, \langle\rangle, S) = S \\
\mathit{foldn}(t, T, \langle \pm t' \rangle \frown C, S) = \\
\quad \mathit{foldn}(t, T, C, \bigcup_{\sigma \in S} \mathit{fold}(t, T, t', \sigma))
\end{array}
$$

Figure 11.1: The *cowt* Function

Chapter 12

Collapsing and Preconditioning

The input preskeleton is preconditioned before it is subjected to authentication test solving. The preskeleton is converted to a skeleton and then collapsing is applied so as to ensure all shapes are found. Collapsing handles the case in which strands merged in the input lead to shapes.

Definition 12.1 (Collapsing)**.** Let k_0 and k_1 be two skeletons such that there are two strands, s and s', and a most general unifier σ such that $\sigma(\Theta_{k_0}(s)(j)) \equiv \sigma(\Theta_{k_0}(s')(j))$ for all $0 \leq j < |\Theta_{k_0}(s)|$. Then k_0 *collapses* to k_1, written $k_0 \xrightarrow{clp} k_1$, if $k_0 \xrightarrow{\mathbb{S}_\sigma} k \xrightarrow{\mathbb{C}_{s,s'}} k' \xrightarrow{skel} k_1$.

Definition 12.2 (Preconditioning)**.** For point-of-view preskeleton k_0, k_0 is preconditioned to k_1, written $k_0 \xrightarrow{pre} k_1$, if $k_0 \xrightarrow{skel} k \, (\xrightarrow{clp})^* \, k_1$.

Chapter 13

Generalization

The cohort reduction system produces a set of realized skeletons. Generalization attempts to convert that set into a set of skeletons. Not all possible cases are implemented due to performance issues, so it is not uncommon to find a realized skeleton in the output of a run of CPSA that is not a shape.

Conjecture 13.1 (Shape Completeness)**.** Without generalization, CPSA produces a complete set of shapes among the realized skeletons in its output.

Definition 13.1 (Generalize)**.** A skeleton k_0 *generalizes* skeleton k_1, written $k_1 \xrightarrow{<}_k k_0$, if both k_0 and k_1 are realized, k_0 and k_1 are not isomorphic, there is a homomorphism from a point-of-view skeleton k to k_0, and a strandwise injective homomorphism $k_0 \mapsto k_1$.

If skeletons are allowed to be isomorphic, we write $k_1 \xrightarrow{\leq}_k k_0$, and note that $\xrightarrow{\leq}_k$ defines a partial ordering. Therefore, there are maximal elements in the partial ordering. A shape associated with a preskeleton is a maximally generalized realized skeleton derived from the preskeleton.

Definition 13.2 (Shape)**.** Let k_0 be a preskeleton such that $k_0 \xrightarrow{pre} k$ for some skeleton k, and let k_1 be a realized skeleton such that $k \mapsto k_1$. Skeleton k_2 is a *shape* of k_0 if $k_1 \xrightarrow{\leq}_k k_2$, and k_2 is maximal among skeletons that generalize k_1.

There are four generalization reductions used to transform a realized skeleton into its shapes: deletion, weakening, forgetting, and separation.

Definition 13.3 (Deletion)**.** Skeleton k_0 *generalizes by deletion* skeleton k_1, written $k_1 \xrightarrow{\mathbb{D}_n}_k k_0$, if $k_1 \xrightarrow{<}_k k_0$, $k_2 \xrightarrow{skel} k_0$, and k_2 is the result of deleting node n in k_1 and all of the nodes that follow it in its strand.

Definition 13.4 (Weakening)**.** Skeleton k_0 *generalizes by weakening* skeleton k_1, written $k_1 \xrightarrow{\mathbb{W}_{n,n'}}_k k_0$, if $k_1 \xrightarrow{<}_k k_0$, $k_2 \xrightarrow{skel} k_0$, and k_2 is k_1 except $\prec_{k_2} = (\prec_{k_1} \setminus \{(n, n')\})^*$.

