Models and analysis of security protocols
1st Semester 2011-2012

Pascal Lafourcade

Université Joseph Fourier, Verimag

November 2011
Last Time

Done by Marie-Laure

- Needham Schroeder ...
- Role based presentation
- Dolev-Yao
Outline of Today:

Dolev Yao’s Intruder
Outline of Today:

Dolev Yao’s Intruder

Passive Intruder: Intruder Deduction Problem
Outline of Today:

Dolev Yao’s Intruder

Passive Intruder: Intruder Deduction Problem

Active Intruder: Security Problem
Outline of Today:

Dolev Yao’s Intruder

Passive Intruder: Intruder Deduction Problem

Active Intruder: Security Problem

Tools

AVISPA
Scyther
Proverif
Comparaison
Outline of Today:

Dolev Yao’s Intruder

Passive Intruder: Intruder Deduction Problem

Active Intruder: Security Problem

Tools
  AVISPA
  Scyther
  Proverif
  Comparaison

Bonus: Undecidability
Outline of Today:

Dolev Yao’s Intruder

Passive Intruder: Intruder Deduction Problem

Active Intruder: Security Problem

Tools
  - AVISPA
  - Scyther
  - Proverif
  - Comparaison

Bonus: Undecidability

Conclusion
Outline

Dolev Yao’s Intruder

Passive Intruder: Intruder Deduction Problem

Active Intruder: Security Problem

Tools

  AVISPA
  Scyther
  Proverif
  Comparaison

Bonus: Undecidability

Conclusion
The Intruder is the Network (Worst Case)
The Intruder is the Network (Worst Case)

Passive: Intruder deduction problem
The Intruder is the Network (Worst Case)

Passive: Intruder deduction problem

Active: Security problem

Listen

Intercept message

(Re)play message

Delete message
The Intruder is the Network (Worst Case)

Passive: Intruder deduction problem
Active: Security problem

Listen
Intercept message
(Re)play message
Delete message

Intruder Capabilities (Dolev-Yao Model 80’s)

- Encryption, Decryption with a key
- Pairing, Projection.
Dolev-Yao 1982

- Intruder controls the network and can:
  - intercept messages
  - modify messages
  - block messages
  - generate new messages
  - insert new messages

- Perfect cryptography:
  - Abstraction with terms algebra
  - Decryption only if inverse key is known

- Protocol has
  - Arbitrary number of principals
  - Arbitrary number of parallel sessions
  - Messages with arbitrary size
# Dolev-Yao Deduction System

### Deduction System: $T_0 \vdash ? s$

- **(A)** \[
\frac{u \in T_0}{T_0 \vdash u}
\]

- **(P)** \[
\frac{T_0 \vdash u \quad T_0 \vdash v}{T_0 \vdash \langle u, v \rangle}
\]

- **(C)** \[
\frac{T_0 \vdash u \quad T_0 \vdash v}{T_0 \vdash \{u\}_v}
\]

- **(UL)** \[
\frac{T_0 \vdash \langle u, v \rangle}{T_0 \vdash u}
\]

- **(UR)** \[
\frac{T_0 \vdash \langle u, v \rangle}{T_0 \vdash v}
\]

- **(D)** \[
\frac{T_0 \vdash \{u\}_v \quad T_0 \vdash v}{T_0 \vdash u}
\]
Example: $T_0 \vdash ? s$

Example

$T_0 = \{k, \{b\}_{c}, \langle a, \{c\}_k \rangle \}$ and $s = b$
Example: \( T_0 \vdash ? \; s \)

\[
T_0 = \{ k, \{b\}_c, \langle a, \{c\}_k \rangle \} \text{ and } s = b
\]

\[
\begin{align*}
\frac{(A) \{b\}_c \in T_0}{T_0 \vdash \{b\}_c} & \quad & \frac{(A) \langle a, \{c\}_k \rangle \in T_0}{T_0 \vdash \langle a, \{c\}_k \rangle} & \quad & \frac{(A) k \in T_0}{T_0 \vdash k} \\
\frac{(D) \{a\}_c \in T_0}{T_0 \vdash \{a\}_c} & \quad & \frac{(UR) T_0 \vdash \{c\}_k}{T_0 \vdash c} & \quad & \frac{}{T_0 \vdash b}
\end{align*}
\]
Outline

Dolev Yao’s Intruder

Passive Intruder: Intruder Deduction Problem

Active Intruder: Security Problem

Tools
  AVISPA
  Scyther
  Proverif
  Comparaison

Bonus: Undecidability

Conclusion
Syntactic Subterms

Equivalent definition for Dolev Yao model

\( S(t) \) is the smallest set such that:

\[ \begin{align*}
& \triangleright t \in S(t) \\
& \triangleright \langle u, v \rangle \in S(t) \Rightarrow u, v \in S(t) \\
& \triangleright \{ u \}_v \in S(t) \Rightarrow u, v \in S(t)
\end{align*} \]

Exercise:

\[ \begin{align*}
& \triangleright \text{Let } t = \{ \langle a, \{ b \}_{k_2} \rangle \}_{k_1}
\end{align*} \]
Syntactic Subterms

Equivalent definition for Dolev Yao model

$S(t)$ is the smallest set such that:

- $t \in S(t)$
- $\langle u, v \rangle \in S(t) \Rightarrow u, v \in S(t)$
- $\{ u \}_v \in S(t) \Rightarrow u, v \in S(t)$

Exercise:

