Computationally Sound Automatic Proofs of Correspondence Assertions

Bruno Blanchet
CNRS, Ecole Normale Supérieure
blanchet@di.ens.fr

January 2007

Introduction

Our goal: implement an automatic, computationally sound prover for security protocols.

We have already implemented a prover for secrecy properties.

In this talk, we show how to extend it to correspondence assertions, that is, properties of the style:

> *If some event has been executed, then some other events have been executed.*

Basic application: authentication.

Proofs by sequences of games

The prover produces the proof is a sequence of games, as in Shoup's or Bellare and Rogaway's method:

- In the first game, the adversary plays against the real protocol.
- The prover transforms each game into the next by syntactic transformations or by applying security assumptions on cryptographic primitives.

  Consecutive games are computationally indistinguishable.
- The desired security property can be proved directly on the last game.

Games are formalized in a process calculus.

Differences with secrecy

For the proof of correspondences:

- The language for games in the same as for secrecy, except for the addition of events.
- The game transformations and the proof strategy are the same as for secrecy. (The events are left unchanged.)
- One needs a new algorithm for checking correspondences on the last game.

Example: a nonce challenge

Simple example inspired by the corrected Woo-Lam public-key protocol (1997)

$$B \rightarrow A : (N, B)$$
$$A \rightarrow B : \{pk_A, B, N\}_{sk_A}$$

In our language:

$c0(); \textbf{new}\ rk_A : keyseed;$

$\textbf{let}\ pk_A = pkgen(rk_A)\ \textbf{in let}\ sk_A = skgen(rk_A)\ \textbf{in}\ \overline{c1}\langle pk_A\rangle;$

$!^{i_A \leq n} c2[i_A](xN : nonce, xB : host); \textbf{new}\ r : seed;$

$\textbf{event}\ e_A(pk_A, xB, xN); \overline{c3[i_A]}\langle sign(concat(pk_A, xB, xN), sk_A, r)\rangle$

$| !^{i_B \leq n} c4[i_B](xpk_A : pkey); \textbf{new}\ N : nonce; \overline{c5[i_B]}\langle N, B\rangle;$

$c6[i_B](s : signature); \textbf{if}\ verify(concat(xpk_A, B, N), xpk_A, s)\ \textbf{then}$

$\textbf{if}\ xpk_A = pk_A\ \textbf{then}\ \textbf{event}\ e_B(xpk_A, B, N)$

After game transformations

Using the unforgeability of signatures, the signature verification with pk_A succeeds only for signatures generated with sk_A.
After game transformations, we obtain the last game:

$c0(); \mathbf{new}\ rk_A : keyseed; \mathbf{let}\ pk_A = pkgen'(rk_A)\ \mathbf{in}\ \overline{c1}\langle pk_A\rangle;$

$!^{i_A \leq n} c2[i_A](xN : nonce, xB : host);$

$\quad \mathbf{event}\ e_A(pk_A, xB, xN); \mathbf{let}\ m = concat(pk_A, xB, xN)\ \mathbf{in}$

$\quad \mathbf{new}\ r : seed; \overline{c3[i_A]}\langle sign'(m, skgen'(rk_A), r)\rangle$

$| !^{i_B \leq n} c4[i_B](xpk_A : pkey); \mathbf{new}\ N : nonce; \overline{c5[i_B]}\langle N, B\rangle;$

$\quad c6[i_B](s : signature); \mathbf{find}\ u \leq n\ \mathbf{suchthat}$

$\quad\quad \mathbf{defined}(m[u], xB[u], xN[u]) \wedge (xpk_A = pk_A) \wedge (B = xB[u])$

$\quad\quad \wedge (N = xN[u]) \wedge verify'(concat(xpk_A, B, N), xpk_A, s)\ \mathbf{then}$

$\quad \mathbf{event}\ e_B(xpk_A, B, N)$

Non-injective correspondences

A non-injective correspondence is a formula of the form $\psi \Rightarrow \phi$ where

$\phi ::=$	formula
M	term (without arrays)
$event(e(M_1, \ldots, M_m))$	event
$\phi_1 \wedge \phi_2$	conjunction
$\phi_1 \vee \phi_2$	disjunction

and ψ is a formula that contains only events and conjunctions.

