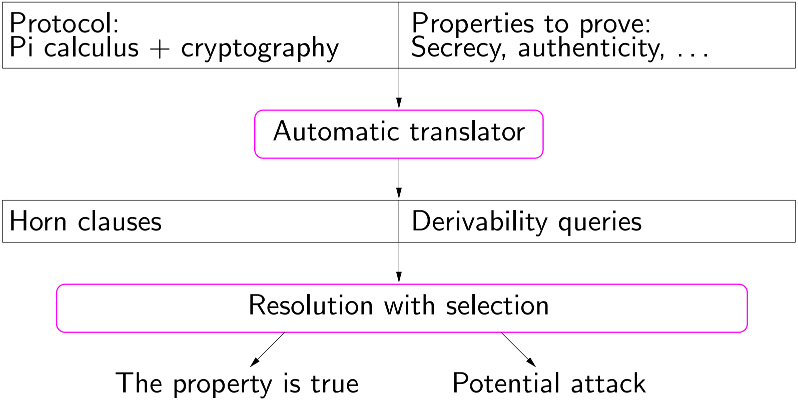
Automatic Verification of Cryptographic Protocols in the Formal Model
Automatic Verifier ProVerif

Bruno Blanchet

CNRS, École Normale Supérieure, INRIA
blanchet@di.ens.fr

March 2009

Overview of the protocol verifier ProVerif



Protocol:
Pi calculus + cryptography

Properties to prove:
Secrecy, authenticity, . . .

Automatic translator

Horn clauses

Derivability queries

Resolution with selection

The property is true

Potential attack

Overview

1. A variant of the spi-calculus
2. Intuitive presentation of the Horn clause representation
3. The solving algorithm
4. Experimental results
5. Formal translation from the spi-calculus.

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 $a, b, c, \dots$.

What is the spi calculus ? (continued)

Example: $\overline{c}\langle a\rangle \mid c(x).\overline{d}\langle x\rangle$
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.

Syntax of the process calculus

Pi calculus + cryptographic primitives

$M, N ::=$	terms
x, y, z	variable
a, b, c, k, s	name
$f(M_1, \dots, M_n)$	constructor application
$P, Q ::=$	processes
$\overline{M}\langle N\rangle.P$	output
$M(x).P$	input
$\mathit{let}\ x = g(M_1, \dots, M_n)\ \mathit{in}\ P\ \mathit{else}\ Q$	destructor application
$\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q$	conditional
0	nil process
$P \mid Q$	parallel composition
$!P$	replication
$(\nu a)P$	restriction

Constructors and destructors

Two kinds of operations:

- Constructors f are used to build terms $f(M_1, \ldots, M_n)$
- Destructors g manipulate terms
  $let\ x = g(M_1, \ldots, M_n)\ in\ P\ else\ Q$
  Destructors are defined by rewrite rules $g(M_1, \ldots, M_n) \rightarrow M$.

Examples of constructors and destructors

Shared-key encryption: $\{M\}_N$; one decrypts with the key N

- Constructor: Shared-key encryption $\mathrm{sencrypt}(M, N)$.
- Destructor: Decryption $\mathrm{sdecrypt}(M', N)$

$$\mathrm{sdecrypt}(\mathrm{sencrypt}(M, N), N) \rightarrow M.$$

Perfect encryption assumption: one can decrypt only if one has the key.

Examples of constructors and destructors

Public-key encryption: $\{M\}_{pk}$; one decrypts with the secret key sk

- Constructors: Public-key encryption $\mathrm{pencrypt}(M, N)$.
  Public key generation $\mathrm{pk}(N)$.
- Destructor: Decryption $\mathrm{pdecrypt}(M', N)$

$$\mathrm{pdecrypt}(\mathrm{pencrypt}(M, \mathrm{pk}(N)), N) \rightarrow M.$$

Examples of constructors and destructors (continued)

Signature: $\{M\}_{sk}$; one verifies with the public key pk

- Constructor: Signature $\mathrm{sign}(M, N)$.
- Destructors: Signature checking $\mathrm{checksign}(M', N')$

$$\mathrm{checksign}(\mathrm{sign}(M, N), \mathrm{pk}(N)) \rightarrow M.$$

Message extraction $\mathrm{getmess}(M')$

$$\mathrm{getmess}(\mathrm{sign}(M, N)) \rightarrow M.$$

Here, we assume that the signed message $\mathrm{sign}(M, N)$ contains the message M in the clear.

Exercise

Model signatures that do not reveal the signed message.

Examples of constructors and destructors (continued)

One-way hash function:

- Constructor: One-way hash function $H(M)$.

Very idealized model of a hash function (essentially corresponds to the random oracle model).

Examples of constructors and destructors (continued)

Tuples:

- Constructor: tuple $(M_1, \ldots, M_n)$.
- Destructors: projections $i\text{th}(M)$

$$i\text{th}((M_1, \ldots, M_n)) \rightarrow M_i$$

Tuples are used to represent all kinds of data structures in protocols.

Example: The Denning-Sacco protocol

$$\begin{array}{llll} \text{Message 1.} & A \rightarrow B : & \{\{k\}_{sk_A}\}_{pk_B} & k \text{ fresh} \\ \text{Message 2.} & B \rightarrow A : & \{s\}_k & \end{array}$$

$(\nu sk_A)(\nu sk_B) let\ pk_A = \mathrm{pk}(sk_A)\ in\ let\ pk_B = \mathrm{pk}(sk_B)\ in$

$\overline{c}\langle pk_A \rangle \overline{c}\langle pk_B \rangle.$

(A) $\quad ! \ c(x\_pk_B).(\nu k)\overline{c}\langle \mathrm{pencrypt}(\mathrm{sign}(k, sk_A), x\_pk_B)\rangle.$

$\quad c(x).let\ s = \mathrm{sdecrypt}(x, k)\ in\ 0$

(B) $\quad | \quad ! \ c(y).let\ y' = \mathrm{pdecrypt}(y, sk_B)\ in$

$\quad let\ k = \mathrm{checksign}(y', pk_A)\ in\ \overline{c}\langle \mathrm{sencrypt}(\mathrm{s}, k)\rangle$

Exercise: The Needham-Schroeder public-key protocol

Exercise

Model the following protocol:

Message 1.	$A \rightarrow B$	$\{N_a, A\}_{pk_B}$	N_a fresh
Message 2.	$B \rightarrow A$	$\{N_a, N_b\}_{pk_A}$	N_b fresh
Message 3.	$A \rightarrow B$	$\{N_b\}_{pk_B}$	

Formal semantics

The semantics is defined by reduction $P \rightarrow P'$: the execution of the process is modeled by transforming it into another process.