Definition 13.5 (Forgetting)**.** Skeleton k_0 *generalizes by origination assumption forgetting* skeleton k_1, written $k_1 \xrightarrow{\mathbb{F}_t}_k k_0$, if $k_1 \xrightarrow{<}_k k_0$, $k_2 \xrightarrow{skel} k_0$, and k_2 is k_1 except $U_{k_2} = U_{k_1} \setminus \{t\}$ and $N_{k_2} = N_{k_1} \setminus \{t\}$.

Sometimes a more general skeleton can be found by replacing some occurrences of one variable by a fresh variable. For variable separation, the location of an occurrence of a variable is defined using a skeleton's instance. Recall that in the external syntax, strand s in skeleton k is described by an instance of the form $\mathsf{i}(r, h, \sigma)$. (Instances are introduced in Section 5.4.)

Definition 13.6 (Location)**.** Variable x is at *location* (s, y, p) in k if the instance at $\Theta_k(s)$ is $\mathsf{i}(r, h, \sigma)$ and $x = \sigma(y) @ p$.

Definition 13.7 (Variable Separation)**.** Skeleton k_0 *generalizes by variable separation* skeleton k_1, written $k_1 \xrightarrow{\mathbb{V}_t}_k k_0$, if $k_1 \xrightarrow{<}_k k_0$, $k_2 \xrightarrow{skel} k_0$, and k_2 is k_1 except t is a variable that occurs in multiple locations in k_1, and k_2 is the result of replacing t with a variable t_0 of the same sort at a proper subset of t's locations, where t_0 occurs nowhere in k_1.

When separating a non-originating term, both the term and its clone are non-originating. When separating a uniquely originating term, either the term or its clone is uniquely originating.

What happens when separating t in k into t and t_0, and $\mathsf{ltk}(t, t) \in N_k$? Should a skeleton k_0 with $\mathsf{ltk}(t, t_0) \in N_{k_0}$ be a candidate separation? Currently, only skeletons k_1 with $\mathsf{ltk}(t, t) \in N_{k_1}$ and $\mathsf{ltk}(t_0, t_0) \in N_{k_1}$ are considered.

Definition 13.8 (Generalization)**.** The reduction $\xrightarrow{gen}_k = \bigcup_n \xrightarrow{\mathbb{D}_n}_k \cup \bigcup_{n,n'} \xrightarrow{\mathbb{W}_{n,n'}}_k \cup \bigcup_t \xrightarrow{\mathbb{F}_t}_k \cup \bigcup_t \xrightarrow{\mathbb{V}_t}_k$ is the *generalization* relation. For the setwise generalization reduction, $\{k_0\} \xrightarrow{gen}_k \{k_1\}$ when $k_0 \xrightarrow{gen}_k k_1$.

The fact that each generalization reduction replaces a singleton with just a singleton requires explanation. It's simply a matter of performance. If all possibilities are considered, CPSA run time would become dominated by generalization. Since generalization failures do not interfere with producing a complete description of the input, an approximation of the set of shapes is okay.

Conjecture 13.2 (Generalization)**.** The relation $\stackrel{gen}{\longrightarrow\!\!\!\!\!\rightarrow}_k$ is terminating.

Discussion

In [5], the shapes of a point-of-view skeleton are said to be minimal, in the partial ordering induced by injective homomorphism, among all realized homomorphic images of the point-of-view skeleton. Minimal corresponds to maximally generalized. The need for origination assumption forgetting was not known when [5] was written. Generalization by variable separation uses non-carried positions, and in particular, positions that traverse an atom edge. Algebras in previous strand space papers have no concept of a position that traverses an atom edge, and therefore cannot be used to specify generalization by variable separation.

Variable separation can be expensive when there are many possible ways to separate variables. The implementation simply truncates the search when it grows too large.

Chapter 14

Skeleton Reduction System

Let reduction $\twoheadrightarrow_k = (\xrightarrow{co}\!\!\!\!\!\twoheadrightarrow \cup \xrightarrow{gen}\!\!\!\!\!\!\twoheadrightarrow_k)^+$. This reduction system specifies the CPSA program.

Conjecture 14.1. The reduction $\twoheadrightarrow_k$ is confluent.

Conjecture 14.2 (Soundness)**.** Let k_0 be a preskeleton and k be an unrealized skeleton such that $k_0 \xrightarrow{pre} k$. Skeleton k_1 is a shape of k_0 if $\{k\} \twoheadrightarrow_k K$, $k_1 \in K$, and K is a normal form.