Example

$$event(e_B(x, y, z)) \Rightarrow event(e_A(x, y, z))$$

means that, if $e_B(x, y, z)$ is executed, then $e_A(x, y, z)$ has also been executed (except in cases of negligible probability).

Non-injective correspondences: formal semantics

Let ρ be an environment that maps variables to bitstrings.
Let $\mathcal{E}$ be a sequence of events.

Definition

$\rho, \mathcal{E} \vdash M$ if and only if M evaluates to *true* in environment ρ
$\rho, \mathcal{E} \vdash \mathit{event}(e(M_1, \ldots, M_m))$ if and only if
for all $j \leq m$, M_j evaluates to a_j in ρ and $e(a_1, \ldots, a_m) \in \mathcal{E}$

Definition

$\mathcal{E} \vdash \psi \Rightarrow \phi$ if and only if
for all ρ defined on $\mathrm{var}(\psi)$ such that $\rho, \mathcal{E} \vdash \psi$,
there exists an extension ρ' of ρ to $\mathrm{var}(\phi)$ such that $\rho', \mathcal{E} \vdash \phi$.

Definition

Q_0 satisfies $\psi \Rightarrow \phi$ with public variables V if and only if
for all evaluation contexts C accessing only variables of V in Q_0,
$\Pr[C[Q_0] \text{ executes } \mathcal{E} \text{ and } \mathcal{E} \not\vdash \psi \Rightarrow \phi]$ is negligible.

Injective correspondences

An injective correspondence also allows injective events $\mathit{inj\text{-}event}(e(M_1, \ldots, M_m))$.

Each execution of the injective events in ψ corresponds to distinct injective events in ϕ.

Example

$$\mathit{inj\text{-}event}(e_B(x, y, z)) \Rightarrow \mathit{inj\text{-}event}(e_A(x, y, z))$$

means that each execution of $e_B(x, y, z)$ corresponds to a distinct execution of $e_A(x, y, z)$.

Collecting true facts

For each program point P, we collect a set of true facts at that point $\mathcal{F}_P$.

- We take into account assignments and tests above P.

Example

In **if** M **then** P, $M \in \mathcal{F}_P$.

- We take into account facts that hold at all definitions of variables.

Example

If **defined**$(x[\widetilde{M}]) \in \mathcal{F}_P$ and M holds at all definitions of $x[\widetilde{i}]$, then $M\{\widetilde{M}/\widetilde{i}\} \in \mathcal{F}_P$.

- We take into account that code is always executed up to the next output before switching to another thread.

Collecting true facts: example (1)

$c0(); \mathbf{new}\ rk_A : keyseed; \mathbf{let}\ pk_A = pkgen'(rk_A)\ \mathbf{in}\ \overline{c1}\langle pk_A\rangle;$

$!^{i_A \leq n} c2[i_A](xN : nonce, xB : host);$

$\quad \mathbf{event}\ e_A(pk_A, xB, xN); \mathbf{let}\ m = concat(pk_A, xB, xN)\ \mathbf{in}$

$\quad \mathbf{new}\ r : seed; \overline{c3[i_A]}\langle sign'(m, skgen'(rk_A), r)\rangle$

$| !^{i_B \leq n} c4[i_B](xpk_A : pkey); \mathbf{new}\ N : nonce; \overline{c5[i_B]}\langle N, B\rangle;$

$\quad c6[i_B](s : signature); \mathbf{find}\ u \leq n\ \mathbf{suchthat}$

$\quad\quad \mathbf{defined}(m[u], xB[u], xN[u]) \wedge (xpk_A = pk_A) \wedge (B = xB[u])$

$\quad\quad \wedge (N = xN[u]) \wedge verify'(concat(xpk_A, B, N), xpk_A, s)\ \mathbf{then}$

$\quad \mathbf{event}\ e_B(xpk_A, B, N)$

At program point $P = \mathbf{event}\ e_B(xpk_A, B, N)$,

$\mathcal{F}_P = \{\mathbf{defined}(m[u[i_B]]), \mathbf{defined}(xB[u[i_B]]), \mathbf{defined}(xN[u[i_B]]),$