- Let $t = \{ \langle a, \{ b \}_{k_2} \rangle \}_{k_1}$

$$S(t) = \{ t, a, b, k_1, k_2, \{ b \}_{k_2}, \langle a, \{ b \}_{k_2} \rangle \}$$
Definition of S-Locality

- A proof $P$ of $T_0 \vdash s$ is S-local:

\[ \forall n \in P, n \in S(T_0 \cup \{s\}) \]
Definition of S-Localcy

- A proof $P$ of $T_0 \vdash s$ is S-local:

$$\forall n \in P, \ n \in S( T_0 \cup \{s\})$$

S-Local Proof:

A proof $P$ of $T \vdash w$ is **S-local** if all nodes are in $S( T \cup \{w\})$. 
Definition of S-Locality

- A proof $P$ of $T_0 \vdash s$ is S-local:

$$\forall n \in P, n \in S(T_0 \cup \{s\})$$

S-Local Proof:

A proof $P$ of $T \vdash w$ is **S-local** if all nodes are in $S(T \cup \{w\})$.

S-Locality:

A proof system is **S-local** if whenever there is a proof of $T \vdash w$ then there is also a S-local proof of $T \vdash w$. 
Locality Idea [MacAllester’93]

P a $S$-local proof of $T_0 \vdash s$

$S(T_0 \cup \{s\})$
Locality Idea [MacAllester’93]

P a S-local proof of $T_0 \vdash s$

$S(T_0 \cup \{s\})$
Locality Idea [MacAllester’93]

P a $S$-local proof of $T_0 \vdash s$

$S(T_0 \cup \{s\})$
Locality Idea [MacAllester’93]

P a S-local proof of $T_0 \vdash s$

$S(T_0 \cup \{s\})$

Intruder Deduction Problem: $T_0 \vdash ? s$

- S-locality
- One-step deductibility
Example: a local proof of $T_0 \vdash s$

Example

$T_0 = \{k, \{b\}_c, \langle a, \{c\}_k \rangle\}$ and $s = b$

\[
\begin{align*}
\frac{(A) \langle a, \{c\}_k \rangle \in T_0}{T_0 \vdash \langle a, \{c\}_k \rangle} & \quad (A) \frac{k \in T_0}{T_0 \vdash k} \\
(UR) \frac{T_0 \vdash \langle a, \{c\}_k \rangle}{T_0 \vdash \{c\}_k} & \quad (D) \frac{T_0 \vdash \langle a, \{c\}_k \rangle}{T_0 \vdash \{c\}_k} \\
(D) \frac{T_0 \vdash \{c\}_k}{T_0 \vdash c} & \quad (D) \frac{T_0 \vdash c}{T_0 \vdash b} \\
(UR) \frac{T_0 \vdash \{c\}_k}{T_0 \vdash \{b\}_c} & \quad (A) \frac{\{b\}_c \in T_0}{T_0 \vdash \{b\}_c}
\end{align*}
\]
Example: a local proof of $T_0 \vdash s$

Example

$T_0 = \{ k, \{ b \}_c, \langle a, \{ c \}_k \rangle \}$ and $s = b$

\[
\begin{align*}
(A) \quad & \langle a, \{ c \}_k \rangle \in T_0 \\
(UR) \quad & \frac{T_0 \vdash \langle a, \{ c \}_k \rangle}{T_0 \vdash \{ c \}_k} \\
(D) \quad & \frac{T_0 \vdash \{ c \}_k}{T_0 \vdash c} \\
(D) \quad & \frac{T_0 \vdash c}{T_0 \vdash b} \\
(A) \quad & k \in T_0 \\
(A) \quad & \{ b \}_c \in T_0
\end{align*}
\]

\[S(T_0 \cup \{ s \}) = T_0 \cup \{ a, b, c, \{ c \}_k \}\]
Locality Theorem

**Theorem of Locality [McAllester 93]**

If a proof system $P$ is SyntacticSubterm-local then there is a $P$-time procedure to decide the deductibility in $P$. 
Locality Theorem

Theorem of Locality [McAllester 93]

If a proof system $P$ is SyntacticSubterm-local then there is a $P$-time procedure to decide the deductibility in $P$.

Restrictions:

- Deduction system must be finite
- Use just syntactic subterms
Adapted McAllester Results

McAllester’s Algorithm

Input : $T_0, w$

$T \leftarrow T_0;$

while ($\exists s \in S(T_0, w)$ such that $T \vdash \leq 1 s$ and $s \notin T$)

$T \leftarrow T \cup \{s\};$

Output : $w \in T$

Theorem

Let be $P$ a proof system, if:

- the size of $S(T)$ is polynomial in the size of $T$,
- $P$ is S-local,
- one-step deducibility is P-time decidable,

then provability in the proof system $P$ is P-time decidable.
Locality Theorem

Theorem of Locality [McAllester 93]

If a proof system $P$ is SyntacticSubterm-local then there is a $P$-time procedure to decide the deductibility in $P$. 
Locality Theorem

Theorem of Locality [McAllester 93]

If a proof system $P$ is SyntacticSubterm-local then there is a $P$-time procedure to decide the deductibility in $P$.