The set of bundles denoted by preskeleton k, $[\![k]\!]$ is defined on Page 13.

Conjecture 14.3 (Completeness)**.** Let k_0 be a preskeleton and k be an unrealized skeleton such that $k_0 \xrightarrow{pre} k$. For all K such that $\{k\} \twoheadrightarrow_k K$, $[\![k_0]\!] = \bigcup_{k_1 \in K} [\![k_1]\!]$.

Acknowledgement

Carolyn Talcott and Leonard Monk provided valuable feedback on drafts of this document.

Appendix A

Penetrator Non-Origination Assumptions

Penetrator non-origination assumptions have been added as an extension to the basic strand space theory. An atom is *penetrator non-originating* in a bundle if it originates on no penetrator strand, but each of its variables occurs in some strand's trace.

Penetrator non-origination assumptions can be used to model passwords. Several regular participants might know a password and originate it in a run of a protocol, but an idealized password is one the penetrator cannot guess.

A penetrator non-originating atom is similar to a non-originating atom, except in that it can be carried. There are two definitions that require change. Penetrator non-originating atoms must be added to the avoidance set of Definition 9.2. When the message at a critical position is an atom (see Definition 11.3, Item 2), instead of being uniquely originating, and can also be penetrator non-originating.

Appendix B

Programs Specified by a Role

Given the definitions in Chapter 3, a role can be viewed as an abstraction of a program, and a strand as an abstraction of a run of a program. But what program is specified by a role?

Consider a role that contains the event $-\{|t_0|\}_{t_1}$. If the program has the decryption key $inv(t_1)$ before the message is received, the program could decrypt the message and extract t_0. Alternatively, if the program has $\{|t_0|\}_{t_1}$, or has t_0 and the encryption key t_1, it might check to see if the received message is the same as the expected message, and abort the run if not.

Here is an example of when the second behavior is desired. In CPSA, before hashing was part of the algebra, an encryption was used to represent hashing. The hash of t, $\#t$, expanded to $\{|\text{“hash”}, t|\}_h$, where h was an asymmetric key known to all, but no one knew h^{-1}. The tag “hash” was added to the encryption so as to ensure a hash was never confused with other uses of encryption.

A role does not specify a valid program if the only possible way of interpreting the event $-\#t$ is by using h^{-1} to decrypt $\{|\text{“hash”}, t|\}_h$. The remainder of this section describes how this class of specification errors is detected for the Basic Crypto Algebra.

The behaviors associated with a trace depend on the set of messages available initially. The behaviors are specified by a data flow relation, $T_0, C \rhd T_1$. For trace C, the relation $T_0, C \rhd T_1$ asserts that when messages T_0 are available initially, there is a behavior of C that produces messages T_1.

A derivation tree used to demonstrate $T_0, C \rhd T_1$ shows the steps that enable the flow of data. The tree can be linearized, and thus specifies a sequential program that implements the role.

The data flow relation is defined with the aid of a data flow relation for a sequence of events, $T_0, C \rhd T_1$.

$$T, \langle\rangle \rhd T \qquad \frac{T_0, \langle \pm t \rangle \rhd T \quad T, C \rhd T_1}{T_0, \langle \pm t \rangle \frown C \rhd T_1}$$

The $T_0, \langle \pm t \rangle \rhd T_1$ relation is defined using the $T_0, C \rhd T_1$ relation. An outbound message can be formed if it is available initially

$$\frac{t \in T}{T, \langle +t \rangle \rhd T}$$

or if it can be formed by construction.

$$\frac{T, \langle +t_1, \ldots, +t_n \rangle \rhd T}{T, \langle +f(t_1, \ldots, t_n) \rangle \rhd T} \quad \left[\begin{array}{l} f(t_1, \ldots, t_n) \\ \text{not an atom} \end{array} \right]$$

An inbound message makes atoms and acquired variables available.

$$T, \langle -t \rangle \rhd T \cup \{t\} \quad [t \text{ an atom or a variable}]$$

When the decryption key is available, the contents of the encryption are also available. Furthermore, the encryption can be sent in future messages without access to its encryption key.