$\quad xpk_A[i_B] = pk_A, B = xB[u[i_B]], N[i_B] = xN[u[i_B]], \ldots$

Collecting true facts: example (2)

$$c0(); \textbf{new } rk_A : keyseed; \textbf{let } pk_A = pkgen'(rk_A) \textbf{ in } \overline{c1}\langle pk_A\rangle;$$
$$!^{i_A \leq n} c2[i_A](xN : nonce, xB : host);$$
$$\textbf{event } e_A(pk_A, xB, xN); \textbf{let } m = concat(pk_A, xB, xN) \textbf{ in}$$
$$\textbf{new } r : seed; \overline{c3[i_A]}\langle sign'(m, skgen'(rk_A), r)\rangle$$
$$| \ldots$$

At program point $P = \textbf{event } e_B(xpk_A, B, N)$,

$$\mathcal{F}_P = \{\textbf{defined}(m[u[i_B]]), \textbf{defined}(xB[u[i_B]]), \textbf{defined}(xN[u[i_B]]),$$
$$xpk_A[i_B] = pk_A, B = xB[u[i_B]], N[i_B] = xN[u[i_B]],$$
$$event(e_A(pk_A, x_B[u[i_B]], xN[u[i_B]])), \ldots\}$$

because $\textbf{defined}(m[u[i_B]]) \in \mathcal{F}_P$.

Equational prover

We use an algorithm inspired by the Knuth-Bendix completion algorithm.

We rewrite pairs $\mathcal{F}, \mathcal{R}$, where $\mathcal{F}$ is a set of facts and $\mathcal{R}$ is a set of rewrite rules $M_1 \rightarrow M_2$.

Rewrite rules stand for oriented equalities: when $(M_1 \rightarrow M_2) \in \mathcal{R}$, $M_1 = M_2$.

$$\frac{\mathcal{F} \cup \{x[M_1, \ldots, M_m] = x[M'_1, \ldots, M'_m]\}, \mathcal{R}}{\mathcal{F} \cup \{M_1 = M'_1, \ldots, M_m = M'_m\}, \mathcal{R}} \text{ if } x \text{ defined by restrictions } \textbf{new } x : T,\ T \text{ large type}$$

$$\frac{\mathcal{F} \cup \{F\}, \mathcal{R}}{\mathcal{F} \cup \{F'\}, \mathcal{R}} \text{ if } F \rightarrow_{\mathcal{R}} F' \qquad \frac{\mathcal{F}, \mathcal{R} \cup \{M_1 \rightarrow M_2\}}{\mathcal{F} \cup \{M_1 = M'_2\}, \mathcal{R}} \text{ if } M_2 \rightarrow_{\mathcal{R}} M'_2$$

$$\frac{\mathcal{F} \cup \{M = M'\}, \mathcal{R}}{\mathcal{F}, \mathcal{R} \cup \{M \rightarrow M'\}} \text{ if } M > M' \qquad \frac{\mathcal{F}, \mathcal{R} \cup \{M_1 \rightarrow M_2\}}{\mathcal{F} \cup \{M'_1 = M_2\}, \mathcal{R}} \text{ if } M_1 \rightarrow_{\mathcal{R}} M'_1$$

We say that $\mathcal{F}$ yields a contradiction when the prover starting from $(\mathcal{F}, \emptyset)$ derives $(\mathcal{F}', \mathcal{R}')$ with $\mathit{false} \in \mathcal{F}'$.

Proof of non-injective correspondences (1)

Let $\psi \Rightarrow \phi = F_1 \wedge \ldots \wedge F_m \Rightarrow \phi$ be a non-injective correspondence, with fresh variables.

Example

$event(e_B(x, y, z)) \Rightarrow event(e_A(x, y, z))$.

If $F_1, \ldots, F_m$ have been executed,
then there exist $P_1, \ldots, P_m$ such that, for all $j \leq m$,

- $F_j = event(e_j(M_{j1}, \ldots, M_{jm_j}))$,
- **event** $e_j(M'_{j1}, \ldots, M'_{jm_j})$; P_j occurs in Q_0, and
- **event** $e_j(M'_{j1}, \ldots, M'_{jm_j})$ has been executed with $F_j = \theta'_j event(e_j(M'_{j1}, \ldots, M'_{jm_j}))$, where θ'_j renames the replication indices at P_j to fresh replication indices.