Main reduction rule = communication

$$\overline{N}\langle M\rangle.Q \mid N(x).P \ \rightarrow\ Q \mid P\{M/x\}$$

The communicating processes are not always in the above form, so we need an equivalence relation to prepare the reduction.

Equivalence relation

$$
\begin{array}{ll}
P \mid 0 \equiv P & P \equiv Q \;\Rightarrow\; P \mid R \equiv Q \mid R \\
P \mid Q \equiv Q \mid P & P \equiv Q \;\Rightarrow\; (\nu a)P \equiv (\nu a)Q \\
(P \mid Q) \mid R \equiv P \mid (Q \mid R) & P \equiv P \\
(\nu a_1)(\nu a_2)P \equiv (\nu a_2)(\nu a_1)P & Q \equiv P \;\Rightarrow\; P \equiv Q \\
(\nu a)(P \mid Q) \equiv P \mid (\nu a)Q \;\text{ if } a \notin \mathrm{fn}(P) & P \equiv Q,\; Q \equiv R \;\Rightarrow\; P \equiv R
\end{array}
$$

Reduction relation

$$\overline{N}\langle M\rangle.Q \mid N(x).P \;\rightarrow\; Q \mid P\{M/x\} \qquad \text{(Red I/O)}$$

$$\begin{array}{ll}
\mathit{let}\ x = g(M_1,\ldots,M_n)\ \mathit{in}\ P\ \mathit{else}\ Q \rightarrow P\{M'/x\} & \\
\quad \text{if } g(M_1,\ldots,M_n) \rightarrow M' & \text{(Red Destr 1)} \\
\mathit{let}\ x = g(M_1,\ldots,M_n)\ \mathit{in}\ P\ \mathit{else}\ Q \rightarrow Q & \\
\quad \text{if there exists no } M' \text{ such that } g(M_1,\ldots,M_n) \rightarrow M' & \text{(Red Destr 2)}
\end{array}$$

$$!P \;\rightarrow\; P \mid !P \qquad \text{(Red Repl)}$$

$$\begin{array}{ll}
P \;\rightarrow\; Q \;\Rightarrow\; P \mid R \;\rightarrow\; Q \mid R & \text{(Red Par)} \\
P \;\rightarrow\; Q \;\Rightarrow\; (\nu a)P \;\rightarrow\; (\nu a)Q & \text{(Red Res)}
\end{array}$$

$$P' \equiv P,\ P \;\rightarrow\; Q,\ Q \equiv Q' \;\Rightarrow\; P' \;\rightarrow\; Q' \qquad \text{(Red $\equiv$)}$$

Another presentation of the semantics

Semantic configurations are $\mathcal{E}, \mathcal{P}$ where

- $\mathcal{E}$ is a set of names
- $\mathcal{P}$ is a multiset of processes

Intuitively, $\mathcal{E}, \mathcal{P}$ where $\mathcal{E} = \{a_1, \dots, a_n\}$ and $\mathcal{P} = \{P_1, \dots, P_m\}$ corresponds to

$$(\nu a_1) \dots (\nu a_n)(P_1 \mid \dots \mid P_m)$$

Initial configuration for P: $\mathrm{fn}(P), \{P\}$.

Another presentation of the semantics: reduction relation

$$\mathcal{E}, \mathcal{P} \cup \{\, 0 \,\} \rightarrow \mathcal{E}, \mathcal{P} \qquad \text{(Red Nil)}$$

$$\mathcal{E}, \mathcal{P} \cup \{\, !P \,\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\, P, !P \,\} \qquad \text{(Red Repl)}$$

$$\mathcal{E}, \mathcal{P} \cup \{\, P \mid Q \,\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\, P, Q \,\} \qquad \text{(Red Par)}$$

$$\mathcal{E}, \mathcal{P} \cup \{\, (\nu a)P \,\} \rightarrow \mathcal{E} \cup \{a'\}, \mathcal{P} \cup \{\, P\{a'/a\} \,\} \qquad \text{(Red Res)}$$

where $a' \notin \mathcal{E}$.

$$\mathcal{E}, \mathcal{P} \cup \{\, \overline{N}\langle M\rangle.Q, N(x).P \,\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\, Q, P\{M/x\} \,\} \qquad \text{(Red I/O)}$$

$$\mathcal{E}, \mathcal{P} \cup \{\, \mathit{let}\ x = g(M_1, \dots, M_n)\ \mathit{in}\ P\ \mathit{else}\ Q \,\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\, P\{M'/x\} \,\}$$

if $g(M_1, \dots, M_n) \rightarrow M'$ (Red Destr 1)

$$\mathcal{E}, \mathcal{P} \cup \{\, \mathit{let}\ x = g(M_1, \dots, M_n)\ \mathit{in}\ P\ \mathit{else}\ Q \,\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\, Q \,\}$$

if there exists no M' such that $g(M_1, \dots, M_n) \rightarrow M'$ (Red Destr 2)

Comparison between the two semantics

The first semantics

- is more standard (comes from the original semantics of the pi calculus)
- makes it easier to add a context around an existing process (see definition of process equivalence)

The second semantics

- directs the reduction more precisely
- makes a minimal use of renaming (for restrictions only)

Except when mentioned explicitly, I will rely on the second semantics.

Adversary

The protocol is executed in parallel with an adversary.

The adversary can be any process.

S = finite set of names (initial knowledge of the adversary).

