How does ProVerif work?

Véronique Cortier
How to analyse security protocols?

Methodology

1. Proposing accurate models
   - symbolic models
   - cryptographic/computational models

2. Proving security
   - decidability/undecidability results
   - tools

confidentiality authenticity non-repudiation
Presence of an attacker

- may read every message sent on the net,
- may intercept and send new messages.

⇒ The system is infinitely branching
How to decide security for unlimited sessions?

→ In general, it is **undecidable**!
   (i.e. there exists no algorithm for checking e.g. secrecy)

How to prove undecidability?
How to decide security for unlimited sessions?

→ In general, it is **undecidable**!
  (i.e. there exists no algorithm for checking e.g. secrecy)

How to prove undecidability?

Post correspondence problem (PCP)

input \( \{(u_i, v_i)\}_{1 \leq i \leq n}, u_i, v_i \in \Sigma^* \)

output \( \exists n, i_1, \ldots, i_n \quad u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n} \)

Example : \( \{ (bab, b), (ab, aba), (a, baba) \} \)

Solution ?
How to decide security for unlimited sessions?

→ In general, it is **undecidable**!
  (i.e. there exists no algorithm for checking e.g. secrecy)

How to prove undecidability?

**Post correspondence problem (PCP)**

input \{ (u_i, v_i) \}_{1 \leq i \leq n}, u_i, v_i \in \Sigma^*

output \exists n, i_1, \ldots, i_n \quad u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n}

Example : \{(bab, b), (ab, aba), (a, baba)\}

Solution? → Yes, 1,2,3,1.

babababab
babababab
How to circumvent undecidability?

- Find **decidable subclasses** of protocols.
- Design **semi-decision procedure**, that works in practice
- ...
How to model an unbounded number of sessions?

“For any $x$, if the agent $A$ receives $\text{enc}(x, k_a)$ then $A$ responds with $x$.”

→ Use of first-order logic.
Horn clauses perfectly reflects the attacker symbolic manipulations on terms.

\[ \forall x \forall y \quad I(x), I(y) \Rightarrow I(\{x\}_y) \quad \text{encryption} \]

\[ \forall x \forall y \quad I(\{x\}_y), I(y) \Rightarrow I(x) \quad \text{decryption} \]

\[ \forall x \forall y \quad I(x), I(y) \Rightarrow I(<x, y>) \quad \text{concatenation} \]

\[ \forall x \forall y \quad I(<x, y>) \Rightarrow I(x) \quad \text{first projection} \]

\[ \forall x \forall y \quad I(<x, y>) \Rightarrow I(y) \quad \text{second projection} \]
Protocol as Horn clauses

Each action of the protocol is expressed by a logical implication.

\[
\forall x \quad I(x) \quad \Rightarrow \quad I(\{x\}_{k_b})
\]

\[
\forall x \quad I(\{x\}_{k_a}) \quad \Rightarrow \quad I(x)
\]
Security reduces to consistency

\[ \forall x \forall y \quad I(x), I(y) \Rightarrow I(<x, y>) \]
\[ \forall x \forall y \quad I(x), I(y) \Rightarrow I(\{x\}, y) \]
\[ \forall x \forall y \quad I(\{x\}, y), I(y) \Rightarrow I(x) \]
\[ \forall x \forall y \quad I(<x, y>), I(y) \Rightarrow I(x) \]
\[ \forall x \forall y \quad I(<x, y>), I(x) \Rightarrow I(y) \]

Does not yield a contradiction?

(i.e. consistent theory?)
Security reduces to consistency

Does not yield a contradiction? (i.e. consistent theory?)

$$\forall x \forall y \quad I(x), I(y) \Rightarrow I(<x, y>)$$

$$\forall x \forall y \quad I(x), I(y) \Rightarrow I(\{x\}, y)$$

$$\forall x \forall y \quad I(\{x\}, y), I(y) \Rightarrow I(x)$$

$$\forall x \forall y \quad I(<x, y>), I(y) \Rightarrow I(x)$$

$$\forall x \forall y \quad I(<x, y>), I(y) \Rightarrow I(y)$$

$$\forall x \quad I(x) \Rightarrow I(\{x\}_{ka})$$

$$\forall x \quad I(\{x\}_{ka}) \Rightarrow I(x)$$

$$\forall x \quad I(\{\text{secret}\}_{ka})$$
How to know if a set of formula is consistent?

