# CPSA and Formal Security Goals

John D. Ramsdell
The MITRE Corporation
CPSA Version 2.5.2

July 27, 2015

## Contents

# List of Figures



# List of Tables

# 1 Introduction

Analyzing a cryptographic protocol means finding out what security properties—essentially, authentication and secrecy properties—are true in all its possible executions.

Given a protocol definition and some assumptions about executions, CPSA attempts to produce descriptions of all possible executions of the protocol compatible with the assumptions. Naturally, there are infinitely many possible executions of a useful protocol, since different participants can run it with varying parameters, and the participants can run it repeatedly.

However, for many naturally occurring protocols, there are only finitely many of these runs that are essentially different. Indeed, there are frequently very few, often just one or two, even in cases where the protocol is flawed. We call these essentially different executions the *shapes* of the protocol. Authentication and secrecy properties are easy to "read off" from the shapes, as are attacks and anomalies, according to the introduction in the CPSA Primer [5].

But how easy is it to read off authentication and secrecy properties? What precisely is it that an expert sees? This paper describes CPSA's support for reasoning about security goals using first-order logic. With security goals expressed in first-order logic, intuition is replaced by determining if a formula is true in all executions of the protocol.

This treatment of security goals relies heavily on a branch of first-order logic called model theory. It deals with the relationship between descriptions in first-order languages and the structures that satisfy these descriptions. In our case, the structures are skeletons that denote a collection of executions of a protocol. This paper attempts to describe the language of security goals and its structures without requiring the reader to have studied model theory.

The model theoretical foundation of this approach to security goals appears in [1]. A practical use of security goals in protocol standardization is described in [2]. The precise semantics of the goal language is in [6, Appendix C]. The syntax of security goals appears in [5, Table 2].

The distribution in which this paper is included contains the sample input CPSA file `goals.scm`. It contains the examples in this paper. You are encouraged to run the examples and examine the output while reading this paper.

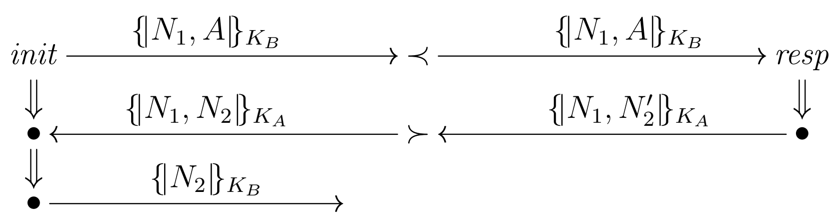
The CPSA Primer [5] is a prerequisite for reading this paper. In particular, the Needham-Schroeder Protocol in Section 10 is reanalyzed using security goals here. The roles are displayed in Figure 1.


```
(defprotocol ns basic
  (defrole init
    (vars (a b name) (n1 n2 text))
    (trace
     (send (enc n1 a (pubk b)))
     (recv (enc n1 n2 (pubk a)))
     (send (enc n2 (pubk b)))))
  (defrole resp
    (vars (b a name) (n2 n1 text))
    (trace
     (recv (enc n1 a (pubk b)))
     (send (enc n1 n2 (pubk a)))
     (recv (enc n2 (pubk b))))))
```

Figure 1: Needham-Schroeder Initiator and Responder Roles

```
(defgoal ns                ; Goal
  (forall ((b name) (n1 text) (z0 node))
    (implies
     (and (p "init" 2 z0)
      (p "init" "n1" z0 n1) (p "init" "b" z0 b)
      (non (privk b)) (uniq n1))
     (exists ((z1 node))
      (and (p "resp" 1 z1) (p "resp" "b" z1 b))))))

(defskeleton ns            ; Point of view skeleton
  (vars (a b name) (n1 n2 text))
  (defstrand init 3 (a a) (b b) (n1 n1) (n2 n2))
  (non-orig (privk b))
  (uniq-orig n1))
```




```
(defskeleton ns            ; Shape
  (vars (n1 n2 text) (a b name))
  (defstrand init 3 (n1 n1) (n2 n2) (a a) (b b))
  (defstrand resp 2 (n2 n2-0) (n1 n1) (b b) (a a))
  (precedes ((0 0) (1 0)) ((1 1) (0 1)))
  (non-orig (privk b))
  (uniq-orig n1)
  (satisfies yes))
```