Definition

The closed process Q is an S-adversary $\Leftrightarrow \mathrm{fn}(Q) \subseteq S$.

Secrecy

Intuitive definition

The secret M cannot be output on a public channel

Definition

A trace $\mathcal{T} = \mathcal{E}_0, \mathcal{P}_0 \rightarrow^* \mathcal{E}', \mathcal{P}'$ outputs M if and only if $\mathcal{T}$ contains a reduction $\mathcal{E}, \mathcal{P} \cup \{\, \overline{c}\langle M \rangle.Q, c(x).P \,\} \rightarrow \mathcal{E}, \mathcal{P} \cup \{\, Q, P\{M/x\} \,\}$ for some $\mathcal{E}$, $\mathcal{P}$, x, P, Q, and $c \in S$.

Definition

The closed process P preserves the secrecy of M from $S \Leftrightarrow$ $\forall S$-adversary Q, $\forall \mathcal{T} = \mathrm{fn}(P) \cup S, \{P, Q\} \rightarrow^* \mathcal{E}', \mathcal{P}'$, $\mathcal{T}$ does not output M.

Several variants of the spi calculus

- Presented variant [Abadi, Blanchet, POPL'02 and JACM'05]
- The spi-calculus [Abadi, Gordon, I&C, 1999]
- The applied pi calculus [Abadi, Fournet, POPL'01]
  Very powerful, thanks to equational theories
- A calculus for asymmetric communication
  [Abadi, Blanchet, FoSSaCS'01 and TCS'03]

Overview

1. A variant of the spi-calculus
2. Intuitive presentation of the Horn clause representation
3. The solving algorithm
4. Experimental results
5. Formal translation from the spi-calculus.

Our goal

Goal: a verifier for cryptographic protocols

- Fully automatic
- For an unbounded number of sessions
  and an unbounded message size
- Handles many cryptographic primitives
- Proves various properties: secrecy, correspondences, equivalences
- Efficient

Our solution

Two ideas (extending [Weidenbach, CADE'99]):

- a simple abstract representation of these protocols, by a set of Horn clauses;
- an efficient solving algorithm to find which facts can be derived from these clauses.

Using this, we can prove secrecy properties of protocols,
or exhibit attacks showing why a message is not secret.

We handle in particular shared- and public-key cryptography, hash functions, Diffie-Hellman key agreements.

Protocol representation

- Messages $\rightsquigarrow$ terms

  $$M ::= x \mid f(M_1, \ldots, M_n) \mid k[M_1, \ldots, M_n]$$

  $\mathrm{pencrypt}(c_0, \mathrm{pk}(sk_A))$.
- Properties $\rightsquigarrow$ facts

  $$F ::= \mathrm{attacker}(M).$$
- Protocol, attacker $\rightsquigarrow$ Horn clauses

  $$F_1 \wedge \ldots \wedge F_n \rightarrow F$$

  $\mathrm{attacker}(m) \wedge \mathrm{attacker}(pk) \rightarrow \mathrm{attacker}(\mathrm{pencrypt}(m, pk))$.

Example - Cryptographic primitives

Public-key encryption:

- Encryption $\mathrm{pencrypt}(m, pk)$.
  $\mathrm{attacker}(m) \wedge \mathrm{attacker}(pk) \rightarrow \mathrm{attacker}(\mathrm{pencrypt}(m, pk))$
- Public key generation $\mathrm{pk}(sk)$.
  (builds a public key from a secret key)
  $\mathrm{attacker}(sk) \rightarrow \mathrm{attacker}(\mathrm{pk}(sk))$
- Decryption $\mathrm{pdecrypt}(\mathrm{pencrypt}(m, \mathrm{pk}(sk)), sk) \rightarrow m$.
  $\mathrm{attacker}(\mathrm{pencrypt}(m, \mathrm{pk}(sk))) \wedge \mathrm{attacker}(sk) \rightarrow \mathrm{attacker}(m)$

General treatment of primitives

- Constructors $f(M_1, \dots, M_n)$
  $\mathsf{attacker}(x_1) \wedge \dots \wedge \mathsf{attacker}(x_n) \rightarrow \mathsf{attacker}(f(x_1, \dots, x_n))$
- Destructors $g(M_1, \dots, M_n) \rightarrow M$
  $\mathsf{attacker}(M_1) \wedge \dots \wedge \mathsf{attacker}(M_n) \rightarrow \mathsf{attacker}(M)$

  (There may be several reductions defining a function.)

Exercise

Give clauses for shared-key encryption and signatures

Names

Normally, fresh names are created each time the protocol is run.
Here, we only distinguish two names when they are created after receiving different messages.

Each name k becomes a function of the messages previously received:

$$k[M_1, \ldots, M_n].$$

(Skolemisation)

These functions can only be applied by the principal that creates the name, not by the attacker.

Denning-Sacco protocol

- $A \to B : \{\{k\}_{sk_A}\}_{pk_B}$ k fresh

  A talks with any principal represented by its public key $\mathrm{pk}(x)$.
  A sends to it the message $\{\{k\}_{sk_A}\}_{\mathrm{pk}(x)}$.

  $\mathrm{attacker}(\mathrm{pk}(x)) \to \mathrm{attacker}(\mathrm{pencrypt}(\mathrm{sign}(k[\mathrm{pk}(x)], sk_A[]), \mathrm{pk}(x)))$.

- $B \to A : \{s\}_k$

  B has received a message $\{\{y\}_{sk_A}\}_{pk_B}$.
  B sends $\{s\}_y$.

  $\mathrm{attacker}(\mathrm{pencrypt}(\mathrm{sign}(y, sk_A[]), \mathrm{pk}(sk_B[]))) \to$
  $\mathrm{attacker}(\mathrm{sencrypt}(\mathrm{s}, y))$.