$$\frac{T_0, \langle + \mathit{inv}(t_1) \rangle \rhd T_0 \quad T_0, \langle -t_0 \rangle \rhd T_1}{T_0, \langle -\{\!|t_0|\!\}_{t_1} \rangle \rhd T_1 \cup \{\{\!|t_0|\!\}_{t_1}\}}$$

A received encryption that can be sent ensures the encryption agrees with currently available terms and makes nothing new available.

$$\frac{T, \langle +\{\!|t_0|\!\}_{t_1} \rangle \rhd T}{T, \langle -\{\!|t_0|\!\}_{t_1} \rangle \rhd T}$$

Consider an operation f other than the encryption operation. The order in which messages that occur in a message constructed using f are made available may determine if the decryption key of an encryption is available. All possible orderings must be explored. Let π_n be a permutation on the domain of a sequence of length n.

$$\frac{T_0, \langle -t_1, \ldots, -t_n \rangle \circ \pi_n \rhd T_1}{T_0, \langle -f(t_1, \ldots, t_n) \rangle \rhd T_1} \left[\begin{array}{l} f(t_1, \ldots, t_n) \\ \text{not an atom} \end{array} \right]$$

The data flow relation is used to find initial sets of atoms that are compatible with some behavior of a trace that produces messages.

```
proc(a, n, K_b, K_a^{-1})
  send({|a, n|}_{K_b});
  x_0 ← recv();
  x_1 ← decrypt(x_0, K_a^{-1});
  x_1 ≠ n → fail;
end
```

```
proc(a, n, K_b, K_a)
  send({|a, n|}_{K_b});
  x_0 ← recv();
  x_1 ← {|n|}_{K_a};
  x_0 ≠ x_1 → fail;
end
```

Figure B.1: Two programs that implement the role $\langle +\{|a, n|\}_{K_b}, -\{|n|\}_{K_a}\rangle$.

Definition B.1 (Trace Parameters)**.** The set of atoms T_0 are *parameters* of trace C if $T_0, C \rhd T_1$ for some T_1, and T_0 is minimal, that is for all T_0' such that $T_0', C \rhd T_1$, $T_0' \not\subset T_0$.

The role $\langle +\{|a, n|\}_{K_b}, -\{|n|\}_{K_a}\rangle$ has two sets of parameters, $\{a, n, K_b, K_a^{-1}\}$ and $\{a, n, K_b, K_a\}$. See Figure B for two examples of programs that implement the role, using distinct parameter sets.

The CPSA distribution contains a program that computes the set of parameters of a role. It was used to find an error in a role's use of hashing as described at the beginning of this section. The role in question is the verifier role that is part of an attestation protocol [3]. In an earlier version of the role, every set of parameters included the decryption key of the encryption used as a hash. The role was so complicated that inspection did not reveal the error.

Note that this section is specific to the basic crypto algebra in the sense that all non-atomic operations are assumed to be constructable, and encryption has the specific de-construction recipe shown. Although carried positions were not mentioned in this section, the inference system specifies the same concept. In fact, in future work we hope to show that the inference system is all one needs to define, and that which positions are carried, which are protected, and which sorts are atoms can be defined in terms of the inferences available.

Appendix C

Shape Analysis Sentences

For each point-of-view skeleton and its shapes found by CPSA, there is a formula in a language of order-sorted first-order logic called a *shape analysis sentence*. The sentence has a special form, $\forall X(\Psi \supset \bigvee_i \exists Y_i(\Delta_i \wedge \Phi_i))$, where Ψ, Δ_i, and Φ_i are conjunctions of atomic formulas and X and Y_i are variable sets. This fragment of first-order logic is called coherent logic. Formula Ψ describes the point-of-view skeleton k_0. For each homomorphism to a shape, $k_0 \stackrel{\delta_i}{\longmapsto} k_i$, formula Δ_i describes the structure preserving maps δ_i, and the shape k_i is described by Φ_i.