Figure 2: Needham-Schroeder Initiator Point of View

The protocol is analyzed from the point of view of a complete run of one instance of the initiator role. The input security goal, followed by the point of view skeleton it generates and the shape produced by CPSA, are shown in Figure 2. The syntax and semantics of the goal will be explained later. Roughly speaking, it asserts that if a realized skeleton contains a full length initiator strand, its private key is uncompromised, and it uniquely generates `n1`, then the skeleton will contain a responder strand that agrees with the initiator on the name `b`. The shape produced by CPSA contains the annotation `(satisfies yes)`. This indicates that its structure satisfies the description given by the security goal. It will be explained later why the properties of CPSA allows us to conclude that this goal is true in all executions of the protocol, and therefore conclude that the Needham-Schroeder protocol achieves this authentication goal.

# 2 Syntax

To be precise, a security goal is an order-sorted first-order logic sentence in a restricted form. The sentence in Figure 2 has the form shared by all security goals. It is a universally quantified implication. The antecedent is a conjunction of atomic formulas. For this sentence, the conclusion is an existentially quantified conjunction of atomic formulas, but in general, the conclusion is a disjunction of existentially quantified conjunctions of atomic formulas. In what follows, `(false)` is a synonym for `(or)`.

| | | |
|---|---|---|
| GOAL | ← | (`defgoal` PROT SENT+ COMMENTS) |
| SENT | ← | (`forall` (DECL∗) (`implies` ANTEC CONCL)) |
| CONCL | ← | (`false`) \| EXISTL \| (`or` EXISTL∗) |
| EXISTL | ← | (`exists` (DECL∗) ANTEC) \| ANTEC |
| ANTEC | ← | ATOMIC \| (`and` ATOMIC∗) |

Variables are declared as they are for roles and skeletons with one exception, there is a new sort symbol `node`. Notice that in the sentence, variables `z0` and `z1` have sort `node`. Every universally quantified variable must occur in the antecedent of the implication.

The predicates used to construct an atomic formula (ATOMIC) are listed in Table 1. There are two classes of predicates, protocol specific and protocol independent predicates, and two kinds of protocol specific predicates, role position and role parameter predicates. Protocol specific predicates are

| Symbol | Sort | Description |
|---|---|---|
| p ROLE POS | **node** | Role position |
| p ROLE PARAM | **node** × **mesg** | Role parameter |
| `str-prec` | **node** × **node** | Precedes on strand |
| `prec` | **node** × **node** | Precedes |
| `non` | **atom** | Non-origination |
| `pnon` | **atom** | Penetrator non-origination |
| `uniq` | **atom** | Unique origination |
| `uniq-at` | **atom** × **node** | Unique origination at node |
| = | **all** × **all** | Equality |

Table 1: Predicates

distinguished from protocol independent predicates by being composed of three tokens, the first of which is `p`.

The first line of the table gives the syntax of a role position predicate. It contains three tokens, `p`, a string that names a role, and an integer that specifies a position within the trace of the role. That is, for role $r$ with a trace of length $n$, there are $n$ role position predicates, `p` $r$ $i$, for $0 \leq i < n$. Thus `(p "init" 2 z0)` is an atomic formula using the role position predicate for position 2 in the `init` role of the protocol in Figure 1.

The second line gives the syntax of a role parameter predicate. It contains three tokens, `p`, a string that names a role, and a string that names a role variable. For role $r$, there is role parameter predicate for each variable declared by $r$. Thus `(p "init" 2 "n1" z0 n1)` is an atomic formula using the role parameter predicate for parameter n1 in the `init` role of the protocol.

The empty string names the listener role of a protocol. The role has the variable `x` of sort `mesg` as its only role variable. There are two positions in the listener role.

When a variable of sort `node` occurs in a formula, its position must be specified using a role position formula. When an algebra variable occurs in formula, its association with the parameter of some role must be specified using a role parameter formula.

# 3 Semantics

In a `defgoal` sentence, the antecedent specifies the point of view skeleton. We focus on the antecedent. In the example,

```
(defstrand init 3 (a a) (b b) (n1 n1) (n2 n2))
```

is extracted from

```
(and (p "init" 2 z0)
  (p "init" "n1" z0 n1) (p "init" "b" z0 b)).
```

Notice that CPSA adds a binding for `a` and `n2` just as it does had

```
(defstrand init 3 (b b) (n1 n1))
```

been given as input.