General coding of a protocol

If a principal A has received the messages $M_1, \ldots, M_n$ and sends the message M,

$$\text{attacker}(M_1) \wedge \ldots \wedge \text{attacker}(M_n) \rightarrow \text{attacker}(M).$$

Exercise

Model the Needham-Shroeder public key protocol protocol:

Message 1.	$A \rightarrow B$	$\{N_a, A\}_{pk_B}$	N_a fresh
Message 2.	$B \rightarrow A$	$\{N_a, N_b\}_{pk_A}$	N_b fresh
Message 3.	$A \rightarrow B$	$\{N_b\}_{pk_B}$	

Approximations

- The freshness of nonces is partially modeled.
- The number of times a message appears is ignored, only the fact that is has appeared is taken into account.
- The state of the principals is not fully modeled.

These approximations are keys for an efficient verification.

Solve the state space explosion problem.

No limit on the number of runs of the protocols.

⇒ essential for the certification of protocols.

Approximations: a more formal view

We can show formally by abstract interpretation that,
with respect to the multiset rewriting model,
the only approximation is that the number of repetitions of actions is ignored [Blanchet, IPL, 2005].

- Multiset rewriting ⇔ linear logic
- After approximation: classical logic
- The modeling of names by skolemisation does not introduce an approximation in classical logic.

Typical situation in which the proof fails:
a protocol first needs to keep some data secret,
and later reveals it.

Secrecy

Secrecy criterion

If attacker(M) *cannot be derived from the clauses, then* M *is secret.*

The term M cannot be built by an attacker.

The solving algorithm will determine whether a given fact can be derived from the clauses.

Overview

1. A variant of the spi-calculus
2. Intuitive presentation of the Horn clause representation
3. The solving algorithm
4. Experimental results
5. Formal translation from the spi-calculus.

Which resolution algorithm

A standard Prolog system would not terminate:

$$\text{attacker}(\text{sencrypt}(x, y)) \wedge \text{attacker}(y) \longrightarrow \text{attacker}(x)$$

generates bigger and bigger facts by SLD-resolution.

We need a different resolution strategy.

Saturation

Completion of the clause base, by resolution with free selection.

Selection function $sel(F_1 \wedge \ldots \wedge F_n \rightarrow F) \in \{F_1, \ldots, F_n, F\}$.

$$sel(F_1 \wedge \ldots \wedge F_n \rightarrow F) = \begin{cases} F \text{ if } \forall i \in \{1, \ldots, n\}, F_i = \text{attacker}(x) \\ F_i \text{ different from attacker}(x), \\ \qquad\quad \text{of maximal size, otherwise} \end{cases}$$

Saturation (2)

$$\frac{R = F_1 \wedge \ldots \wedge F_n \rightarrow F \qquad R' = F'_1 \wedge \ldots \wedge F'_{n'} \rightarrow F'}{\sigma F_1 \wedge \ldots \wedge \sigma F_n \wedge \sigma F'_2 \wedge \ldots \wedge \sigma F'_{n'} \rightarrow \sigma F'}$$

where σ is the most general unifier of F and F'_1,
$\quad sel(R) = F$, and $sel(R') = F'_1$.

Starting from an initial set of clauses $\mathcal{R}_0$,
perform this resolution step until a fixed point is reached,
eliminating subsumed clauses: $H \rightarrow C$ subsumes $H' \rightarrow C'$ when there exists σ such that $\sigma H \subseteq H'$ (multiset inclusion) and $\sigma C = C'$.

$\mathrm{saturate}(\mathcal{R}_0)$ is the set of obtained clauses R such that $sel(R)$ is the conclusion of R.

Saturation (3)

Example of a step:

$$\frac{\begin{array}{l}\text{attacker}(x) \wedge \text{attacker}(y) \rightarrow \text{attacker}(\text{pencrypt}(x, y)) \\ \text{attacker}(\text{pencrypt}(\text{sign}(z, sk_A[]), \text{pk}(sk_B[]))) \rightarrow \text{attacker}(\text{sencrypt}(\text{s}, z))\end{array}}{\text{attacker}(\text{sign}(z, sk_A[])) \wedge \text{attacker}(\text{pk}(sk_B[])) \rightarrow \text{attacker}(\text{sencrypt}(\text{s}, z))}$$

Theorem

The clauses obtained after saturation $\mathrm{saturate}(\mathcal{R}_0)$ *prove the same facts as the starting clauses* $\mathcal{R}_0$.

Proof (1): some notations

If $R = H \rightarrow C$, $R' = F_0 \wedge H' \rightarrow C'$, and σ is the most general unifier of C and F_0, then $R \circ_{F_0} R' = \sigma H \wedge \sigma H' \rightarrow \sigma C'$.

If R subsumes R', $R \sqsupseteq R'$.

$\mathcal{R}_0$: initial set of clauses.
$\mathcal{R}_1$: set of clauses when the fixpoint is reached.
$\mathcal{R}_2 = \mathrm{saturate}(\mathcal{R}_0) = \{H \rightarrow C \in \mathcal{R}_1 \mid \mathit{sel}(H \rightarrow C) = C\}$

Proof (2): derivation

Definition (Derivation)

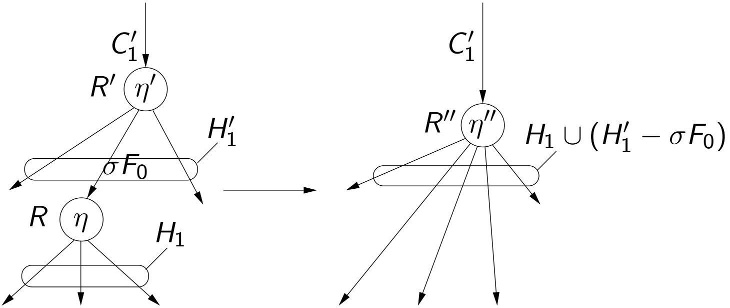
Let F be a closed fact. Let $\mathcal{R}$ be a set of clauses. A derivation of F from $\mathcal{R}$ is a finite tree defined as follows:

1. Its nodes (except the root) are labeled by clauses $R \in \mathcal{R}$.
2. Its edges are labeled by closed facts. (Edges go from a node to each of its sons.)
3. If the tree contains a node labeled by R with one incoming edge labeled by F_0 and n outgoing edges labeled by $F_1, \dots, F_n$, then $R \sqsupseteq \{F_1, \dots, F_n\} \rightarrow F_0$.
4. The root has one outgoing edge, labeled by F. The unique son of the root is named the *subroot*.

