Symbolic Cryptographic Protocol Analysis using CPSA

John D. Ramsdell

The MITRE Corporation

October 2012

MITRE

Cyptographic Protocol Shapes Analyzer

Goal: Protocol analysis using Dolev-Yao model

Foundation: Strand Spaces

Distribution: `http://hackage.haskell.org/package/cpsa`

Contributors: Joshua D. Guttman, John D. Ramsdell, Jon C. Herzog, Shaddin F. Doghmi, F. Javier Thayer, Paul D. Rowe, and Moses D. Liskov

Pages of this form attempt to fill in the spoken part of a class using these slides.

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.

This presentation gives a very brief tour of the tool and how it is used in practice.

Anatomy of a CPSA Installation

- Programs
  - `cpsa`: main analysis tool
  - `cpsashapes`: extracts shapes from `cpsa` output
  - `cpsagraph`: visualize output using XHTML and SVG
  - `cpsadiff`: compare two `cpsa` output files
  - `cpsaannotations`: support rely-guarantee method
  - `cpsasas`: produce shape analysis sentences
  - `cpsapp`: pretty print input and output
  - Build tools: GNU makefile template and a Haskell script
- Documentation
  - User documentation: user guide, primer, and overview
  - Sample input: Needham-Schroeder, Blanchet, Woo-Lam, Otway-Rees, Yahalom, ffgg

Although there are many programs in the CPSA package, typical usage is very simple.

Using CPSA

- With a text editor, user enters a description of a
  - protocol as a set of roles
  - problem point-of-view
    (what is assumed to have happened)
- User runs the tool (`$ echo build | ghci Make.hs`
  or on Windows, click on `Make.hs` and type `build`)
- In a web browser, user views the output that shows
  - what else CPSA inferred must have happened, or
  - all CPSA steps used to produce the answers

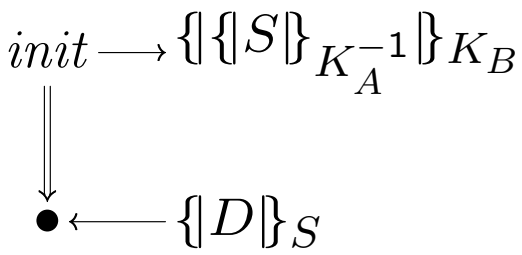
At this point, if you have not already done so, follow the instructions in `index.html` and analyze the examples. The rest of these slides will focus on Blanchet's "Simple Example Protocol", so open `blanchet.xhtml`. Click on Derivation Tree 9.

Blanchet's "Simple Example Protocol"

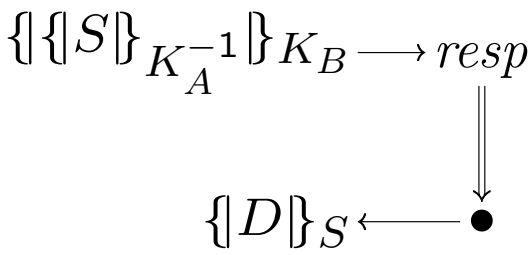
$$A \to B : \{|\{|S|\}_{K_A^{-1}}|\}_{K_B}$$
$$B \to A : \{|D|\}_S$$

CPSA Style Roles

Initiator (*init* role)



Responder (*resp* role)



MITRE

Blanchet's protocol has two steps. Alice sends to Bob a freshly generated symmetric key S signed with Alice's private key K_A^{-1} and then encrypted with Bob's public key K_B. Bob sends data D encrypted with the symmetric key.

Locate the description of the protocol in `blanchet.xhtml` in the third derivation tree.

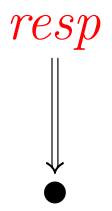
Blanchet Protocol in CPSA

```
(defprotocol blanchet basic
  (defrole init
    (vars (a b name) (s skey) (d data))
    (trace
     (send (enc (enc s (privk a)) (pubk b)))
     (recv (enc d s))))
  (defrole resp
    (vars (a b name) (s skey) (d data))
    (trace
     (recv (enc (enc s (privk a)) (pubk b)))
     (send (enc d s)))))
```

Locate the description of scenario in `blanchet.xhtml` in Derivation Tree 9.

Blanchet Problem Statement

- Analyze from the point-of-view of a full length responder
- Assume K_A^{-1} and K_B^{-1} are uncompromised
- Assume symmetric key S is freshly generated




```
(defskeleton blanchet
  (vars (a b name) (s skey) (d data))
  (defstrand resp 2 (a a) (b b) (s s) (d d))
  (non-orig (privk a) (privk b))
  (uniq-orig s))
```


MITRE

CPSA takes one step to find the shape associated with the scenario. Locate the shape for the scenario in this derivation tree.