Our aim now is to specify how to decide if a security goal is true in all possible executions of a protocol. A skeleton defines a set of executions that contain the skeleton's structure. We say a skeleton *satisfies* a formula if the formula is true in all executions that contain the skeleton's structure. To decide if a skeleton satisfies a formula, we decide if it satisfies each of its atomic formulas, and combine the results using the rules of first-order logic.

Atomic formula `(p "init" 2 z0)` is called a role position formula. A skeleton $k$ satisfies the formula if `z0` maps to a node $n = (s, 2)$ in $k$ such that

1. the trace of strand $s$ in $k$ has a length greater than 2, and
2. the trace when truncated to length 3 is an instance of the init role.

Consider the shape in Figure 2. It satisfies `(p "init" 2 z0)` when `z0` maps to $(0, 2)$.

Atomic formula `(p "init" "n1" z0 n1)` is called a role parameter formula. A skeleton $k$ satisfies the formula if `z0` maps to a node $n = (s, i)$ in $k$ and `n1` maps to a message algebra term $t$ in $k$ such that

1. the trace of strand $s$ in $k$ has a length greater than $i$,
2. the trace truncated to length $i + 1$ is an instance of the init role, and
3. the truncated trace is compatible with mapping the init role's `"n1"` role variable to $t$.

The shape in Figure 2 satisfies `(p "init" "n1" z0 n1)` when `z0` maps to $(0, 2)$ and `n1` maps to the message algebra term `n1`.

Collectively, a skeleton satisfies the formula

```
(and (p "init" 2 z0)
     (p "init" "a" z0 a) (p "init" "b" z0 b)
     (p "init" "n1" z0 n1) (p "init" "n2" z0 n2))
```

if the skeleton contains the structure specified by

```
(defstrand init 3 (a a) (b b) (n1 n1) (n2 n2)).
```

The antecedent in Figure 2 contains two origination assertions. The formula `(non (privk b))` is extracted as `(privk b)`. A skeleton $k$ satisfies the formula if `b` maps to a message algebra term $t$ in $k$ such that $k$ assumes that $t$ is non-originating. The unique origination formula for `n1` is similarly extracted.

Putting it all together, the mapping

$$\{\mathtt{n1} \mapsto \mathtt{n1}, \mathtt{n2} \mapsto \mathtt{n2}, \mathtt{a} \mapsto \mathtt{a}, \mathtt{b} \mapsto \mathtt{b}, \mathtt{z0} \mapsto (0, 2)\}$$

shows that the shape in Figure 2 satisfies the antecedent of the security goal.

The `prec` predicate is used to assert a node precedes another node. The conclusion in Figure 2 with `(prec z1 z0)` added is satisfied by the shape using the mapping $\mathtt{z0} \mapsto (0, 2)$ and $\mathtt{z1} \mapsto (1, 1)$.

The `str-prec` predicate is used to assert a node precedes another node and that both are on the same strand.

The `uniq-at` predicate is used to assert not only that an atom uniquely originates, but also the node at which it originates. It is typically used in conjunction with a `str-prec` formula. In the Figure 2 goal, the `(uniq n1)` formula could have been replaced by

```
(p "init" 0 z2) (str-prec z2 z0) (uniq-at n1 z2),
```

where `z2` has sort `node`. The extracted point of view skeleton is the same. Of course, an error is raised if the role position formula is replaced by `(p "init" 1 z2)`.

Recall that our aim in analyzing a protocol is to find out what security goals are true in all of its possible executions. We are interested in runs of CPSA that show that when every shape satisfies a goal, it is true in every execution.

When CPSA performs a shape analysis, every shape it generates refines the input skeleton. Skeleton refinement is defined in [5, Section 6], but roughly speaking, skeleton $A$ refines skeleton $B$ if $A$ contains the structure of skeleton $B$.

The skeleton $k_0$ extracted from the antecedent of a security goal has the property that every skeleton than refines $k_0$ satisfies the antecedent. A skeleton with this property is called the *characteristic skeleton* of the antecedent.

