An Overview of STA, a Tool for Automatic Analysis of Security Protocols

Marzia Buscemi, Università di Catania
buscemi@dmi.unict.it

joint work with Michele Boreale

Background

- Protocols modeled as concurrent systems, described by means of an appropriate language.
- Finite-state model checking and reacheability analysis rely on exploration of the space of this system. The sequences of actions (traces) are analyzed to check whether they lead to an insecure state.

State Explosion Problem

- At any stage, the intruder may nondeterministically choose a message and send it to a participant.
- There are *infinitely many* messages that the intruder can produce (by nonce creation, nested encryption and pairing, etc.).
- Thus, the protocol model is *infinitely branching* - in particular, gives rise to infinitely many input transitions. This holds even if a bounded number of runs is modeled.

Symbolic Analysis: Idea

- To replace the infinitely many transitions arising from an input action by a single *symbolic* transition.
- To represent the received message as a free variable.
- To add constraints on this variable as the computation proceeds.

Bibliography: [Huima99], [Amadio *et al.* 00], [Boreale 01], [Fiore, Abadi 01], [Boreale, Buscemi 02].

STA Overview

STA is a prototype for the analysis of cryptographic protocols, based on symbolic analysis. The current version of STA is written in ML and supports shared- and public-key cryptography, and hashing.

STA uses a systematic method to avoid the explicit construction of all possible execution traces in the exploration of the state-space. This results in:

- A more accurate protocol model.
- Increased efficiency.
- A more straightforward protocol formalization.

Outline of the Talk

- Explain the model underlying STA.
- Illustrate the use of STA.
- Show some results of the application of STA to analyze the Asymmetric Needham Schroeder protocol.
- Draw some conclusions on STA w.r.t efficiency, accuracy and usability.

STA Model: System States

A state of the system is a *configuration* $\langle t,\ P\rangle$ consisting of a trace t and a process P.

A *trace* is a sequence of actions such that any input message is deducible from the knowledge the environment has previously acquired. Formally, it is a closed string t such that

$$\forall\, t_1, t_2, a?M, \quad \text{if} \quad t = t_1 \cdot a?M \cdot t_2 \quad \text{then} \quad t_1 \vdash M.$$

STA Model: Communication Rules

System Execution:

$$(\textsc{Inp}) \quad \langle t,\ a?x.\,P\rangle \longrightarrow \langle t \cdot a?M,\ P[{}^{M}\!/x]\rangle \quad t \vdash M,\ M \text{ closed}$$

$$(\textsc{Out}) \quad \langle t,\ a!M.\,P\rangle \longrightarrow \langle t \cdot a!M,\ P\rangle$$

As noted, Rule (Inp) makes the transition relation *infinite-branching*, since M ranges over the infinite set of deducible messages.

Symbolic System Execution:

$$(\textsc{Inp}_S) \quad \langle t,\ a?x.P\rangle \longrightarrow \langle t \cdot a?x,\ P\rangle$$

Variable x is not instantiated. Rather, constraints on x are added as the computation proceeds.

STA Model: Other Rules

$$(\textsc{Let}) \quad \langle s,\ \mathsf{let}\, y = \zeta\, \mathsf{in}\, P\rangle \longrightarrow \langle s,\ P[{}^{M}\!/\!_{y}]\rangle \qquad [\![\zeta]\!] = M, M \text{ closed}$$

$$(\textsc{Match}) \quad \langle s,\ [\zeta = \eta]P\rangle \longrightarrow \langle s,\ P\rangle \qquad [\![\zeta]\!] = [\![\eta]\!] \neq \bot$$

$$(\textsc{Par}) \quad \frac{\langle s,\ P\rangle \longrightarrow \langle s',\ P'\rangle}{\langle s,\ P \parallel Q\rangle \longrightarrow \langle s',\ P' \parallel Q\rangle}$$

where $[\![\cdot]\!]$ is an evaluation function that maps terms to messages.