Proof (3): resolution step

Lemma (Resolution)

Consider a derivation containing a node η', labeled R'. Let F_0 be a hypothesis of R'. Then there exists a son η of η', labeled R, such that the edge $\eta' \to \eta$ is labeled by an instance of F_0, $R \circ_{F_0} R'$ is defined, and one obtains a derivation of the same fact by replacing the nodes η and η' with a node η'' labeled $R'' = R \circ_{F_0} R'$.



Proof (4): subsumption

Lemma (Subsumption)

If a node η of a derivation D is labeled by R, then one obtains a derivation D' of the same fact as D by relabeling η with a clause R' such that $R' \sqsupseteq R$.

By transitivity of $\sqsupseteq$.

Proof (5): saturation properties

Lemma (Saturation)

$\mathcal{R}_1$ satisfies the following properties:

1. *For all $R \in \mathcal{R}_0$, there exists $R' \in \mathcal{R}_1$ such that $R' \sqsupseteq R$;*
2. *Let $R = H \rightarrow C, R' = H' \rightarrow C' \in \mathcal{R}_1$. Assume that $sel(R) = C$, $sel(R') = F_0$, and $R \circ_{F_0} R'$ is defined. In this case, there exists $R'' \in \mathcal{R}_1$, $R'' \sqsupseteq R \circ_{F_0} R'$.*

1. A clause is removed only when it is subsumed by another one.
2. The fixpoint is reached.

Proof (6): If F is derivable from $\mathcal{R}_0$, then F is derivable from $\mathrm{saturate}(\mathcal{R}_0)$.

Consider a derivation of F from $\mathcal{R}_0$.

For each $R \in \mathcal{R}_0$, there exists $R' \in \mathcal{R}_1$ such that $R' \sqsupseteq R$ (Lemma saturation, Property 1).
We relabel each node labeled by $R \in \mathcal{R}_0 \setminus \mathcal{R}_1$ with $R' \in \mathcal{R}_1$ such that $R' \sqsupseteq R$ (by Lemma subsumption).
Therefore, we obtain a derivation D of F from $\mathcal{R}_1$.

Next, we build a derivation of F from $\mathcal{R}_2$, by transforming D.

Proof (7): If F is derivable from $\mathcal{R}_0$, then F is derivable from $\mathrm{saturate}(\mathcal{R}_0)$ (continued).

If D contains a clause not in $\mathcal{R}_2$, we transform D as follows.
Let η' be a lowest node of D labeled by a clause not in $\mathcal{R}_2$. All sons of η' are labeled by elements of $\mathcal{R}_2$.
Let R' be the clause labeling η'. Since $R' \notin \mathcal{R}_2$, $\mathit{sel}(R') = F_0$ is a hypothesis of R'.
By Lemma resolution, there exists a son of η of η' labeled by R, such that $R \circ_{F_0} R'$ is defined. Since all sons of η' are labeled by elements of $\mathcal{R}_2$, $R \in \mathcal{R}_2$. Hence $\mathit{sel}(R)$ is the conclusion of R. So, by Lemma saturation, Property 2, there exists $R'' \in \mathcal{R}_1$ such that $R'' \sqsupseteq R \circ_{F_0} R'$.
By Lemma resolution, we replace η and η' with η'' labeled by $R \circ_{F_0} R'$.
By Lemma subsumption, we replace $R \circ_{F_0} R'$ with R''.
The total number of nodes strictly decreases since η and η' are replaced with a single node η''. Hence, this replacement process terminates.
Upon termination, we obtain a derivation of F from $\mathcal{R}_2$.

Why it works

The facts attacker(x) unify with all facts attacker(M).

If we allow resolution on facts attacker(x), we will create many clauses.

The choice of the selection function implies that we avoid performing resolution upon attacker(x).

$\Rightarrow$ This is key to obtaining termination in most cases.

Derivation

$\mathrm{solve}_{\mathcal{R}_0}(\mathit{pred}(p_1, \dots, p_n)) = \{H \rightarrow \mathit{pred}(p'_1, \dots, p'_n) \mid H \rightarrow \mathit{pred}'(p'_1, \dots, p'_n) \in \mathrm{saturate}(\mathcal{R}_1)\}$, where pred' is a new predicate and $\mathcal{R}_1 = \mathcal{R}_0 \cup \{\mathit{pred}(p_1, \dots, p_n) \rightarrow \mathit{pred}'(p_1, \dots, p_n)\}$.

$\sigma \mathit{pred}(p_1, \dots, p_n)$ is derivable from $\mathcal{R}_0$ if and only if
$\sigma \mathit{pred}'(p_1, \dots, p_n)$ is derivable from $\mathcal{R}_1$ if and only if
$\sigma \mathit{pred}'(p_1, \dots, p_n)$ is derivable from $\mathrm{saturate}(\mathcal{R}_1)$ (previous theorem) if and only if
there exists a clause $H \rightarrow \mathit{pred}(p'_1, \dots, p'_n)$ in $\mathrm{solve}_{\mathcal{R}_0}(\mathit{pred}(p_1, \dots, p_n))$ and a substitution σ' such that $\sigma' \mathit{pred}(p'_1, \dots, p'_n) = \sigma \mathit{pred}(p_1, \dots, p_n)$ and $\sigma' H$ is derivable from $\mathrm{saturate}(\mathcal{R}_1)$.

If $\mathrm{solve}_{\mathcal{R}_0}(F) = \emptyset$, then no instance of F is derivable from $\mathcal{R}_0$.

Technique similar to the ordered resolution with selection [Weidenbach, CADE'99].

Optimizations

- Elimination of tautologies
- Elimination of duplicate hypotheses
- Elimination of hypotheses attacker(x) when x does not appear elsewhere.
- Tuples
- Secrecy assumptions: use conjectures to prune the search space.