Result:

Dolev Yao deduction system is $S$-local.
Outline

Dolev Yao’s Intruder

Passive Intruder: Intruder Deduction Problem

Active Intruder: Security Problem

Tools

   AVISPA
   Scyther
   Proverif
   Comparaison

Bonus : Undecidability

Conclusion
The Intruder is the Network (Worst Case)

Passive: Intruder deduction problem
The Intruder is the Network (Worst Case)

Passive: Intruder deduction problem

Active Intruder Security problem

- intercept messages (add messages to his knowledge)
- Play messages from his knowledge
- Start new sessions

Execution tree has:

- infinite branching (size of messages is not bounded)
- infinite depth (number of sessions is not bounded)
Active Intruder with bounded number of sessions

- Theoretically: decidable
- Interesting practically:
  - Find flaws
  - Usually attacks use few sessions!
Protocol: Needham Schroeder

\[\{N_A, A\}_K_B\] 

\[\{N_A, N_B\}_K_A\] 

\[\{N_B\}_K_B\]
Model: actions, roles and protocol

Definition (Action)

An action is a couple \((\text{recv}(u), \text{send}(v))\) such that
\(u \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \cup \{\text{init}\}, \ v \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \cup \{\text{stop}\}\). Denoted \((u \rightarrow v)\).
Model: actions, roles and protocol

Definition (Action)

An action is a couple \((\text{recv}(u), \text{send}(v))\) such that \(u \in T(F, X) \cup \{\text{init}\}\), \(v \in T(F, X) \cup \{\text{stop}\}\). Denoted \((u \rightarrow v)\).

Example

First and last actions of Needham Schroeder

- \((\text{init}, X_b \rightarrow \{N_a, A\}_{pk(X_b)})\)
- \((\{N_b\}_{pk(B)} \rightarrow \text{stop})\)
Model: actions, roles and protocol

Definition (Role)

A role is a finite sequence of actions:

\[(u_1 \rightarrow v_1), \ldots, (u_n \rightarrow v_n)\]

such that \(\text{vars}(v_i) \subseteq \bigcup_{1 \leq j \leq i} \text{vars}(u_j)\).
Model: actions, roles and protocol

Definition (Role)

A role is a finite sequence of actions:

\[(u_1 \rightarrow v_1), \ldots, (u_n \rightarrow v_n)\]

such that \(\text{vars}(v_i) \subseteq \bigcup_{1 \leq j \leq i} \text{vars}(u_j)\).

Definition (Protocol)

A protocol \(P\) is a finite set of roles: \(P = \{R_1, \ldots, R_k\}\)
1st Example:

Example (Needham-schroeder)

1. $A \rightarrow B : \{N_a, A\}_{pk(B)}$
2. $B \rightarrow A : \{N_a, N_b\}_{pk(A)}$
3. $A \rightarrow B : \{N_b\}_{pk(B)}$

Write down each agent’s role description, this $A$ talks with anybody.

$R_A = (init, X_b \rightarrow \{N_a, A\}_{pk(X_b)}),$
\[
(\{N_a, x_{N_b}\}_{pk(A)} \rightarrow \{x_{N_b}\}_{pk(X_b)})$

$R_B = (\{x_{N_a}, x_A\}_{pk(B)} \rightarrow \{x_{N_a}, N_b\}_{pk(x_A)})$
\[
(\{N_b\}_{pk(B)} \rightarrow stop)$
Semantics

**Definition (States)**

- $T$ is a set of ground terms (intruder knowledge)
- $P$ a protocol

A state is a couple $(T, P)$

**Definition (Transition)**

Is a relation between states $(T, P) \rightarrow^\sigma (T', P')$

- $P = \bigcup_i^k R_i$, take an $i$ : $R_i = (u_i \rightarrow v_i)$
- Possible $\sigma : T \vdash u_i \sigma$  ($\text{dom}(\sigma) = \text{vars}(u_i)$)
- Update intruder knowledge : $T' = T \cup \{v_i \sigma\}$
- Update Protocol $\forall j \neq i, R_j \in P', P' = (P \setminus \{R_i\}) \cup R_j \sigma$
Example

Simple Let $T = \{a, b, k_I\}$ and $P = \{R\}$ where $R = (\langle x, y \rangle \rightarrow \langle \{y\}_{k}, x \rangle), (z \rightarrow \langle x, \langle y, z \rangle \rangle)$.

$\triangleright (T, P) \rightarrow^\sigma (T \cup \{\langle b \rangle_{k}, a\}, \{(z \rightarrow \langle a, \langle b, z \rangle \rangle)\})$

$\sigma = \{x \leftarrow a, y \leftarrow b\}$
Example

Simple Let \( T = \{a, b, k_I\} \) and \( P = \{R\} \) where
\[
R = (\langle x, y \rangle \rightarrow \langle \{y\}_k, x \rangle), (z \rightarrow \langle x, \langle y, z \rangle \rangle).
\]

\( (T, P) \rightarrow^\sigma (T \cup \{\langle\{b\}_k, a\rangle\}, \{(z \rightarrow \langle a, \langle b, z \rangle \rangle)\}) \)
\[
\sigma = \{x \leftarrow a, y \leftarrow b\}
\]

\( (T, P) \rightarrow^\sigma (T \cup \{\langle\{a\}_k\}_k, a\rangle\}, \{(z \rightarrow \langle a, \langle\{a\}_k, z\rangle \rangle)\}) \)
\[
\sigma = \{x \leftarrow a, y \leftarrow \{a\}_k\}
\]
Example

Simple Let $T = \{a, b, k_1\}$ and $P = \{R\}$ where $R = (\langle x, y \rangle \rightarrow \langle \{y\}_k, x \rangle), (z \rightarrow \langle x, \langle y, z \rangle \rangle)$. 