An interpretation of a shape analysis sentence is a skeleton. If CPSA finds all of the shapes and the homomorphisms associated with a point-of-view skeleton, the analysis' sentence is satisfied in all realized skeletons. Let Σ be a shape analysis sentence and Ψ be a security goal. If $\Sigma \supset \Psi$ is a theorem in order-sorted first-order logic, then Ψ is satisfied in all realized skeletons and its protocol achieves this goal.

Shape analysis sentences are closely related to security goals in [7], and were motivated by that work. This material was extracted from [11, Appendix B], the paper that introduced shape analysis sentences.

C.1 Security Goals

The signature for terms extends the one used for the underlying message algebra with one new sort **node**, the sort of nodes. Sort **node** has no subsorting relations with any other sort symbol.

Security goals make use of protocol specific and protocol independent

predicates. For each role $\mathsf{r}_Y(C, N, U) \in P$ and $i < |C|$, there is a protocol specific unary position predicate $P[r, i] : \mathbf{node}$. For each role $\mathsf{r}_Y(C, N, U) \in P$ and variable $x : S$ that occurs in C, there is a protocol specific binary parameter predicate $P[r, x] : \mathbf{node} \times S$. The protocol independent unary predicates are $\mathsf{non} : B$ and $\mathsf{uniq} : B$ for each base sort B. The remaining protocol independent predicates are binary, and are $\mathsf{uniq\text{-}at} : B \times \mathbf{node}$, $\mathsf{str\text{-}prec} : \mathbf{node} \times \mathbf{node}$, $\mathsf{prec} : \mathbf{node} \times \mathbf{node}$, and equality.

Soon we define $\mathcal{F}(k) = (Y, \Phi)$, where Φ is k's skeleton formula, and Y is the formula's variable set, but first we define the relevant nodes of a skeleton N. Let $k = \mathsf{k}_X(P, I, \prec, N, U)$ and let $\prec^-$ be the transitive reduction of $\prec$. Recall that Θ_X is the strand space defined by I, see Section 5.4. The *relevant nodes* of k are $N = N_s \cup N_\prec \cup N_u$ where

$$
\begin{aligned}
N_s &= \{(s, i) \mid s \in \mathit{Dom}(\Theta_X) \wedge i = |\Theta_X(s)| - 1\} \\
N_\prec &= \{(s, i) \mid (s', i') \in \mathit{nodes}(\Theta_X) \wedge s \neq s' \\
&\qquad \wedge ((s, i) \prec^- (s', i') \vee (s', i') \prec^- (s, i))\} \\
N_u &= \{(s, i) \mid t \in U, (s, i) \in \mathcal{O}_k(t)\}
\end{aligned}
$$

For $\mathcal{F}(k) = (Y, \Phi)$, the variable set Y is X augmented with a fresh variable of sort $\mathbf{node}$ for each node in N, and let $v(n)$ be the variable associated with node n.

The formula Φ is a conjunction of atomic formulas composed as follows.

- For each $(s, i) \in N$, assert $P[r, i](v(s, i))$, where $I(s) = \mathsf{i}(r, h, \sigma)$.
- For each $s \in \mathit{Dom}(I)$, let $I(s) = \mathsf{i}(r, h, \sigma)$. For each variable $x \in \mathit{Vars}(r^c \dagger h)$ and term $t = \sigma(x)$, assert $P[r, x](v(s, h - 1), t)$, where $r^c = C$ when $r = \mathsf{r}(C, N, U)$.
- For each $(s, i), (s, i') \in N$ such that $i < i'$, assert $\mathsf{str\text{-}prec}(v(s, i), v(s, i'))$.
- For each $(s, i) \prec^- (s', i')$ such that $s \neq s'$, assert $\mathsf{prec}(v(s, i), v(s', i'))$.
- For each $t \in N$, assert $\mathsf{non}(t)$.
- For each $t \in U$ and node n such that $n \in \mathcal{O}_k(t)$, assert $\mathsf{uniq\text{-}at}(t, v(n))$.
- For each $t \in U$ without a point of origination in k, assert $\mathsf{uniq}(t)$.