Given a goal $\Phi$, consider a shape analysis starting from the characteristic skeleton $k_0$ of its antecedent. Assume CPSA finds shapes $k_1, \ldots, k_n$ and that CPSA determines that each $k_i$ satisfies $\Phi$. Consider the executions that contain the structure in $k_0$. CPSA tells us that each execution is in the executions that contain the structure of some $k_i$. Furthermore, because $k_0$ is a characteristic skeleton, each $k_i$ satisfies the antecedent of $\Phi$. Therefore, $\Phi$ is true in all executions of the protocol and maximally informative.

# 4 Examples

This section contains examples of both authentication and secrecy goals. The first example shows the feedback the user receives when a shape does not satisfy a security goal. The second example shows how to use a listener to state a secrecy goal.

## 4.1 Needham-Schroeder Responder

Figure 3 contains an analysis of Needham-Schroeder from the point of view of a complete run of the responder under the assumption that the responder's private key is uncompromised and the nonce it generates uniquely originates.

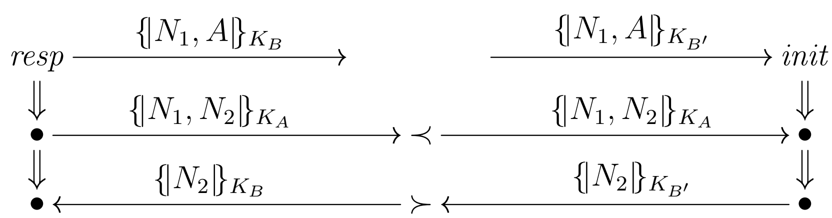
The conclusion of the goal asserts that in an execution compatible with the point of view, there must be an initiator strand that agrees with the responder strand on the name `b`. The shape produced by CPSA is a counterexample to this assertion. Because of this, CPSA annotates the shape with

```
(satisfies (no (a a) (b b) (n2 n2) (z0 (0 2)))).
```

The annotation includes a variable mapping for the shape that satisfies the antecedent of the goal but does not satisfy its conclusion. The reason the shape does not satisfy the goal is because the mapping `(b b)` maps the initiator's `b` parameter to `b`, not `b-0` as is required to model the shape.

```
(defgoal ns                 ; Goal
  (forall ((a b name) (n2 text) (z0 node))
    (implies
     (and (p "resp" 2 z0) (p "resp" "n2" z0 n2)
      (p "resp" "a" z0 a) (p "resp" "b" z0 b)
      (non (privk a)) (uniq n2))
     (exists ((z1 node))
      (and (p "init" 1 z1) (p "init" "b" z1 b))))))

(defskeleton ns             ; Point of view skeleton
  (vars (a b name) (n1 n2 text))
  (defstrand init 3 (a a) (b b) (n1 n1) (n2 n2))
  (non-orig (privk a))
  (uniq-orig n2))
```




```
(defskeleton ns             ; Shape
  (vars (n1 n2 text) (a b b-0 name))
  (defstrand resp 3 (n2 n2) (n1 n1) (b b) (a a))
  (defstrand init 3 (n1 n1) (n2 n2) (a a) (b b-0))
  (precedes ((0 1) (1 1)) ((1 2) (0 2)))
  (non-orig (privk a))
  (uniq-orig n2)
  (satisfies (no (a a) (b b) (n2 n2) (z0 (0 2)))))
```

Figure 3: Needham-Schroeder Responder Point of View

```
(defgoal ns
  (forall ((a b name) (n1 text) (z0 z1 node))
    (implies
     (and (p "init" 2 z0) (p "init" "n1" z0 n1)
      (p "init" "a" z0 a) (p "init" "b" z0 b)
      (p "" 0 z1) (p "" "x" z1 n1)    ; Listener
      (non (privk a)) (non (privk b)) (uniq n1))
     (false))))
```

Figure 4: Needham-Schroeder Secrecy Goal

Galvin Lowe identified this authentication failure in Needham-Schroeder and provided a fix. In the Needham-Schroeder-Lowe Protocol, the name `b` is included within the encryption in second message of both roles. With this modification, the shape found by CPSA satisfies the security goal in Figure 3, so Needham-Schroeder-Lowe achieves this authentication goal.