\begin{itemize}
  \item $(T, P) \rightarrow^\sigma (T \cup \{\langle \{b\}_k, a \rangle\}, \{(z \rightarrow \langle a, \langle b, z \rangle \rangle)\})$
  \[
  \sigma = \{x \leftarrow a, y \leftarrow b\}
  \]
  \item $(T, P) \rightarrow^\sigma (T \cup \{\langle \{a\}_{k_1}, a \rangle\}, \{(z \rightarrow \langle a, \langle \{a\}_{k_1}, z \rangle \rangle)\})$
  \[
  \sigma = \{x \leftarrow a, y \leftarrow \{a\}_{k_1}\}
  \]
  \item $(T, P) \not\rightarrow^\sigma (T \cup \{\langle \{a\}_k \rangle_k, a \rangle\}, \{(z \rightarrow \langle a, \langle \{a\}_k, z \rangle \rangle)\})$
  \[
  \sigma = \{x \leftarrow a, y \leftarrow \{a\}_k\}
  \]
\end{itemize}
Simple Let \( T = \{a, b, k_I\} \) and \( P = \{R\} \) where \( R = (\langle x, y \rangle \rightarrow \langle \{y\}_k, x \rangle), (z \rightarrow \langle x, \langle y, z \rangle \rangle) \).

\[
\begin{align*}
\triangleright & \quad (T, P) \rightarrow^\sigma (T \cup \{\langle \{b\}_k, a \rangle\}, \{(z \rightarrow \langle a, \langle b, z \rangle \rangle)\}) \\
& \quad \sigma = \{x \leftarrow a, y \leftarrow b\} \\
\triangleright & \quad (T, P) \rightarrow^\sigma (T \cup \{\langle \{a\}_{k_I} \rangle_k, a \}, \{(z \rightarrow \langle a, \langle \{a\}_{k_I}, z \rangle \rangle)\}) \\
& \quad \sigma = \{x \leftarrow a, y \leftarrow \{a\}_{k_I}\} \\
\triangleright & \quad (T, P) \nrightarrow^\sigma (T \cup \{\langle \{a\}_k \rangle_k, a \}, \{(z \rightarrow \langle a, \langle \{a\}_k, z \rangle \rangle)\}) \\
& \quad \sigma = \{x \leftarrow a, y \leftarrow \{a\}_k\}
\end{align*}
\]

Each branch has a finite depth (protocol are finite),
Example

Simple Let $T = \{a, b, k_I\}$ and $P = \{R\}$ where

$R = (\langle x, y \rangle \rightarrow \langle \{y\}_k, x \rangle), (z \rightarrow \langle x, \langle y, z \rangle \rangle)$.

- $(T, P) \rightarrow^\sigma (T \cup \{\langle \{b\}_k, a \rangle\}, \{(z \rightarrow \langle a, \langle b, z \rangle \rangle)\})$
  $\sigma = \{x \leftarrow a, y \leftarrow b\}$

- $(T, P) \rightarrow^\sigma (T \cup \{\langle \{a\}_k \rangle_k, a \rangle\}, \{(z \rightarrow \langle a, \langle \{a\}_k, z \rangle \rangle)\})$
  $\sigma = \{x \leftarrow a, y \leftarrow \{a\}_k\}$

- $(T, P) \not\rightarrow^\sigma (T \cup \{\langle \{a\}_k \rangle_k, a \rangle\}, \{(z \rightarrow \langle a, \langle \{a\}_k, z \rangle \rangle)\})$
  $\sigma = \{x \leftarrow a, y \leftarrow \{a\}_k\}$

Each branch has a finite depth (protocol are finite), but possibly a infinite branching (infinite number of terms).
Preservation of the secrecy

**Definition (Secrecy)**

Let $T_1$ be a ground set of terms (Initial knowledge of the intruder). A protocol $P$ does not preserve the secrecy of a ground term $s$ for $T_1$ if there does not exist a state $(T', P')$, such that

- $T' \vdash s$
- $(T_1, P) \rightarrow^* (T', P')$

where $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$.

If there does not exist a such state $(T', P')$ we say that $P$ preserves the secrecy of $s$ for the initial intruder knowledge $T_1$. 
### Interleaving

#### Definition (Partial Order $<_{P}$)

A protocol $P$ define a partial order $<_{P}$ on actions of $P$, s.t

\[(u_i \rightarrow v_i) <_{P} (u_j \rightarrow v_j)\]

if $R \in P$, $R = (u_1 \rightarrow v_1) \ldots (u_i \rightarrow v_i) \ldots (u_j \rightarrow v_j) \ldots (u_n \rightarrow v_n)$ $(1 \leq i \leq j \leq n)$. 

Interleaving

Definition (Partial Order $<_P$)

A protocol $P$ define a partial order $<_P$ on actions of $P$, s.t

$$(u_i \to v_i) <_P (u_j \to v_j)$$

if $R \in P$, $R = (u_1 \to v_1) \ldots (u_i \to v_i) \ldots (u_j \to v_j) \ldots (u_n \to v_n)$ ($1 \leq i \leq j \leq n$).