Hilbert’s program (1928)
“Entscheidung Problem”

David Hilbert
How to know if a set of formula is consistent?

Hilbert’s program (1928)  
“Entscheidung Problem”

It is undecidable! (1936)  
→ There is no algorithm that answers this question.

(at a time with no computers)
Back to our business

Does not yield a contradiction? (i.e. consistent theory?)

All this for nothing?

\[
\forall x \forall y \quad I(x), I(y) \implies I(<x, y>)
\]
\[
\forall x \forall y \quad I(x), I(y) \implies I(\{x\}y)
\]
\[
\forall x \forall y \quad I(\{x\}y), I(y) \implies I(x)
\]
\[
\forall x \forall y \quad I(<x, y>) \implies I(y)
\]
\[
\forall x \forall y \quad I(<x, y>) \implies I(x)
\]
\[
\forall x \quad I(x) \implies I(\{x\}ka)
\]
\[
\forall x \quad I(\{x\}ka) \implies I(x)
\]
\[
\neg (\text{secret})
\]
A standard technique: resolution

Idea: add logical consequences...

\[ \forall x P(x) \Rightarrow I(s(x)) \]
\[ \forall x I(x) \Rightarrow P(s(x)) \]
\[ P(0) \]
\[ \neg I(s(s(s(0)))) \]

...until a contradiction is found.

We need a method (a strategy) which is:

- correct: adds formula that are indeed consequences
- complete: finds a contradiction (if it exists)
- in a finite number of steps (decidable fragment)
A standard technique: resolution

Idea: add logical consequences...

\[
\forall x P(x) \Rightarrow I(s(x)) \\
\forall x I(x) \Rightarrow P(s(x)) \\
P(0) \\
\neg I(s(s(s(0)))) \\
\]

... until a contradiction is found.

We need a method (a strategy) which is:

- **correct**: adds formula that are indeed consequences
- **complete**: finds a contradiction (if it exists)
- **in a finite number of steps**: (decidable fragment)
A standard technique: resolution

Idea: add logical consequences...

\[ \forall x P(x) \Rightarrow I(s(x)) \]
\[ \forall x I(x) \Rightarrow P(s(x)) \]
\[ P(0) \]
\[ \neg I(s(s(0))) \]

\[ \forall x P(x) \Rightarrow I(s(x)) \]
\[ \forall x I(x) \Rightarrow P(s(x)) \]
\[ P(0) \]
\[ \neg I(s(s(0))) \]

... until a contradiction is found.

We need a method (a strategy) which is:

- **correct**: adds formula that are indeed consequences
- **complete**: finds a contradiction (if it exists)
- **in a finite number of steps** (decidable fragment)
A standard technique: resolution

Idea: add logical consequences...

∀x\(P(x) \Rightarrow l(s(x))\)
∀x\(l(x) \Rightarrow P(s(x))\)
\(P(0)\)
\(!I(s(s(0))))\)

\(\forall x P(x) \Rightarrow l(s(x))\)
\(\forall x l(x) \Rightarrow P(s(x))\)
\(P(0)\)
\(!I(s(s(0))))\)

... until a contradiction is found.

We need a method (a strategy) which is:

- **correct**: adds formula that are indeed consequences
- **complete**: finds a contradiction (if it exists)
- **in a finite number of steps**: (decidable fragment)
Binary resolution

$A, B$ are atoms and $C, D$ are clauses.

An intuitive rule

$$
\begin{array}{c}
A \Rightarrow C
\
A
\
C
\end{array}
$$

In other words

$$
\begin{array}{c}
\neg A \vee C
\
A
\
C
\end{array}
$$

Generalizing

$$
\neg A \vee C \quad A
\
C
\theta \quad \theta
\quad \text{mgu}(A, B)
$$

Generalizing a bit more

$$
\neg A \vee C \quad A
\
D \quad \theta
\quad \text{mgu}(A, B)
$$
Binary resolution

\( A, B \) are atoms and \( C, D \) are clauses.