$$
\begin{array}{l}
\forall a_0, b_0 \colon \mathsf{A}, s_0 \colon \mathsf{S}, d_0 \colon \mathsf{D}, n_0 \colon \mathsf{N}(\\
\quad \mathit{resp}_1(n_0) \wedge \mathit{resp}_a(n_0, a_0) \wedge \mathit{resp}_b(n_0, b_0) \\
\qquad \wedge \mathit{resp}_s(n_0, s_0) \wedge \mathit{resp}_d(n_0, d_0) \\
\qquad \wedge \mathsf{non}(a_0^{-1}) \wedge \mathsf{non}(b_0^{-1}) \\
\quad \supset \\
\quad \exists a_1, b_1, b_2 \colon \mathsf{A}, s_1 \colon \mathsf{S}, d_1 \colon \mathsf{D}, n_1, n_2, n_3 \colon \mathsf{N}(\\
\qquad n_0 = n_1 \wedge a_0 = a_1 \wedge b_0 = b_1 \wedge s_0 = s_1 \wedge d_0 = d_1 \\
\qquad\quad \wedge \mathit{resp}_1(n_1) \wedge \mathit{resp}_a(n_1, a_1) \wedge \mathit{resp}_b(n_1, b_1) \\
\qquad\quad \wedge \mathit{resp}_s(n_1, s_1) \wedge \mathit{resp}_d(n_1, d_1) \wedge \mathit{resp}_0(n_2) \\
\qquad\quad \wedge \mathit{init}_0(n_3) \wedge \mathit{init}_a(n_3, a_1) \wedge \mathit{init}_b(n_3, b_2) \wedge \mathit{init}_s(n_3, s_1) \\
\qquad\quad \wedge \mathsf{uniq\text{-}at}(s_1, n_2) \wedge \mathsf{prec}(n_3, n_2) \wedge \mathsf{str\text{-}prec}(n_2, n_1) \\
\qquad\quad \wedge \mathsf{non}(a_1^{-1}) \wedge \mathsf{non}(b_1^{-1})))
\end{array}
$$

Figure C.1: Shape Analysis Sentence for Blanchet's Protocol

Given a set of homomorphisms $\delta_i \colon k_0 \mapsto k_i$, its shape analysis sentence is

$$\mathcal{S}(\delta_i \colon k_0 \mapsto k_i) = \forall X_0(\Phi_0 \supset \bigvee_i \exists X_i(\Delta_i \wedge \Phi_i)), \tag{C.1}$$

where $\mathcal{F}(k_0) = (X_0, \Phi_0)$. The same procedure produces X_i and Φ_i for shape k_i with one proviso—the variables in X_i that also occur in X_0 must be renamed to avoid trouble while encoding the structure preserving maps δ_i.

The structure preserving maps $\delta_i = (\phi_i, \sigma_i)$ are encoded in Δ_i by a conjunction of equalities. Map σ_i is coded as equalities between a message algebra variable in the domain of σ_i and the term it maps to. Map ϕ_i is coded as equalities between node variables in Φ_0 and node variables in Φ_i. Let v_0 be the node variables freshly generated for k_0, and v_i be the ones generated for k_i. The strand mapping part of Δ_i is

$$\bigwedge_{(s,j) \in Dom(v_0)} v_0(s, j) = v_i(\phi_i(s), j).$$

The shape analysis sentence for the second analysis of Blanchet's Simple Example Protocol in Section 5.1 is displayed in Figure C.1. The sort **node** is abbreviated as N, and the parameter predicate $P[r, x](z, t)$ is written $r_x(z, t)$ with the protocol left implicit.

Semantics of Skeleton Formulas. Let $k = \mathsf{k}_X(P, I, \prec, N, U)$. The universe of discourse is $\mathfrak{D} = (\mathbb{N} \times \mathbb{N}) \cup \mathfrak{A}_X$. When formula Φ is satisfied in

skeleton k with variable assignment $\alpha\colon Y \to \mathfrak{D}$, we write $k, \alpha \models \Phi$. We write $\bar{\alpha}$ when α is extended to terms in the obvious way. When sentence Γ is satisfied in skeleton k, we write $k \models \Gamma$.

- $k, \alpha \models P[r, i](y)$ iff $\alpha(y) \in \mathit{nodes}(\Theta_X)$, $\alpha(y) = (s, i)$, and for some σ,

$$\Theta_X(s) \dagger i + 1 = \sigma \circ r^c \dagger i + 1.$$

  Recall $r^c = C$ when $r = \mathsf{r}(C, N, U)$.