STA System Execution: Example

Let $P \stackrel{\text{def}}{=} a?x.\mathsf{let}\, y = \mathsf{dec}_k(x)\, \mathsf{in}\, P'$.

$$\begin{array}{rcl} \langle \epsilon,\ P\rangle & \longrightarrow & \langle a?x,\ \mathsf{let}\, y = \mathsf{dec}_k(x)\, \mathsf{in}\, P'\rangle \\ & \longrightarrow & \langle a?\{x'\}_k,\ P'[{}^{\{x'\}_k}\!/\!_{x}][{}^{x'}\!/\!_{y}]\rangle \end{array}$$

Note: The decryption of x using k is resolved by unifying x and $\{x'\}_k$ for some fresh x' (mgu $\theta = [{}^{\{x'\}_k}\!/\!_{x}]$); y is replaced by the result of the decryption, i.e. x'.

STA Model: Security Properties

- Expressed in terms of a correspondence assertion $\alpha \longleftarrow \beta$:

  For every generated trace, action α always occurs in the trace prior to action β.

- Verified by trace analysis.

Examples

Message Authentication: 'Every message encrypted with a key k that is accepted by B, has been previously sent by A'.

$$emit_A!(\{x\}_k) \longleftarrow final_B(\{x\}_k).$$

STA Specification

Messages:

{X}+K denotes asymmetric encryption; +K is the encryption key and −K is the corresponding decryption key.

Processes:

- `a!M`: output action of `M`;
- `a?M`: input action. `M` can be a generic a generic message pattern.
- `>>`: sequencing of actions;
- `(M is N).P`: tries to 'unify' `M` and `N`, and, if the matching succeeds, proceeds like P; otherwise stops";
- `P||Q`: parallel composition of `P` and `Q`;

STA Specifications (Cont.)

Configurations: `(L,P)`, where

- `L` is a list of I/O actions that provide the adversary with some initial knowledge;
- P is the parallel composition of all the participants to the protocol.

Properties: `a <-- b`, where `a` e `b` are actions.

Asymmetric Needham-Schroeder Protocol

- A = initiator; B = responder;
- $\{X\}_{+KY}$ = X encrypted under public key Y;
- $+KA$ and $+KB$ are assumed to be known by the beginning.

$$a1. \quad A \rightarrow B: \quad \{NA, A\}_{+KB}$$

$$a2. \quad B \rightarrow A: \quad \{NA, NB\}_{+KA}$$

$$a3. \quad A \rightarrow B: \quad \{NB\}_{+KB}$$

STA Code of ANSP

```
(* Initiator and Responder *)

val InA = a1!{NA,A}+KB >> a2?{NA,xNB}+KA >>
          a3!{xNB}+KB >> stop
         ||
          a’1!{N’A,A}+KI >> a’2?{N’A,xNI}+KA >>
          a’3!{xNI}+KI >> stop;

val ReB = b1?{yNA,A}+KB >> b2!{yNA,NB}+KA >>
          b3?{NB}+KB >> stop;
```


STA Code of ANSP (Cont.)

```
(* The initial configuration *)

val NS  = ( [ disclose!(KI,+KA,+KB,A,B,I) ],
          (InA || ReB) ) ;

(* Prop.1:  A is correctly authenticated to B *)

val AuthAtoB = ( a3!u <-- b3?u );

(* Prop.2:  B is correctly authenticated to A *)

val AuthBtoA = ( b2!u <-- a2?u );
```

STA Analysis of ANSP

```
> CHECK NS AuthAtoB;

An attack was found:

  disclose!(KI,+KA,+KB,A,B,I). a'1!{N'A,A}+KI.
  a1!{NA,A}+KB.b1?{N'A,A}+KB. b2!{N'A,NB}+KA.
  a'2?{N'A,NB}+KA. a'3!{NB}+KI. b3?{NB}+KB.

26 symbolic configurations reached.
```


A well-known attack to ANSP

The previous trace corresponds to the well-known attack shown below. It consists of two *runs* of the protocol, a e b: in run a, A tries to establish a session with I; in run b, I establishes a false session with B, pretending to be A.