Definition (Execution Order $<_E$)

An execution order $<_E$ of $P$ is a total order on the subset $A$ of actions of $P$, compatible with $<_P$ and stable by predecessor, i.e.

if $b \in A$ et $a <_P b$ then $a \in A$ and $a <_E b$

It corresponds to an interleaving of roles.
Secrecy

Definition (Secrecy over $<_E$)

Let an execution order $<_E$ of $P$. We assume that

$$(u_1 \to v_1) <_E \ldots <_E (u_n \to v_n)$$

Does not preserve the secrecy of $s$, given $T_1$ if there exists $\sigma_1, \ldots, \sigma_n$ such that

$$(T_1, P) \to (T_1 \cup \{v_1\sigma_1\}, P_1) \to \ldots \to (T_1 \cup \{v_1\sigma_1, \ldots, v_n\sigma_n\}, P_n)$$

and $T_1 \cup \{v_1\sigma_1, \ldots, v_n\sigma_n\} \vdash s$. 
Outline

Dolev Yao’s Intruder

Passive Intruder: Intruder Deduction Problem

Active Intruder: Security Problem

Tools

AVISPA
Scyther
Proverif
Comparaison

Bonus: Undecidability

Conclusion
Necessity of Tools

- Protocols are small recipes.
- Non trivial to design and understand.
- The number and size of new protocols.
- Out-pacing human ability to rigorously analyze them.

**GOAL**: A tool is finding flaws or establishing their correctness.
- completely automated,
- robust,
- expressive,
- and easily usable.

**Existing Tools**: AVISPA, Scyther, Proverif, Hermes, Casper/FDR, Murphi, NRL ...

Comparison of Tools Dealing with Algebraic Properties ?
Complexity

Complexity depends on intruder capabilities. In classical Dolev-Yao intruder model (pair + encryption), we have the following results:

- **Passive Intruder**
  Problem is **polynomial**

- **Bounded Number of sessions**
  Problem is **NP-complete**
  Tools can verify 3-4 sessions: useful to find flaws! OFMC, Cl-Atse, SATMC, FDR, Athena...

- **Unbounded Number of sessions**
  Problem is in general **undecidable**
  Tools are correct, but uncomplete (can find false attacks, cannot terminate) Proverif, TA4SP, Scyther.
Formal Landscape and Our Focus

Security Protocol Analysis

Formal Models
- Belief Logics
- Semi-automated
  - Inductive Proofs
  - Model Checking

Computational Models
- Dolev-Yao (perfect cryptography)
- Probabilistic cryptographic view
  - Cryptographically faithful proofs

N.B. Challenging as general problem is \textit{undecidable} due e.g. to the possibility of unbounded number of protocol sessions.
Tools studied next time

- **Avispa**: Plateform with 4 tools: OFMC, CL-AtSe, SATMC, and TA4SP.
- **Proverif**: Analyses unbounded number of session using over-approximation with Horn Clauses.
- **Scyther**: Verifies bounded and unbounded number of session with backwards search based on partially ordered patterns.
The AVISPA Tool: Architecture

High-Level Protocol Specification Language (HLPSL) → Translator

HLPSL2IF → Intermediate Format (IF)

Intermediate Format (IF) → Output Format (OF)

On-the-fly Model-Checker OFMC → CL-based Attack Searcher AtSe → SAT-based Model-Checker SATMC → Tree Automata-based Protocol Analyser TA4SP
The AVISPA Tool: the Back-Ends

On-the-fly Model-Checker (OFMC) employs several symbolic techniques to explore the state space in a demand-driven way.

CL-AtSe (Constraint-Logic-based Attack Searcher) applies constraint solving with simplification heuristics and redundancy elimination techniques.

The SAT-based Model-Checker (SATMC) builds a propositional formula encoding all the possible traces (of bounded length) on the protocol and uses a SAT solver.

TA4SP (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols) approximates the intruder knowledge by using regular tree languages and rewriting to produce under and over approximations.
The Web Interface  www.avispa-project.org
Results in AVISPA

AVISPA Tool Summary

OFMC : UNSAFE
CL-AtSe : UNSAFE
SATMC : UNSAFE
TA4SP : INCONCLUSIVE

Refer to individual tools output for details

View detailed output or Return to file selection

File Selection
MSC Description of the Attack in AVISPA

\[
\text{start} \quad \{na0(a,6).a\}_{ki} \quad \{na0(a,6).a\}_{kb} \\
\quad \{na0(a,6).nb0(b,4)\}_{ka} \\
\quad \{na0(a,6).nb0(b,4)\}_{ka} \\
\quad \{nb0(b,4)\}_{ki}
\]
Needham-Schroeder : Alice

role alice (A, B : agent,
    Ka, Kb : public_key,
    SND, RCV: channel (dy))

played_by A def=
    local State : nat,
    Na, Nb : text

init State := 0

transition

0. State = 0 \ RCV(start) =>
   State' := 2 \ Na' := new() \ SND({Na'.A}_Kb)
   \ secret(Na',na,{A,B})

2. State = 2 \ RCV({Na.Nb'}_Ka) =>
   State' := 4 \ SND({Nb'}_Kb)

end role
Needham-Schroeder: Bob

role bob(A, B : agent,
    Ka, Kb : public_key,
    SND, RCV : channel (dy))

played_by B def=
    local State : nat,
    Na, Nb : text

init State := 1

transition

    1. State = 1 \ RCV({Na'.A}_Kb) =>
        State' := 3 \ Nb' := new() \ SND({Na'.Nb'}_Ka)
        \ secret(Nb',nb,{A,B})

    3. State = 3 \ RCV({Nb}_Kb) =>

end role
Needham-Schroededer: Session, Environment & Goal

role session(A, B: agent, Ka, Kb: public_key) def=
    local SA, RA, SB, RB: channel (dy)
    composition
    alice(A,B,Ka,Kb,SA,RA) /\
    bob (A,B,Ka,Kb,SB,RB)
end role

role environment() def=
    const a, b : agent,
    ka, kb, ki : public_key,
    na, nb, : protocol_id
    intruder_knowledge = {a, b, ka, kb, ki, inv(ki)}
    composition
    session(a,b,ka,kb) /\
    session(a,i,ka,ki)
    /\
    session(i,b,ki,kb)
end role

goal secrecy_of na, nb
end goal
environment()
Agent: names of principles

public key: asymmetric keys

symmetric key: symmetric keys

nat: natural numbers

function: to model hash functions etc

bool: Boolean values for modeling flags

Kinds of variables:

State variables: Those that are within the scope of a role.