- $k, \alpha \models P[r, x](y, t)$ iff $\alpha(y) \in \mathit{nodes}(\Theta_X)$, $\bar{\alpha}(t) \in \mathfrak{A}_X$, and with $\alpha(y) = (s, i)$ and for some σ with $\sigma(x) = \bar{\alpha}(t)$,

$$\Theta_X(s) \dagger i + 1 = \sigma \circ r^c \dagger i + 1.$$

The interpretation of the protocol independent predicates is straightforward.

- $k, \alpha \models \mathsf{prec}(y, z)$ iff $\alpha(y) \prec \alpha(z)$.
- $k, \alpha \models \mathsf{str\text{-}prec}(y, z)$ iff $\alpha(y) \prec \alpha(z)$, $\alpha(y) = (s, i)$, and $\alpha(z) = (s, i')$.
- $k, \alpha \models \mathsf{non}(t)$ iff $\bar{\alpha}(t) \in \nu$.
- $k, \alpha \models \mathsf{uniq}(t)$ iff $\bar{\alpha}(t) \in \upsilon$.
- $k, \alpha \models \mathsf{uniq\text{-}at}(t, y)$ iff $\bar{\alpha}(t) \in \upsilon$ and $\alpha(y) \in \mathcal{O}_k(\bar{\alpha}(t))$.
- $k, \alpha \models y = z$ iff $\bar{\alpha}(y) = \bar{\alpha}(z)$.

Let Σ be a shape analysis sentence and Ψ be a security goal. If $\Sigma \supset \Psi$ is a theorem in order-sorted first-order logic, then Ψ is satisfied in all realized skeletons and its protocol achieves this goal.

Since $\prec$ is transitive, transitivity of *prec* can be used to prove a protocol achieves a goal. That is,

$$\mathsf{prec}(x, y) \wedge \mathsf{prec}(y, z) \supset \mathsf{prec}(x, z).$$

Furthermore, $\mathsf{str\text{-}prec}(x, y) \supset \mathsf{prec}(x, y)$.

Bibliography

[1] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[2] Franz Baader and Wayne Snyder. Unification theory. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 8. The MIT Press, 2001. `http://www.cs.bu.edu/~snyder/publications/UnifChapter.pdf`.

[3] George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian O'Hanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen. Principles of remote attestation. *International Journal of Information Security*, 10:63–81, 2011. 10.1007/s10207-011-0124-7.

[4] Dorothy E. Denning and Giovanni M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[5] Shaddin F. Doghmi, Joshua D. Guttman, and F. Javier Thayer. Searching for shapes in cryptographic protocols. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, number 4424 in LNCS, pages 523–538. Springer, March 2007. Extended version at `http://eprint.iacr.org/2006/435`.

[6] Joseph A. Goguen and Jose Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.

[7] Joshua D. Guttman. Security theorems via model theory. In *Express: Expressiveness in Concurrency, Workshop affiliated with Concur*, September 2009. Post-proceedings to appear in EPTCS, `http://www.eptcs.org/`.

[8] Joshua D. Guttman. Shapes: Surveying crypto protocol runs. In Veronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*. IOS Press, 2011.

[9] Moses Liskov and John D. Ramsdell. Implementing strand space algebras. `http://www.ccs.neu.edu/home/ramsdell/papers/algimpl.pdf`, 2011.

[10] Moses D. Liskov, Paul D. Rowe, and F. Javier Thayer. Completeness of CPSA. Technical Report MTR110479, The MITRE Corporation, 2011.

[11] John D. Ramsdell. Deducing security goals from shape analysis sentences. The MITRE Corporation, April 2012. `http://arxiv.org/abs/1204.0480`.

[12] John D. Ramsdell and Joshua D. Guttman. *CPSA Design*. The MITRE Corporation, 2009. In `http://hackage.haskell.org/package/cpsa` source distribution, `doc` directory.

[13] John D. Ramsdell and Joshua D. Guttman. *CPSA Primer*. The MITRE Corporation, 2009. In `http://hackage.haskell.org/package/cpsa` source distribution, `doc` directory.

Index