An intuitive rule

\[
\frac{A \Rightarrow C}{C}
\]

In other words

\[
\frac{\neg A \lor C}{C}
\]

Generalizing

\[
\frac{\neg A \lor C}{C\theta} \quad B \quad \theta = mgu(A, B) \quad (i.e. \ A\theta = B\theta)
\]
$A, B$ are atoms and $C, D$ are clauses.

An intuitive rule

\[
A \Rightarrow C \quad A \\
\hline
\quad C
\]

In other words

\[
\neg A \lor C \quad A \\
\hline
\quad C
\]

Generalizing

\[
\neg A \lor C \quad B \\
\hline
C \theta
\]

$\theta = \text{mgu}(A, B)$ \ (i.e. $A\theta = B\theta$)

Generalizing a bit more

\[
\neg A \lor C \quad B \lor D \\
\hline
C \theta \lor D \theta
\]

$\theta = \text{mgu}(A, B)$ \quad Binary\ resolution
Binary resolution and Factorization

\[ \neg A \lor C \quad B \lor D \]
\[ \frac{}{C\theta \lor D\theta} \quad \theta = \text{mgu}(A, B) \]

Binary resolution

\[ A \lor B \lor C \]
\[ \frac{}{A\theta \lor C\theta} \quad \theta = \text{mgu}(A, B) \]

Factorisation

Theorem (Soundness and Completeness)

Binary resolution and factorisation are sound and refutationally complete, i.e. a set of clauses \( C \) is not satisfiable if and only if \( \bot \) (the empty clause) can be obtained from \( C \) by binary resolution and factorisation.

Exercise : Why do we need the factorisation rule?
Example

\[ C = \{ \neg I(s), \quad I(k_1), \quad I(\{s\} \langle k_1, k_1 \rangle), \]

\[ I(\{x\}_y), I(y) \Rightarrow I(x), \quad I(x), I(y) \Rightarrow I(\langle x, y \rangle) \]

\[ I(\{s\} \langle k_1, k_1 \rangle) \quad I(\{x\}_y), I(y) \Rightarrow I(x) \]

\[ \frac{I(\langle k_1, k_1 \rangle) \Rightarrow I(s)}{I(s)} \]

\[ \neg I(s) \quad I(s) \]

\[ I(\langle k_1, k_1 \rangle) \]

\[ I(k_1) \quad I(x), I(y) \Rightarrow I(\langle x, y \rangle) \]

\[ I(\langle k_1, k_1 \rangle) \]

\[ I(k_1) \quad I(y) \Rightarrow I(\langle k_1, y \rangle) \]
But it is not terminating!

\[
\begin{align*}
I(s) & \quad I(x), I(y) \Rightarrow I(\langle x, y \rangle) \\
I(s) & \quad I(y) \Rightarrow I(\langle s, y \rangle) \\
I(y) & \Rightarrow I(\langle s, y \rangle) \\
I(\langle s, s \rangle) & \quad I(\langle s, \langle s, s \rangle \rangle) \\
I(\langle s, \langle s, \langle s, s \rangle \rangle \rangle) & \\
\ldots & \\
\rightarrow & \text{This does not yield any decidability result.}
\end{align*}
\]
Ordered Binary resolution and Factorization

Let \(<\) be any order on clauses.

\[
\begin{array}{c}
\neg A \lor C & B \lor D \\
\hline
C\theta \lor D\theta
\end{array}
\]

\[\theta = \mgu(A, B)\]

\[A\theta \not< C\theta \lor D\theta\]

Ordered binary resolution

\[
\begin{array}{c}
A \lor B \lor C \\
\hline
A\theta \lor C\theta
\end{array}
\]

\[\theta = \mgu(A, B)\]

\[A\theta \not< C\theta\]