Declared at the top of a role

Unprimed versions indicate current state

Primed versions indicate next state
Role Definition

1. Role declaration: its name and the list of formal arguments, along with (in the case of basic roles) a player declaration;
2. Declaration of local variables and ownership rules, if any;
3. Initialization of variables, if required;
4. Declaration of accepting states, if any;
5. Knowledge declarations, if applicable;
6. Either (optionally) : a transition section (for basic roles) or a composition section (for composed roles).
Scyther

- Alternative: backwards search based on patterns
  - Security properties represented by claim events in the protocol.
  - Supports symmetric and asymmetric keys, cryptographic hash functions, key-tables, multiple protocols in parallel, composed keys, etc (but no user-definable algebraic functions)
  - Can perform unbounded verification of protocols
  - Provides complete characterization of protocol roles: Answer to: “after execution of a protocol role, what events must also have occurred?”

- Also state-of-art. Freely available for download for Windows, Linux and Mac OS X.
- Will be used in the exercise sessions.
Input

```plaintext
protocol ns3(I,R) {
  role I {
    const ni: Nonce;
    var nr: Nonce;
    send_1(I,R, {ni,I}pk(R) );
    read_2(R,I, {ni,nr}pk(I) );
    send_3(I,R, {nr}pk(R) );
    claim_i1(I,Secret,ni);
    claim_i2(I,Nisynch);
  }
  role R {
    var ni: Nonce;
    const nr: Nonce;
    read_1(I,R, {ni,I}pk(R) );
    send_2(R,I, {ni,nr}pk(I) );
    read_3(I,R, {nr}pk(R) );
    claim_r1(R,Secret,ni);
    claim_r2(R,Nisynch);
  }
}
```

Output (<0.02 seconds)

```
Run #1
Alice in role R
I -> Bob
R -> Eve
Const nr#1
Var ni -> ni#2
send_1 to Eve
{ ni#2,Bob }pk(Eve)
read_2 from Eve
{ ni#2,nr#1 }pk(Bob)
decrypt
nr#1
encrypt
ni#2
Initial intruder knowledge
pk(Alice)
sk(Eve)
```
Proverif

Proverif uses spi-calculus or Horn Clauses

analyser -in horn toto.pv OR analyser -in pi toto.pv
Proverif: Horn Clauses

(* Needham Shroeder Lowe *)
pred c/1 elimVar,decompData.
nounif c:x.
fun pk/1.
fun encrypt/2.

query c:secret[].
reduc

(* Initialization *)
c:c[];
c:pk(sA[]);
c:pk(sB[]);

c:x & c:encrypt(m,pk(x)) -> c:m;
c:x -> c:pk(x);
c:x & c:y -> c:encrypt(x,y);
Proverif: Horn Clauses

(* The protocol *)

(* A *)
c:pk(x) \rightarrow c:\text{encrypt}((\text{Na[pk(x)]}, \text{pk(sA[])}), \text{pk(x)});

c:pk(x) \& c:\text{encrypt}((\text{Na[pk(x)]}, y), \text{pk(sA[])}))
\rightarrow c:\text{encrypt}((y, k[pk(x)]), \text{pk(x)});

(* B *)
c:\text{encrypt}((x, y), \text{pk(sB[])}))
\rightarrow c:\text{encrypt}((x, \text{Nb[x,y]}, \text{pk(sB[]})), y);

c:\text{encrypt}((x,\text{pk(sA[])}))
\& c:\text{encrypt}((\text{Nb[x, pk(sA[])]}, z), \text{pk(sB[])}))
\rightarrow c:\text{encrypt}(\text{secret[]}, \text{pk(z)}).
goal reachable: c:secret[]

rule 7 c:secret[]
  any c:x_182
rule 1 c:encrypt(secret[], pk(x_182))
  rule 5 c:encrypt((Na[pk(x_168)], pk(sA[])), pk(sB[]))
    2-tuple c:(Na[pk(x_168)], pk(sA[]))
      0-th c:Na[pk(x_168)]
      rule 7 c:(Na[pk(x_168)], pk(sA[]))
        any c:x_168
        rule 4 c:encrypt((Na[pk(x_168)], pk(sA[])), pk(x_168))
          rule 6 c:pk(x_168)
            any c:x_168
          rule 9 c:pk(sA[])
        rule 8 c:pk(sB[])
      rule 5 c:encrypt((Nb[Na[pk(x_168)], pk(sA[])], x_182), pk(sB[]))
        2-tuple c:(Nb[Na[pk(x_168)], pk(sA[])], x_182)
          0-th c:Nb[Na[pk(x_168)], pk(sA[])]
          rule 7 c:(Nb[Na[pk(x_168)], pk(sA[])], k[pk(x_168)])
            any c:x_168
          rule 3 c:encrypt((Nb[Na[pk(x_168)], pk(sA[])], k[pk(x_168)]), pk(x_168))
            rule 6 c:pk(x_168)
              any c:x_168
          rule 2 c:encrypt((Na[pk(x_168)], Nb[Na[pk(x_168)], pk(sA[])]), pk(sA[]))
            rule 5 c:encrypt((Na[pk(x_168)], pk(sA[])), pk(sB[]))
              2-tuple c:(Na[pk(x_168)], pk(sA[]))
                0-th c:Na[pk(x_168)]
                rule 7 c:(Na[pk(x_168)], pk(sA[]))
                  any c:x_168
                rule 4 c:encrypt((Na[pk(x_168)], pk(sA[])), pk(x_168))
                  rule 6 c:pk(x_168)
                    any c:x_168
                rule 9 c:pk(sA[])
              rule 8 c:pk(sB[])
          any c:x_182
        rule 8 c:pk(sB[])
What is the spi-calculus?