$$\begin{array}{rcl}
a1. & A \to I: & \{NA, A\}_{+KI} \\
b1. & I(A) \to B: & \{NA, A\}_{+KB} \\
b2. & B \to I(A): & \{NA, NB\}_{+KA} \\
a2. & I \to A: & \{NA, NB\}_{+KA} \\
a3. & A \to I: & \{NB\}_{+KI} \\
b3. & I(A) \to B: & \{NB\}_{+KB}
\end{array}$$

Corrected ANSP

The protocol can be fixed by inserting explicit identities inside each encrypted message. Below is the corrected STA version of ANSP:

```
(* Initiator and Responder *)

val InA'= a1!{NA,A,B}+KB >> a2?{NA,xNB,A,B}+KA
          >> a3!{xNB,A,B}+KB >> stop
        ||
          a'1!{N'A,A,I}+KI >>
          a'2?{N'A,x'NI,A,I}+KA >>
          a'3!{x'NI,A,I}+KI >> stop;

val ReB'= b1?{yNA,A,B}+KB >> b2!{yNA,NB,A,B}+KA
          >> b3?{NB,A,B}+KB >> stop;
```


Corrected ANSP (Cont.)

```
(* The initial configuration *)

val NS' = ( [ disclose!(KI,+KA,+KB,A,B,I) ],
          (InA' || ReB') ) ;

(* Prop.1:  A is correctly authenticated to B *)

val AuthAtoB = ( a3!u <-- b3?u );
```

Trusted Third-Party Needham-Schroeder Protocol

A = initiator; B = responder; S = server; `(X)+sigY` means "X electronically signed by Y"; `+sigY` is the private key of Y; `-sigY` is the corresponding public key.

$$
\begin{array}{lll}
a1. & A \rightarrow S: & B \\
a2. & S \rightarrow A: & \{+KB, B\}{+sigS} \\
a3. & A \rightarrow B: & \{NA, A\}_{+KB} \\
a4. & B \rightarrow S: & A \\
a5. & S \rightarrow B: & \{+KA, A\}{+sigS} \\
a6. & B \rightarrow A: & \{NA, NB\}_{+KA} \\
a7. & A \rightarrow B: & \{NB\}_{+KB}
\end{array}
$$

A Type Flaw on TPNSP

```
val ReB = b1?{yNA,A}+KB >> b2!A >>
    b3?{ypkA,A}+SigS >> b4!{yNA,NB,B}(ypkA) >>
    b5?{NB}+KB >> stop;

(* Prop.1:  A is correctly authenticated to B *)
val AuthAtoB = ( a5!u <-- b5?u);


> CHECK NS AuthAtoB;

An attack was found:
  out!{A,B,-SigS,+KA,+KB}.  a1!B. b1?{yNA,A}+KB.
  b2!A.s'1?A. s'2!{+KA,A}+SigS. b3?{+KA,A}+SigS.
  b4!{yNA,NB,B}+KA. s1?B. s2!{+KB,B}+SigS.
  a'1?{yNA,NB,B}+KA. a'2!{NB,B}.  b5?{NB}+KB
```

Efficiency

As noted, symbolic analysis does not suffer from any state explosion problem depending on message exchange.

In the table is reported the number of states generated respectively by Murφ and STA in the analysis of ANSP:

Initiators	**Responders**	**Murφ**	**STA**
1	1	1706	60
2	1	17,277	411
2	2	514,550	24,655

Accuracy

- STA makes no assumption on typing or on the number of messages the adversary may generate. As a consequence, the model generated makes no approximation with respect to the infinite-state standard model.

Usability

- Currently, the user interface of STA is rather rudimentary, thus specifying a protocol requires a certain acquaintance with process algebras.

- No description of the adversary must be explicitly given, apart from its initial knowledge.

Current and Future Work

- Extension of the model underlying STA to analyze protocols based on low-level cryptographic operations, such as modular exponentiation (Diffie-Hellman key exchange).
- Optimization of STA code and testing real-life case studies.

References

The distribution of STA is downloadable at the URL:

`http://www.dsi.unifi.it/~boreale/tool.html`

A basic documentation and examples on the STA analysis of some cryptographic protocols (WMF, Otway-Rees, Kerberos, etc.) are also available.