Ordered factorisation

Theorem (Soundness and Completeness)

Ordered binary resolution and factorisation are sound and refutationally complete provided that \(<\) is liftable.
Ordered Binary resolution and Factorization

Let \(<\) be any order on clauses.

\[
\neg A \lor C \quad B \lor D \quad \theta = \text{mgu}(A, B) \quad \text{Ordered binary resolution}
\]
\[
C \theta \lor D \theta \\
A \theta \lor C \theta
\]
\[
A \theta \not< C \theta
\]

\[
A \lor B \lor C \quad \theta = \text{mgu}(A, B) \quad \text{Ordered factorisation}
\]
\[
A \theta \lor C \theta
\]
\[
A \theta \not< C \theta
\]

Theorem (Soundness and Completeness)

*Ordered binary resolution and factorisation are sound and refutationally complete provided that \(<\) is liftable*

\[
\forall A, B, \theta \quad A < B \Rightarrow A \theta < B \theta
\]
Examples of liftable orders

\[ \forall A, B, \theta \quad A < B \Rightarrow A\theta < B\theta \]

First example : subterm order

\[ P(t_1, \ldots, t_n) < Q(u_1, \ldots, u_k) \quad \text{iff any } t_i \text{ is a subterm of } u_1, \ldots, u_k \]

→ extended to clauses as follows : \( C_1 < C_2 \) iff any literal of \( C_1 \) is smaller than some literal of \( C_2 \).

Exercise : Show that \( C \) is not satisfiable by ordered resolution (and factorisation).
Examples of liftable orders - continued

Second example: \( P(t_1, \ldots, t_n) \preceq Q(u_1, \ldots, u_k) \) iff

1. \( \text{depth}(P(t_1, \ldots, t_n)) \leq \text{depth}(Q(u_1, \ldots, u_k)) \)
2. For any variable \( x \),
   \[ \text{depth}_x(P(t_1, \ldots, t_n)) \leq \text{depth}_x(Q(u_1, \ldots, u_k)) \]

Exercise: Show that \( \forall A, B, \theta \ A \preceq B \Rightarrow A\theta \preceq B\theta \)
Examples of liftable orders - continued

Second example: $P(t_1, \ldots, t_n) \preceq Q(u_1, \ldots, u_k)$ iff

1. $\text{depth}(P(t_1, \ldots, t_n)) \leq \text{depth}(Q(u_1, \ldots, u_k))$

2. For any variable $x$,
   $\text{depth}_x(P(t_1, \ldots, t_n)) \leq \text{depth}_x(Q(u_1, \ldots, u_k))$

Exercise: Show that $\forall A, B, \theta \hspace{1pt} A \preceq B \Rightarrow A \theta \preceq B \theta$
Examples of liftable orders - continued

Second example: \( P(t_1, \ldots, t_n) \preceq Q(u_1, \ldots, u_k) \) iff

1. \( \text{depth}(P(t_1, \ldots, t_n)) \leq \text{depth}(Q(u_1, \ldots, u_k)) \)
2. For any variable \( x \), \( \text{depth}_x(P(t_1, \ldots, t_n)) \leq \text{depth}_x(Q(u_1, \ldots, u_k)) \)

Exercise: Show that \( \forall A, B, \theta \quad A \preceq B \Rightarrow A\theta \preceq B\theta \)
Back to protocols

Intruder clauses are of the form

\[ \pm l(f(x_1, \ldots, x_n)), \pm l(x_i), \pm l(x_j) \]

Protocol clauses

\[ I(\{\text{pin}\}_{k_a}) \]
\[ I(x) \Rightarrow I(\{x\}_{k_b}) \]
\[ I(\{x\}_{k_a}) \Rightarrow I(x) \]

At most one variable per clause!
Intruder clauses are of the form

$$\pm I(f(x_1, \ldots, x_n)), \ \pm I(x_i), \ \pm I(x_j)$$

Protocol clauses

$$\Rightarrow \quad I(\{\text{pin}\}_{k_a})$$

$$I(x) \Rightarrow I(\{x\}_{k_b})$$

$$I(\{x\}_{k_a}) \Rightarrow I(x)$$

At most one variable per clause!