The **spi-calculus** is an extension of the pi-calculus designed to represent cryptographic protocols.

The **pi-calculus** is a process calculus:

- processes *communicate*: they can send and receive messages on channels several processes can *execute in parallel*.
- In the pi-calculus, messages and channels are *names*, that is, atomic values $a, b, c, \ldots$. 
What is the spi-calculus? (continued)

Example:

\[ \overline{c} \triangleleft a \triangleright |c(x).\overline{d} \triangleleft x \triangleright \]

The first process sends a on channel c, the second one inputs this message, puts it in variable x and sends x on channel d.

The link with cryptographic protocols is clear:

- Each participant of the protocol is represented by a process
- The messages exchanged by processes are the messages of the protocol.

However, in protocols, messages are not necessarily atomic values. The names of the pi calculus are replaced by terms in the spi calculus.
Proverif

Pi calculus + cryptographic primitives

\[ M, N ::= \text{terms} \]
\[ x, y, z \quad \text{variable} \]
\[ a, b, c, k, \quad \text{name} \]
\[ f(M_1, \ldots, M_n) \quad \text{constructor application} \]

\[ P, Q ::= \text{processes} \]
\[ \overline{M} < N > . P \quad \text{output} \]
\[ M(x).P \quad \text{input} \]
\[ \text{let } x = g(M_1, \ldots, M_n) \text{ in } P \text{ else } Q \quad \text{destructor application} \]
\[ \text{if } M = N \text{ then } P \text{ else } Q \quad \text{conditional} \]
\[ 0 \quad \text{nil process} \]
\[ P | Q \quad \text{parallel composition} \]
\[ !P \quad \text{replication} \]
\[ (\nu a)P \quad \text{restriction} \]
Example: Denning Sacco

Message 1. \( A \rightarrow B : \{\{k\}_{sk_A}\}_{pk_B} \)
Message 2. \( B \rightarrow A\{s\}_k, k \text{ fresh} \)

\((\nu sk_A)(\nu sk_B) \text{ let } pk_A = pk(sk_A) \text{ in let } pk_B = pk(sk_B) \text{ in} \)
\( \overline{c} < pk_A > \overline{c} < pk_B > . \)

(A) \( !c(x_{-pk_B}).(\nu k)\overline{c} < pencrypt(sign(k, sk_A), x_{-pk_B}) > \)
\( .c(x).\text{lets} = sdecrypt(x, k) \text{ in 0} \)

(B) \( |!c(y).\text{lety'} = pdecrypt(y, sk_B) \text{ in let } k = checksign(y', pk_A) \text{ in} \)
\( \overline{c} < sencrypt(s, k) > \)
Proverif: Pi Calculus

```plaintext
free c.
(* Public key cryptography *)
fun pk/1.
private fun sk/1.
(* just encryption, no signing *)
fun encrypt/2.
reduc decrypt(encrypt(x,pk(y)),sk(y)) = x.

(* Symmetric key cryptography *)
fun symcrypt/2.
reduc symdecrypt(symcrypt(z,j),j) = z.

(* Effectively the claim signals *)
private free secretANa, secretANb, secretBNa, secretBNb, secretAtoB, secretBtoA.

(* Security claims to verify *)
query attacker:secretANa;
    attacker:secretANb;
    attacker:secretAtoB;
    attacker:secretBNa;
    attacker:secretBNb;
    attacker:secretBtoA.
```
Proverif: Pi Calculus

let processA =
  (* Choose the other host *)
  in(c, X);
new Na;
out(c, encrypt((Na,X),pk(X)));
in(c,m2);
let (=Na, nb) = decrypt(m2, sk(A)) in
out(c, encrypt(nb,pk(X)));
if X = A then
  out(c, symcrypt(secretANa, Na));
  out(c, symcrypt(secretANb, nb))
else if X = B then
  out(c, symcrypt(secretAtoB, Na));
  out(c, symcrypt(secretAtoB, nb));
  out(c, symcrypt(secretANa, Na));
  out(c, symcrypt(secretANb, nb)).
Proverif: Pi Calculus

let processB =

    in(c,m1);
    let (na,Y) = decrypt(m1, sk(B)) in
    new Nb;
    out(c, encrypt((na, Nb), pk(Y)));
    in(c,m3);
    let (=Nb) = decrypt(m3, sk(B)) in
    if Y = A then
        out(c, symcrypt(secretBtoA, na));
        out(c, symcrypt(secretBtoA, Nb));
        out(c, symcrypt(secretBNa, na));
        out(c, symcrypt(secretBNb, Nb))
    else
        if Y = B then
            out(c, symcrypt(secretBNa, na));
            out(c, symcrypt(secretB Nb, Nb)).
Proverif: Pi Calculus

let processBbyA =
  in(c,m1);
  let (na,Y) = decrypt(m1, sk(A)) in
  new Nb;
  out(c, encrypt((na, Nb), pk(Y)));
  in(c,m3);
  let (=Nb) = decrypt(m3, sk(A)) in
  if Y = A then
    out(c, symcrypt(secretBtoA, na));
    out(c, symcrypt(secretBtoA, Nb));
    out(c, symcrypt(secretBNa, na));
    out(c, symcrypt(secretBNb, Nb))
  else if Y = B then
    out(c, symcrypt(secretBNa, na));
    out(c, symcrypt(secretBNb, Nb)).
Proverif: Pi Calculus

process

    new A;
    new B;
    new I;

    out(c,A);
    out(c,B);
    out(c,I);
    out(c,sk(I));