Termination

The saturation algorithm does not always terminate,
but we have proved that it terminates for tagged protocols

That is, when each encryption, signature, ... is distinguished from others by a constant tag c_i

$$\{c_i, M_1, ..., M_n\}_K$$

- Large class of protocols
- Easy to add tags
- Good design practice

[Blanchet, Podelski, Fossacs'03]

Enforcing termination for all cases

Termination can be enforced by additional approximations.

Example: approximate clauses with clauses in decidable class $\mathcal{H}_1$.
[Nielson, Nielson, Seidel, SAS'02; Goubault-Larrecq, JFLA'04]

$\mathcal{H}_1$ = clauses whose conclusion is $P(f(x_1, \ldots, x_n))$, with distinct variables $x_1, \ldots, x_n$.

$$\frac{H \rightarrow P(f(p_1, \ldots, p_n)) \qquad p_1, \ldots, p_n \text{ are not all variables}}{Q_1(x_1), \ldots, Q_n(x_n) \rightarrow P(f(x_1, \ldots, x_n)) \qquad H \rightarrow Q_i(p_i)}$$

$$\frac{H \rightarrow P(f(x_1, \ldots, x_i, \ldots, x_i, \ldots, x_n))}{H, H\{x/x_i\} \rightarrow P(f(x_1, \ldots, x_i, \ldots, x, \ldots, x_n))}$$

Termination

- Ordered resolution with factorization and splitting
  [Comon, Cortier, 2003]
  Terminates on clauses with at most one variable.
  Protocols which blindly copy at most one term.
- Decision procedure for a class of tagged protocols
  without blind copies.
  [Ramanujam, Suresh, 2003]

Overview

1. A variant of the spi-calculus
2. Intuitive presentation of the Horn clause representation
3. The solving algorithm
4. Experimental results
5. Formal translation from the spi-calculus.

Experimental results

Pentium III, 1 GHz.

Protocol	Result	ms
Needham-Schroeder public key	Attack [Lowe]	10
Needham-Schroeder public key corrected	Secure	7
Needham-Schroeder shared key	Attack [Denning]	52
Needham-Schroeder shared key corrected	Secure	109
Denning-Sacco	Attack [AN]	6
Denning-Sacco corrected	Secure	7
Otway-Rees	Secure	10
Otway-Rees, variant of Paulson98	Attack [Paulson]	12
Yahalom	Secure	10
Simpler Yahalom	Secure	11
Main mode of Skeme	Secure	23

Overview

1. A variant of the spi-calculus
2. Intuitive presentation of the Horn clause representation
3. The solving algorithm
4. Experimental results
5. Formal translation from the spi-calculus.

Translation pi + crypto → Horn clauses

We consider a protocol P_0, executed in the presence of an S-adversary.

A protocol is translated into a set of Horn clauses using 2 predicates:

$\mathrm{mess}(p, p')$	the message p' may be sent on the channel p
$\mathrm{attacker}(p)$	the adversary may have p

Translation: attacker clauses

For each $a \in S$, $\text{attacker}(a[])$ (Init)

$\text{attacker}(b[])$ where b does not occur in P_0 (Name gen)

For each constructor f of arity n,
$\text{attacker}(x_1) \wedge \ldots \wedge \text{attacker}(x_n) \rightarrow \text{attacker}(f(x_1, \ldots, x_n))$ (Constr)

For each destructor g, for each reduction $g(M_1, \ldots, M_n) \rightarrow M$,
$\text{attacker}(M_1) \wedge \ldots \wedge \text{attacker}(M_n) \rightarrow \text{attacker}(M)$ (Destr)

$\text{mess}(x, y) \wedge \text{attacker}(x) \rightarrow \text{attacker}(y)$ (Listen)

$\text{attacker}(x) \wedge \text{attacker}(y) \rightarrow \text{mess}(x, y)$ (Send)

Translation: protocol clauses

ρ: environment (variables, names $\mapsto$ terms)
h: hypothesis (messages that must be received before reaching the current process)

- $[\![0]\!]\rho h = \emptyset$,
- $[\![P \mid Q]\!]\rho h = [\![P]\!]\rho h \cup [\![Q]\!]\rho h$,
- $[\![!P]\!]\rho h = [\![P]\!]\rho h$
- $[\![(\nu a)P]\!]\rho h = [\![P]\!](\rho[a \mapsto a[p_1, \ldots, p_n]])h$
  when $h = \mathrm{mess}(c_1, p_1) \wedge \ldots \wedge \mathrm{mess}(c_n, p_n)$.

Translation: protocol clauses (continued)

- $[\![M(x).P]\!]\rho h = [\![P]\!](\rho[x \mapsto x'])(h \wedge \mathrm{mess}(\rho(M), x'))$
  x' new variable
- $[\![\overline{M}\langle N\rangle.P]\!]\rho h = [\![P]\!]\rho h \cup \{h \rightarrow \mathrm{mess}(\rho(M), \rho(N))\}$
- $[\![\mathit{if}\ \ M = N\ \mathit{then}\ P\ \mathit{else}\ Q]\!]\rho h = [\![P]\!](\sigma\rho)(\sigma h) \cup [\![Q]\!]\rho h$
  where σ is the most general unifier of $\rho(M)$ and $\rho(N)$.
- $[\![\mathit{let}\ x = g(M_1, \ldots, M_n)\ \mathit{in}\ P\ \mathit{else}\ Q]\!]\rho h =$
  $\cup\{[\![P]\!]((\sigma\rho)[x \mapsto \sigma' p'])(\sigma h) \mid g(p'_1, \ldots, p'_n) \rightarrow p'$ is a rewrite rule of g and (σ, σ') is a most general pair of substitutions such that $\sigma\rho(M_1) = \sigma' p'_1, \ldots, \sigma\rho(M_n) = \sigma' p'_n\} \cup [\![Q]\!]\rho h$.

