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
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?

**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 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 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.

\[
\begin{align*}
babababab \\
babababab
\end{align*}
\]
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.

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

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

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

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

\[ \forall x \forall y \quad I(<x, y>), I(y) \Rightarrow I(y) \quad \text{second projection} \]
Each action of the protocol is expressed by a logical implication.

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

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

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

∀x ∀y \ I(x), I(y) ⇒ I(<x, y>)
∀x ∀y \ I(x), I(y) ⇒ I({x}y)
∀x ∀y \ I({x}y), I(y) ⇒ I(x)
∀x ∀y \ I(<x, y>), I(y) ⇒ I(x)
∀x ∀y \ I(<x, y>), I(y) ⇒ I(x)
∀x ∀y \ I(<x, y>) ⇒ I(y)

∀x \ I(x) ⇒ I({x}k_a)
∀x \ I({x}_ka) ⇒ I(x)
Security reduces to consistency

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

\( \neg I(\text{secret}) \)

Does not yield a contradiction?

(i.e. consistent theory?)
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?

∀x∀y I(x), I(y) ⇒ I(<x, y>)
∀x∀y I(x), I(y) ⇒ I({x}, y)
∀x∀y I({x}, y), I(y) ⇒ I(x)
∀x∀y I(<x, y>) ⇒ I(x)
∀x∀y I(<x, y>) ⇒ I(y)
∀x∀y I(<x, y>) ⇒ I({secret}_{ka})
∀x I(x) ⇒ I({x}_{kb})
∀x I({x}_{ka}) ⇒ I(x)
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...

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

... 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...

\[
\begin{align*}
\forall x P(x) & \Rightarrow I(s(x)) \\
\forall x I(x) & \Rightarrow P(s(x)) \\
P(0) & \\
\neg I(s(s(s(0))))
\end{align*}
\]

... 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...

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

... 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

\[ A \Rightarrow C \quad A \]

\[ C \]

In other words

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

\[ C \]
Binary resolution

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

An intuitive rule

\[
\begin{array}{c}
\frac{A \Rightarrow C}{\bar{A} \lor C}
\end{array}
\]

In other words

\[
\begin{array}{c}
\frac{\neg A \lor C}{A \lor C}
\end{array}
\]

Generalizing

\[
\begin{array}{c}
\frac{\neg A \lor C}{A \theta = \text{mgu}(A, B)}
\end{array}
\]

(i.e. \( A\theta = B\theta \))
Binary resolution

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

An intuitive rule

\[
A 
\implies C
\]

\[
A
\]

\[
C
\]

In other words

\[
\neg A \lor C
\]

\[
A
\]

\[
C
\]

Generalizing

\[
\neg A \lor C
\]

\[
B
\]

\[
\theta = \text{mgu}(A, B)
\]

(i.e. $A\theta = B\theta$)

Generalizing a bit more

\[
\neg A \lor C
\]

\[
B \lor D
\]

\[
C\theta \lor D\theta
\]

\[
\theta = \text{mgu}(A, B)
\]

Binary resolution
Binary resolution and Factorization

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

\[\text{Binary resolution}\]

\[
\begin{align*}
A \lor B \lor C & \\
\hline
A\theta \lor C\theta & \quad \theta = \text{mgu}(A, B) \\
\end{align*}
\]

\[\text{Factorisation}\]

**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

\[ C = \{-\neg I(s), \quad I(k_1), \quad I(\{s\}_{k_1,k_1}), \]
\[ I(\{x\}_y), I(y) \Rightarrow I(x), \quad I(x), I(y) \Rightarrow I(\langle x, y \rangle) \]

\[ \text{Proof:} \]
\[ I(\{s\}_{k_1,k_1}) \quad I(\{x\}_y), I(y) \Rightarrow I(x) \]
\[ \quad \hline \]
\[ I(\langle k_1, k_1 \rangle) \Rightarrow s \]
\[ \text{\hline} \quad I(k_1) \quad I(x), I(y) \Rightarrow I(\langle x, y \rangle) \]
\[ \quad \hline \quad I(k_1) \quad I(y) \Rightarrow I(\langle k_1, y \rangle) \]
\[ \quad \hline \quad I(\langle k_1, k_1 \rangle) \]
\[ \neg I(s) \quad I(s) \]
\[ \quad \hline \quad \bot \]
But it is not terminating!

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

Let $\prec$ be any order on clauses.

\[
\frac{\neg A \lor C \quad B \lor D}{C \theta \lor D \theta} \quad \theta = \text{mgu}(A, B) \quad A \theta \not\prec C \theta \lor D \theta
\]

Ordered binary resolution

\[
\frac{A \lor B \lor C}{A \theta \lor C \theta} \quad \theta = \text{mgu}(A, B) \quad A \theta \not\prec C \theta
\]

Ordered factorisation

Theorem (Soundness and Completeness)

Ordered binary resolution and factorisation are sound and refutationally complete provided that $\prec$ is liftable:

\[
\forall A, B, \theta \quad A \prec B \Rightarrow A \theta \prec B \theta
\]
Ordered Binary resolution and Factorization

Let $<$ be any order on clauses.

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

Ordered binary resolution

\[
A \lor B \lor C \\
\hline
A\theta \lor C\theta \quad A\theta \not< C\theta
\]

Ordered factorisation

Theorem (Soundness and Completeness)

Ordered binary resolution and factorisation are sound and refutationally complete provided that $<$ is liftable

\[
\forall A, B, \theta \quad A < B \Rightarrow A\theta < B\theta
\]
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: $P(t_1, \ldots, t_n) \lesssim Q(u_1, \ldots, u_k)$ iff

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

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

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

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

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

\[
\begin{align*}
f \quad & \quad f \\
x \quad & \quad f \\
x \quad & \quad f \\
y \quad & \quad a \\
x \quad & \quad h \\
& \quad h \\
& \quad h \\
& \quad y
\end{align*}
\]
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 \( \forall A, B, \theta \quad A \preceq B \Rightarrow A\theta \preceq B\theta \)
Intruder clauses are of the form

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

Protocol clauses

\[ I(\{\text{pin}\}_{ka}) \]  
\[ I(x) \Rightarrow I(\{x\}_{kb}) \]
\[ I(\{x\}_{ka}) \Rightarrow I(x) \]

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

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

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 (\( \lesssim \)) 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)
Resolution strategy with selection

Definition

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

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

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

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

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

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

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:**
  
  \[
in(x).\text{new } n.\text{out}(\text{enc}((x, n); k))
  \]
  
  is intuitively translated into the clause
  
  \[
  I(x) \Rightarrow I(\text{enc}((x, n(x)); k))
  \]
In general, secrecy preservation is **undecidable**.
In general, secrecy preservation is undecidable.

For a bounded number of sessions, secrecy is co-NP-complete [RusinowitchTurunani CSFW01] → several tools for detecting attacks (Casper, Avispa, Scyther, ... )
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
    
    \[ \text{[CortierComon RTA03]} \quad \text{[SeildVerma LPAR04]} \]

  - for **message-length bounded protocols**, secrecy is
    
    DEXPTIME-complete \[ \text{[Durgin et al FMSP99]} \quad \text{[Chevalier et al CSL03]} \]

  → **some tools for proving security** (ProVerif, Scyther)