Blanchet Shape

```
(defskeleton blanchet
  (vars (d data) (a b b-0 name) (s skey))
  (defstrand resp 2 (d d) (a a) (b b) (s s))
  (defstrand init 1 (a a) (b b-0) (s s))
  (precedes ((1 0) (0 0)))
  (non-orig (privk a) (privk b))
  (uniq-orig s)
  ...)
```

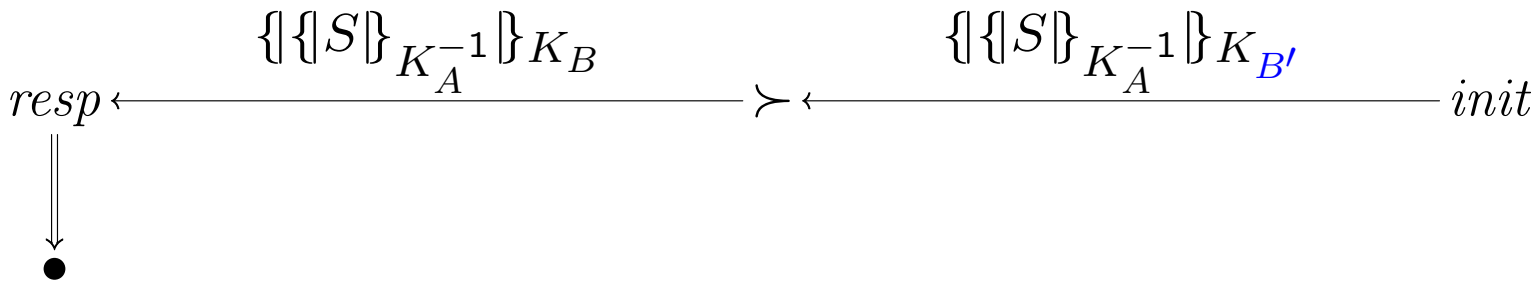
Observe that the responder role variable `b` maps to skeleton variable `b`, but the initiator role variable `b` maps to skeleton variable `b_0`. The two strands disagree on Bob's identity when agreement is expected. The dotted line in Skeleton 10 indicates the message transmitted differs from the one sent.

In the next slide, the traces show the implication of the disagreement on the instantiations of variable `b`.

Blanchet Shape with Traces

```
(defskeleton blanchet
  (vars (d data) (a b b-0 name) (s skey))
  (defstrand resp 2 (d d) (a a) (b b) (s s))
  (defstrand init 1 (a a) (b b-0) (s s))
  ...
  (traces
    ((recv (enc (enc s (privk a)) (pubk b)))
     (send (enc d s)))
    ((send (enc (enc s (privk a)) (pubk b-0)))))
  ...)
```

Blanchet Shape




```
(defskeleton blanchet
  (vars (d data) (a b b-0 name) (s skey))
  (defstrand resp 2 (d d) (a a) (b b) (s s))
  (defstrand init 1 (a a) (b b-0) (s s))
  (precedes ((1 0) (0 0)))
  ...)
```


MITRE

The operation field describes the authentication test that was solved to produce the shape. Authentication tests are introduced in `cpsaprimer.pdf`.

Blanchet Shape with Operation

```
(defskeleton blanchet
  (vars (d data) (a b b-0 name) (s skey))
  (defstrand resp 2 (d d) (a a) (b b) (s s))
  (defstrand init 1 (a a) (b b-0) (s s))
  ...
  (operation
    encryption-test          ; Authentication test type
    (added-strand init 1)    ; Regular augmentation
    (enc s (privk a))        ; Critical message
    (0 0))                   ; Test node
  ...)
```

The operation field explains why CPSA produced strands that do not agree on the identity of `b`. The critical message does not refer to it, so there is nothing that forces agreement.

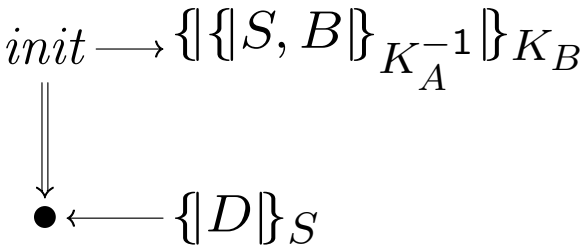
Navigate to Derivation Tree 23 in `blanchet.xhtml`.

Corrected Blanchet Example Protocol

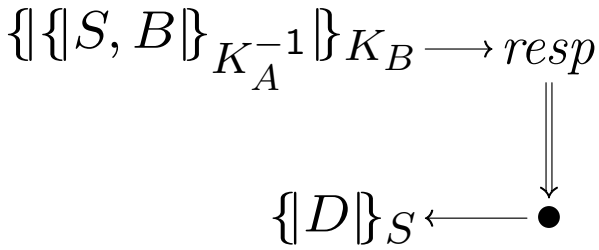
$$A \to B : \{|\{|S, B|\}_{K_A^{-1}}|\}_{K_B}$$
$$B \to A : \{|D|\}_S$$

CPSA Style Roles

Initiator (*init* role)



Responder (*resp* role)



MITRE

Corrected Blanchet Protocol in CPSA

```
(defprotocol blanchet-fixed basic
  (defrole init
    (vars (a b name) (s skey) (d data))
    (trace
     (send (enc (enc s b (privk a)) (pubk b)))
     (recv (enc d s))))
  (defrole resp
    (vars (a b name) (s skey) (d data))
    (trace
     (recv (enc (enc s b (privk a)) (pubk b)))
     (send (enc d s)))))
```

The corrected protocol ensures the critical message refers to `b`. As a result, CPSA infers the desired agreement.

Corrected Blanchet Shape

```
(defskeleton blanchet-fixed
  (vars (d data) (a b name) (s skey))
  (defstrand resp 2 (d d) (a a) (b b) (s s))
  (defstrand init 1 (a a) (b b) (s s))
  (precedes ((1 0) (0 0)))
  (non-orig (privk a) (privk b))
  (uniq-orig s)
  (operation encryption-test
    (added-strand init 1)
    (enc s b (privk a))   ; Critical message
    (0 0))
  ...)
```