Example: Denning-Sacco protocol

Message 1. $A \rightarrow B : \quad \{\{k\}_{sk_A}\}_{pk_B} \quad k$ fresh

Message 2. $B \rightarrow A : \quad \{s\}_k$

$$(\nu sk_A)(\nu sk_B)\mathit{let}\ pk_A = \mathrm{pk}(sk_A)\ \mathit{in}\ \mathit{let}\ pk_B = \mathrm{pk}(sk_B)\ \mathit{in}$$
$$\overline{c}\langle pk_A\rangle\overline{c}\langle pk_B\rangle.$$

(A) $\quad ! \ c(x\_pk_B).(\nu k)\overline{c}\langle \mathrm{pencrypt}(\mathrm{sign}(k, sk_A), x\_pk_B)\rangle.$

$\quad c(x).\mathit{let}\ s = \mathrm{sdecrypt}(x, k)\ \mathit{in}\ 0$

(B) $\quad | \quad ! \ c(y).\mathit{let}\ y' = \mathrm{pdecrypt}(y, sk_B)\ \mathit{in}$

$\quad \mathit{let}\ k = \mathrm{checksign}(y', pk_A)\ \mathit{in}\ \overline{c}\langle \mathrm{sencrypt}(\mathrm{s}, k)\rangle$

Example: protocol clauses

$$[\![P_0]\!]\{c \mapsto c[]\}\emptyset$$

$$[\![let \ \ldots]\!]\{c \mapsto c[], sk_A \mapsto sk_A[], sk_B \mapsto sk_B[]\}\emptyset$$

$$[\![\overline{c}\langle pk_A\rangle \ldots]\!]\rho_0\emptyset$$

$$\rho_0 = \{c \mapsto c[], sk_A \mapsto sk_A[], sk_B \mapsto sk_B[], pk_A \mapsto \mathrm{pk}(sk_A[]), pk_B \mapsto \mathrm{pk}(sk_B[])\}$$

$$[\![!P_A \mid !P_B]\!]\rho_0\emptyset$$

$\cup \{\mathrm{mess}(c[], \mathrm{pk}(sk_A[])),$ comes from $\overline{c}\langle pk_A\rangle$

$\mathrm{mess}(c[], \mathrm{pk}(sk_B[]))\}$ comes from $\overline{c}\langle pk_B\rangle$

$$[\![P_A]\!]\rho_0\emptyset \cup [\![P_B]\!]\rho_0\emptyset \cup \{\mathrm{attacker}(\mathrm{pk}(sk_A[])), \mathrm{attacker}(\mathrm{pk}(sk_B[]))\}$$

Note: $\mathrm{attacker}(M)$ is equivalent to $\mathrm{mess}(c[], M)$ when $c \in S$, by (Listen) and (Send).

Example: protocol clauses (A)

$[\![P_A]\!]\rho_0\emptyset$

$[\![(\nu k)\ldots]\!]\ \rho_0[x\_pk_B \mapsto x_{pk_B}]\ \mathsf{mess}(c[], x_{pk_B})$

$[\![\overline{c}\langle \mathsf{pencrypt}(\ldots)\rangle\ldots]\!]\ \rho_0[x\_pk_B \mapsto x_{pk_B}, k \mapsto k[x_{pk_B}]]\ \mathsf{mess}(c[], x_{pk_B})$

$[\![c(x)\ldots]\!]\ \rho_0[x\_pk_B \mapsto x_{pk_B}, k \mapsto k[x_{pk_B}]]\ \mathsf{mess}(c[], x_{pk_B})$
$\cup \{\mathsf{mess}(c[], x_{pk_B}) \rightarrow \mathsf{mess}(c[], \mathsf{pencrypt}(\mathsf{sign}(k[x_{pk_B}], sk_A[]), x_{pk_B}))\}$

$\{\mathsf{mess}(c[], x_{pk_B}) \rightarrow \mathsf{mess}(c[], \mathsf{pencrypt}(\mathsf{sign}(k[x_{pk_B}], sk_A[]), x_{pk_B}))\}$

Example: protocol clauses (B)

$[\![P_B]\!]\rho_0\emptyset$

$[\![\mathit{let}\ y' \ldots]\!]\ \rho_0[y \mapsto y]\ \mathrm{mess}(c[], y)$

$[\![\mathit{let}\ k \ldots]\!]\ \rho_0[y \mapsto \mathrm{pencrypt}(y', \mathrm{pk}(sk_B[])), y' \mapsto y']$
$\quad \mathrm{mess}(c[], \mathrm{pencrypt}(y', \mathrm{pk}(sk_B[])))$

$[\![\overline{c}\langle \ldots \rangle]\!]\ \rho_0[y \mapsto \mathrm{pencrypt}(\mathrm{sign}(k, sk_A[]), \mathrm{pk}(sk_B[])), y' \mapsto \mathrm{sign}(k, sk_A[]),$
$\quad k \mapsto k]\ \mathrm{mess}(c[], \mathrm{pencrypt}(\mathrm{sign}(k, sk_A[]), \mathrm{pk}(sk_B[])))$

$\{\mathrm{mess}(c[], \mathrm{pencrypt}(\mathrm{sign}(k, sk_A[]), \mathrm{pk}(sk_B[]))) \rightarrow \mathrm{mess}(c[], \mathrm{sencrypt}(\mathrm{s}, k))\}$

Proof of secrecy

Closed process: P_0
Initial knowledge of the adversary: S finite set of names
Clauses for the protocol and the adversary: $\mathcal{R}_{P_0,S}$.

Theorem

If attacker(s) *cannot be derived from* $\mathcal{R}_{P_0,S}$*,*
then P_0 *preserves the secrecy of* s *from* S*.*

Theorem

If $\mathrm{solve}_{\mathcal{R}_{P_0,S}}(\text{attacker(s)}) = \emptyset$*,*
then P_0 *preserves the secrecy of* s *from* S*.*

Example

For the Denning-Sacco protocol, attacker(s) is derivable from the clauses.

The derivation corresponds to the description of the known attack.

For the corrected version, attacker(s) is not derivable from the clauses: s is secret.

Comparison with typing [Abadi, Blanchet, POPL'02 and JACM'05]

We have defined a generic type system for the explained variant of the spi-calculus.

Theorem

A secrecy property can be proved by the Horn clause verifier

$\Leftrightarrow$

it can be proved by any instance of the type system.