    (!processA) | (!processB) | (!processBbyA)
RESULT not attacker:secretANa[] is true.
RESULT not attacker:secretANb[] is false.
RESULT not attacker:secretAtoB[] is true.
RESULT not attacker:secretBNa[] is false.
RESULT not attacker:secretBNb[] is false.
RESULT not attacker:secretBtoA[] is false.
Needham-Schroeder: secrecy of na and nb for A,B

- CL-Atse
- OFMC
- ProVerif
- Sat-MC
- Scyther
- TA4SP

Graph showing comparison of time (s) and number of runs for different tools.
Outline

Dolev Yao’s Intruder

Passive Intruder: Intruder Deduction Problem

Active Intruder: Security Problem

Tools

AVISPA
Scyther
Proverif
Comparaison

Bonus : Undecidability

Conclusion
Undecidability

Definition (Post Correspondence Problem (PCP))

Let $\Sigma$ be a finite alphabet.

**Input**: Sequence of pairs $\langle u_i, v_i \rangle_{1 \leq i \leq n}$, $u_i, v_i \in \Sigma^*$, $n \in \mathbb{N}$

**Question**: Existence of $k, i_1, \ldots, i_k \in \mathbb{N}$ such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$?
Undecidability

Definition (Post Correspondence Problem (PCP))

Let \( \Sigma \) be a finite alphabet.

**Input:** Sequence of pairs \( \langle u_i, v_i \rangle_{1 \leq i \leq n} \) \( u_i, v_i \in \Sigma^*, \ n \in \mathbb{N} \)

**Question:** Existence of \( k, i_1, \ldots, i_k \in \mathbb{N} \) such that \( u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k} \)?

**Example**

\[
\begin{array}{cccccc}
  u_1 & u_2 & u_3 & u_4 & v_1 & v_2 & v_3 & v_4 \\
  aba & bbb & aab & bb & a & aaa & abab & babba \\
\end{array}
\]

**Solution:** \( 1431 \)

\[
u_1 \cdot u_4 \cdot u_3 \cdot u_1 = aba \cdot bb \cdot aab \cdot aba = \alpha \cdot babba \cdot abab \cdot a = v_1 \cdot v_4 \cdot v_3 \cdot v_1
\]

But no solution for \( \langle u_1, v_1 \rangle, \langle u_2, v_2 \rangle, \langle u_3, v_3 \rangle \)
Undecidability

Definition (Post Correspondence Problem (PCP))

Let $\Sigma$ be a finite alphabet.

**Input**: Sequence of pairs $\langle u_i, v_i \rangle_{1 \leq i \leq n}$, $u_i, v_i \in \Sigma^*$, $n \in \mathbb{N}$

**Question**: Existence of $k, i_1, \ldots, i_k \in \mathbb{N}$ such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$?

**Example**

<table>
<thead>
<tr>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$u_3$</th>
<th>$u_4$</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$v_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>aba</td>
<td>bbb</td>
<td>aab</td>
<td>bb</td>
<td>a</td>
<td>aaa</td>
<td>abab</td>
<td>babba</td>
</tr>
</tbody>
</table>

Solution: **1431**

$u_1 \cdot u_4 \cdot u_3 \cdot u_1 = aba \cdot bb \cdot aab \cdot aba = a \cdot babba \cdot abab \cdot a = v_1 \cdot v_4 \cdot v_3 \cdot v_1$

But no solution for $\langle u_1, v_1 \rangle, \langle u_2, v_2 \rangle, \langle u_3, v_3 \rangle$

**PCP is undecidable**
Undecidability for Protocols

We construct a protocol such that decidability of secret implies decidability of PCP.

\[ A: \text{send}(\{\langle u_i, v_i \rangle \}_{K_{ab}}) \quad (1 \leq i \leq n) \]

\[ B: \text{receive}(\{\langle x, y \rangle \}_{K_{ab}}) \]
\[ \text{send}(\{\langle x \cdot u_i, y \cdot v_i \rangle \}_{K_{ab}}, \{s\} \{\langle x \cdot u_i, x \cdot u_i \rangle \}_{K_{ab}}) \quad (1 \leq i \leq n) \]

We assume that \( K_{AB} \) is a shared key between \( A \) and \( B \).

Intruder can find \( s \) iff he can solve PCP.
Outline

Dolev Yao’s Intruder
Passive Intruder: Intruder Deduction Problem
Active Intruder: Security Problem

Tools
   AVISPA
   Scyther
   Proverif
   Comparaison

Bonus: Undecidability

Conclusion
Summary

Today

- Hermes
- Scyther
- Avispa
- Proverif
Conclusion

- Automatic verification is necessary.
- Tool are very helpful for design and verification.
- Use your favorite tool.
- Modeling of a protocol is quite tricky.
- Know the limitations of the tool and what you are checking.
Next

- Others Protocols
- Others properties
- Others Tools: Maude, NPA, TA4SP, new OFMC
Thank you for your attention.

Questions?