Theorem

Given a set $C$ of clauses such that each clause of $C$

- either contains at most one variable
- or is of the form $\pm I(f(x_1, \ldots, x_n)), \ \pm I(x_i), \ \pm I(x_j)$

Then ordered ($\prec$) binary resolution and factorisation is terminating.
Decidability for an unbounded number of sessions

Corollary

*For any protocol that can be encoded with clauses of the previous form, then checking secrecy is decidable.*

But how to deal with protocols that need more than one variable per clause?
Developed by Bruno Blanchet, Paris, France.

- No restriction on the clauses
- Implements a sound semi-decision procedure (that may not terminate).
- Based on a resolution strategy well adapted to protocols.
- Performs very well in practice!
  - Works on most of existing protocols in the literature
  - Is also used on industrial protocols (e.g. certified email protocol, JFK, Plutus filesystem)
Definition

A selection function is any function \( \text{sel} \) such that \( \text{sel}(H \Rightarrow C) \subseteq H \).

\[
\begin{align*}
\neg A \lor C & \quad B \lor D \\
\frac{}{C \theta \lor D \theta} \\
\theta = \text{mgu}(A, B) & \\
A \in \text{sel}(\neg A \lor C) \text{ or } \text{sel}(\neg A \lor C) = \emptyset \\
\text{sel}(B \lor D) = \emptyset
\end{align*}
\]
**Definition**

A *selection function* is any function $sel$ such that $sel(H \Rightarrow C) \subseteq H$.

\[
\begin{array}{cc}
\neg A \lor C & B \lor D \\
\hline
C\theta \lor D\theta & \theta = \text{mgu}(A, B)
\end{array}
\]

$A \in sel(\neg A \lor C)$ or $sel(\neg A \lor C) = \emptyset$

$sel(B \lor D) = \emptyset$

**Theorem**

*Resolution and factorisation with selection are sound and refutationally complete* for any selection function.
Limitations of ProVerif

What is the gap between processes and Horn clauses?
Limitations of ProVerif

What is the gap between processes and Horn clauses?

- The order of actions is abstracted
- Impossible to specify “just once”
  \[ P \text{ and } !P \text{ have the same translation in Horn clauses.} \]
- Nonces are abstracted by function of the inputs.
  Example:
  \[
  \text{in}(x).\text{new}
  n.\text{out}(\text{enc}((x, n); k))
  \]
  is intuitively translated into the clause
  \[
  l(x) \Rightarrow l(\text{enc}((x, n(x)); k))
  \]
What formal methods allow to do?

- In general, secrecy preservation is **undecidable**.
What formal methods allow to do?

- In general, secrecy preservation is **undecidable**.

- For a **bounded number of sessions**, secrecy is co-NP-complete \[\text{[RusinowitchTuruani CSFW01]}\] → **several tools for detecting attacks** (Casper, Avispa, Scyther, ... )

- For an unbounded number of sessions for one-copy protocols, secrecy is DEXPTIME-complete \[\text{[CortierComon RTA03]} \text{ [SeildVerma LPAR04]}\] for message-length bounded protocols, secrecy is DEXPTIME-complete \[\text{[Durgin et al FMSP99]} \text{ [Chevalier et al CSL03]}\] → **some tools for proving security** (ProVerif, Scyther)
What formal methods allow to do?

- In general, secrecy preservation is undecidable.

- For a **bounded number of sessions**, secrecy is co-NP-complete [RusinowitchTuruani CSFW01]
  → **several tools for detecting attacks** (Casper, Avispa, Scyther, ...)

- For an **unbounded number of sessions**
  - for one-copy protocols, secrecy is DEXPTIME-complete [CortierComon RTA03] [SeildVerma LPAR04]
  
  - for message-length bounded protocols, secrecy is DEXPTIME-complete [Durgin et al FMSP99] [Chevalier et al CSL03]
  → **some tools for proving security** (ProVerif, Scyther)