A tight relation between two superficially different frameworks.

Extension to equational theories: Diffie-Hellman

Goal: Establish a shared key between two participants

Message 1.	$A \to B$:	g^{n_0}	n_0 fresh
Message 2.	$B \to A$:	g^{n_1}	n_1 fresh

A computes $k = (g^{n_1})^{n_0}$, B computes $k = (g^{n_0})^{n_1}$.
The exponentiation is such that these quantities are equal.

$$(g^{n_1})^{n_0} = (g^{n_0})^{n_1}$$

The exponentiation is computed in $\mathbb{Z}_p^*$, where p is a prime and g generator of $\mathbb{Z}_p^*$.

Extension to equational theories: Diffie-Hellman example

Simplified version of the secure shell protocol (SSH):

Message 1.	$C \rightarrow S$:	$KExDHInit, g^{n_0}$	n_0 fresh
Message 2.	$S \rightarrow C$:	$KExDHReply, pk_S, g^{n_1}, \{h\}_{sk_S}$	n_1 fresh

where $K = (g^{n_1})^{n_0} = (g^{n_0})^{n_1}$
and $h = H((pk_S, g^{n_0}, g^{n_1}, K))$.
K and h are shared secrets between C (client) and S (server).
They are used to compute encryption keys.

Extension to equational theories: other examples

- XOR: associative, commutative, $xor(x, x) = 0$, $xor(x, 0) = x$
- Primitives whose success is not observable
  (for decryption for instance)

$$\begin{aligned}\mathrm{sdecrypt}(\mathrm{sencrypt}(x, y), y) &= x\\ \mathrm{sencrypt}(\mathrm{sdecrypt}(x, y), y) &= x\end{aligned}$$

- Subtle interactions between primitives
  Example: XOR and crc

$$crc(xor(x, y)) = xor(crc(x), crc(y))$$

Extension to equational theories

We have built algorithms that translate the equations into a set of rewrite rules, which generates enough terms (equal modulo the equational theory). [Blanchet, Abadi, Fournet, JLAP'08]

We have shown that, for each trace with equations, there is a corresponding trace with rewrite rules, and conversely.

Efficient because it avoids unification modulo.
(Standard syntactic resolution can still be used.)

Still fairly limited, since it leads to non-termination for many equational theories.
(For example, cannot handle theories that contain associativity.)

Extension to equational theories: Diffie-Hellman

Equation:

$$(\mathrm{g}\hat{\ }x)\hat{\ }y = (\mathrm{g}\hat{\ }y)\hat{\ }x$$

is translated into the rewrite rules:

$$\mathrm{g} \rightarrow \mathrm{g} \qquad x\hat{\ }y \rightarrow x\hat{\ }y \qquad (\mathrm{g}\hat{\ }x)\hat{\ }y \rightarrow (\mathrm{g}\hat{\ }y)\hat{\ }x$$

Terms may have several normal forms: applying ˆ to $\mathrm{g}\hat{\ }x$ and y yields two normal forms of $(g\hat{\ }x)\hat{\ }y$: $(g\hat{\ }x)\hat{\ }y$ and $(g\hat{\ }y)\hat{\ }x$.

Extension to equational theories: encryption

Equations:

$$\mathsf{sdecrypt}(\mathsf{sencrypt}(x,y),y) = x$$
$$\mathsf{sencrypt}(\mathsf{sdecrypt}(x,y),y) = x$$

are translated into the rewrite rules:

$$\mathsf{sdecrypt}(x,y) \rightarrow \mathsf{sdecrypt}(x,y) \qquad \mathsf{sencrypt}(x,y) \rightarrow \mathsf{sencrypt}(x,y)$$
$$\mathsf{sdecrypt}(\mathsf{sencrypt}(x,y),y) \rightarrow x \qquad \mathsf{sencrypt}(\mathsf{sdecrypt}(x,y),y) \rightarrow x$$

Each term has a single normal form, irreducible by $\mathsf{sdecrypt}(\mathsf{sencrypt}(x,y),y) \rightarrow x$ and $\mathsf{sencrypt}(\mathsf{sdecrypt}(x,y),y) \rightarrow x$.

Extension to equational theories

Unification modulo the equational theory could be used,
for example to handle associativity and commutativity.

- Better model of Diffie-Hellman (modelling the multiplicative group plus the exponentiation).
  [Meadows, Narendran, WITS'02]
  [Goubault-Larrecq, Roger, Verma, JLAP'04]
- XOR
  [Comon, Shmatikov, LICS'03]
  [Chevalier, Küsters, Rusinowitch, Turuani, LICS'03]
  (Bounded number of sessions)

Conclusion: Some other results

- Automatic proof of correspondence assertions (authentication) [Blanchet, JCS, to appear]
- Automatic proof of strong secrecy [Blanchet, Oakland'04] and other observational equivalences [Blanchet, Abadi, Fournet, LICS'05 and JLAP'08]
- Reconstruction of attacks from derivations [Allamigeon, Blanchet, CSFW'05]
- Case studies: Certified email protocol [Abadi, Blanchet, SAS'03], JFK [Abadi, Blanchet, Fournet, ESOP'04], Plutus [Blanchet, Chaudhuri, S&P'08]

Software and papers at `www.proverif.ens.fr`

Conclusion: Advantages of this technique

- A particularly efficient verifier
- Can handle complex protocols (JFK, ...)
- Unbounded number of runs of the protocol
  Unbounded message size

  ⇒ Can be used for certification of protocols
- Can prove various properties: secrecy, correspondences, observational equivalence
- Can handle a wide range of cryptographic primitives, specified by rewrite rules or by equations.

Conclusion: Limitations

- The proofs are done in the Dolev-Yao model. We would like automatic proof of protocols in a computational setting. There is a recent tool for that: CryptoVerif.
- The proofs are done on a model of the protocol. We would like automatic proof of implementations of protocols (Already some work, for example [Goubault-Larrecq, Parennes, VMCAI'05], [Bhargavan, Fournet, Gordon, Tse, CSFW'06])