### 4.2 A Needham-Schroeder Secrecy Goal

Figure 4 contains an analysis of Needham-Schroeder from the point of view of a complete run of the initiator under the assumption that the responder's and its peer's private keys are uncompromised and the nonce `n1` it generates uniquely originates. Futhermore, the point of view asserts that the nonce is leaked using a listener.

```
(p "" 0 z1) (p "" "x" z1 n1)    ; Listener
```

CPSA finds no shapes, so Needham-Schroeder achieves this secrecy goal and does not leak `n1`.

### 4.3 Unilateral Authentication

This example focuses on unilateral authentication. To best visualize this example, process the sample file `goal.scm` with CPSA and display its output in a browser.

The protocol and its goal are shown in Figure 5. The goal asserts that if a realized skeleton contains a full length initiator strand that uniquely generates `n1`, and peer `a`'s private key is uncompromised, then the skeleton

```
(defprotocol unilateral basic
  (defrole init
    (vars (a name) (n text))
    (trace
     (send (enc n (pubk a)))
     (recv n)))
  (defrole resp
    (vars (a name) (n text))
    (trace
     (recv (enc n (pubk a)))
     (send n))))
```

```
(defgoal unilateral
  (forall ((a name) (n text)
           (z0 node))
   (implies
    (and (p "init" 1 z0)
     (p "init" "n" z0 n)
     (p "init" "a" z0 a)
     (non (privk a)) (uniq n))
    (exists ((z1 node))
     (and (p "resp" 1 z1)
      (p "resp" "a" z1 a))))))
```

Figure 5: Unilateral Protocol and Authentication Goal

```
(defgoal ns
  (forall ((a name) (n text) (z0 node))
   (implies
    (and (p "init" 1 z0) (p "init" "n1" z0 n)
     (p "init" "b" z0 a) (non (privk a)) (uniq n))
    (exists ((z1 node))
     (and (p "resp" 1 z1) (p "resp" "b" z1 a))))))
```

Figure 6: Translated Unilateral Goal for the Initiator

will contain a responder strand that agrees with the initiator on the name `a`. When given as input, CPSA verifies that the goal is true in all executions of the protocol.

In Section 4.1, a goal written for the Needham-Schroeder Protocol was reused to analyze the Needham-Schroeder-Lowe Protocol. We now use the unilateral authentication goal to analyze Needham-Schroeder.

Consider the goal in Figure 6. This goal is the result of translating the Unilateral role position and parameter predicates as follows.

```
p "init" "n"  →  p "init" "n1"
p "init" "a"  →  p "init" "b"
p "resp" "a"  →  p "resp" "b"
```

When given Figure 6 as input, CPSA verifies that the goal is true in all executions of the protocol.

```
(defgoal ns
  (forall ((a name) (n text) (z0 node))
   (implies
    (and (p "resp" 2 z0) (p "resp" "n2" z0 n)
     (p "resp" "a" z0 a) (non (privk a)) (uniq n))
    (exists ((z1 node))
     (and (p "init" 2 z1) (p "init" "a" z1 a))))))
```

Figure 7: Translated Unilateral Goal for the Responder

There is another way to translate the Unilateral goal.

```
p "init" 1    →  p "resp" "2"
p "init" "n"  →  p "resp" "n2"
p "init" "a"  →  p "resp" "a"
p "resp" "1"  →  p "init" "2"
p "resp" "a"  →  p "init" "a"
```

When given Figure 7 as input, CPSA verifies that the goal is true in all executions of the protocol.

CPSA has demonstrated two ways in which Needham-Schroeder achieves the goals of the Unilateral Protocol.

# 5 The Rest of the Story

The examples in the previous section demonstrate the typical way security goals are analyzed with CPSA. However, there are more features that may be useful.

A `defgoal` form may contain more than one sentence. See Figure 8 for an example. When presented with more than one sentence, CPSA extracts the point of view skeleton from the first sentence.

It is wise to ensure that the antecedent in every sentence is identical. When CPSA performs satisfaction-checking on sentence $\Phi$, it only determines if each shape it finds is satisfied. If the point of view skeleton is not the characteristic skeleton of the antecedent of $\Phi$, the fact that all skeletons satisfy $\Phi$ cannot be used to conclude that $\Phi$ is true in all executions of the protocol.

```
(defgoal ns
  (forall ((a b name) (n text) (z0 node))
    (implies
     (and (p "init" 1 z0) (p "init" "n1" z0 n)
      (p "init" "a" z0 a) (p "init" "b" z0 b)
      (non (privk a)) (non (privk b)) (uniq n))
     (exists ((z1 node))
      (and (p "resp" 1 z1) (p "resp" "b" z1 b)))))
  (forall ((a b name) (n text) (z0 node))
   (implies
     (and (p "init" 1 z0) (p "init" "n1" z0 n)
      (p "init" "a" z0 a) (p "init" "b" z0 b)
      (non (privk a)) (non (privk b)) (uniq n))
     (exists ((z1 node))
      (and (p "resp" 1 z1) (p "resp" "a" z1 a))))))
```

Figure 8: Two Initiator Authentication Goals

CPSA performs satisfaction-checking when an input skeleton in annotated with one or more security goals. The annotation uses the `goals` key.

```
(defskeleton
   ...
   (goals SENT+))
```

The program `cpsasas`, discussed in the next section, can be used to generate a formula with an antecedent such that the input skeleton is the characteristic skeleton of the antecedent.

## 5.1 Shape Analysis Sentences

A shape analysis sentence expresses all that can be learned from a run of CPSA as a security goal. If a security goal can be derived from a shape analysis sentence, then the protocol achieves the security goal, that is, the goal is true in all executions of the protocol.

The program `cpsasas` extracts shape analysis sentences from the output of CPSA. Consider the first example in this paper (Figure 2), which is in the

```
(defgoal ns
  (forall ((n1 n2 text) (b a name) (z z-0 node))
    (implies
      (and (p "init" 0 z) (p "init" 2 z-0)
        (p "init" "n1" z-0 n1) (p "init" "n2" z-0 n2)
        (p "init" "a" z-0 a) (p "init" "b" z-0 b)
        (str-prec z z-0) (non (privk b)) (uniq-at n1 z))
      (exists ((n2-0 text) (z-1 z-2 z-3 node))
        (and (p "init" 1 z-1) (p "resp" 0 z-2)
          (p "resp" 1 z-3) (p "resp" "n2" z-3 n2-0)
          (p "resp" "n1" z-3 n1) (p "resp" "b" z-3 b)
          (p "resp" "a" z-3 a) (prec z z-2) (prec z-3 z-1)
          (str-prec z z-1) (str-prec z-1 z-0)
          (str-prec z-2 z-3))))))
```

Figure 9: Initiator Shape Analysis Sentence

sample file `goals.scm`. To generate a maximally informative security goal from the initiator point of view with `ghci` and `Make.hs`, type

```
$ ghci Make.hs
*Make> sas "goals"
```

When using GNU make, type "`make goals_sas.text`". The resulting shape analysis sentence is displayed in Figure 9.

A shape analysis sentences asserts that either a realized skeleton does not satisfy its antecedent or it satisfies one or more of the disjuncts in its conclusion. CPSA has been designed so that this assertion is true. Therefore, every shape analysis sentence is true in all executions.

A security goal is true in all executions if it can be derived from a shape analysis sentence [4]. CPSA comes with a Prolog program that translates shape analysis sentences into Prover9 syntax [3]. Prover9 can then be used to perform the required theorem-proving.

In practice, theorem-proving using shape analysis sentences is rarely employed. It's clumsy and it requires too much expertise. The main use of `cpsasas` is for generating a formula that is edited to become a desired security goal.

# References


[1] Joshua D. Guttman. Establishing and preserving protocol security goals. *Journal of Computer Security*, 22(2):201–267, 2014.

[2] Joshua D. Guttman, Moses D. Liskov, and Paul D. Rowe. Security goals and evolving standards. In Liqun Chen and Chris Mitchell, editors, *Security Standardization Research*, volume 8839 of *LNCS*, pages 93–110. Springer, December 2014.

[3] Bill W. McCune. Prover9. `http://www.cs.unm.edu/~mccune/mace4/`.

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

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

[6] John D. Ramsdell, Joshua D. Guttman, Moses D. Liskov, and Paul D. Rowe. *The CPSA Specification: A Reduction System for Searching for Shapes in Cryptographic Protocols*. The MITRE Corporation, 2009. In `http://hackage.haskell.org/package/cpsa` source distribution, `doc` directory.