Example

event $e_B(xpk_A, B, N)$ has been executed, with
$event(e_B(x, y, z)) = event(e_B(xpk_A[i'_B], B, N[i'_B]))$, $\theta' = \{i'_B / i_B\}$.

Proof of non-injective correspondences (2)

Then the facts $\mathcal{F}_j = \theta'_j \mathcal{F}_{P_j} \cup \{\theta'_j M'_{j1} = M_{j1}, \ldots, \theta'_j M'_{jm_j} = M_{jm_j}\}$ hold.

Example

$\mathcal{F}_P\{i'_B/i_B\} \cup \{x = xpk_A[i'_B], y = B, z = N[i'_B]\}$ hold, where $\mathcal{F}_P$ has been described previously.

For each such $P_1, \ldots, P_m$, we show that

$$\mathcal{F} = \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_m \text{ implies } \theta\phi$$

for some θ equal to the identity on $\mathrm{var}(\psi)$, by the equational prover.

(For this proof, we prove atomic facts contained in $\theta\phi$, we choose θ by matching facts to prove with elements of $\mathcal{F}$, and we show that $\mathcal{F}$ implies M by showing that $\mathcal{F} \cup \{\neg M\}$ yields a contradiction.)

Example

$x = xpk_A[i'_B], y = B, z = N[i'_B], xpk_A[i'_B] = pk_A, B = xB[u[i'_B]]$, $N[i'_B] = xN[u[i'_B]]$, $event(e_A(pk_A, x_B[u[i'_B]], xN[u[i'_B]]))$ imply $event(e_A(x, y, z))$.

Proof of injective correspondences (1)

We add replication indices to events.

Example

event $e_A(i_A, pk_A, xB, xN)$, **event** $e_B(i_B, xpk_A, B, N)$.
$inj\text{-}event(e_B(i, x, y, z)) \Rightarrow inj\text{-}event(e_A(i', x, y, z))$.

For each injective event in ϕ, we collect tuples $(\mathcal{F}, \widetilde{M}, \mathcal{I}, \mathcal{V})$ in set $\mathcal{S}$:

- $\mathcal{F} = \mathcal{F}_1 \cup \ldots \cup \mathcal{F}_m$, the facts that are known to hold
- $\widetilde{M}$, the replication indices at that event in $\mathcal{F}$
- $\mathcal{I} = \{j \mapsto \mathrm{Im}(\theta'_j) \mid F_j \text{ is an injective event}\}$
- $\mathcal{V}$, the variables of ψ and of the image of $\theta'_1, \ldots, \theta'_m$

Example

$\mathcal{S} = \{(\mathcal{F}, \widetilde{M}, \mathcal{I}, \mathcal{V})\}$ where
$\mathcal{F} = \mathcal{F}_P\{i'_B/i_B\} \cup \{x = xpk_A[i'_B], y = B, z = N[i'_B]\}$,
$event(e_A(u[i'_B], pk_A, x_B[u[i'_B]], xN[u[i'_B]])) \in \mathcal{F}$,
$\widetilde{M} = u[i'_B]$, $\mathcal{I} = \{1 \mapsto i'_B\}$, $\mathcal{V} = \{x, y, z, i'_B\}$.

Proof of injective correspondences (2)

For each $(\mathcal{F}, \widetilde{M}, \mathcal{I}, \mathcal{V})$, $(\mathcal{F}', \widetilde{M}', \mathcal{I}', \mathcal{V}')$ in $\mathcal{S}$, we show that

$$\mathcal{F} \cup \theta'' \mathcal{F}' \cup \{\widetilde{M} = \theta'' \widetilde{M}', \bigvee_{j \in \mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta'' \mathcal{I}'(j)\}$$

yields a contradiction, where θ'' renames the variables of $\mathcal{V}'$ to fresh variables.

This result shows injectivity: the injective events in ψ cannot be executed twice (different replication indices $\bigvee_{j \in \mathrm{Dom}(\mathcal{I})} \mathcal{I}(j) \neq \theta'' \mathcal{I}'(j)$) with a single corresponding injective event in ϕ (same replication indices $\widetilde{M} = \theta'' \widetilde{M}'$).

Example

$N[i'_B] = xN[u[i'_B]]$, $N[i''_B] = xN[u[i''_B]]$, $u[i'_B] = u[i''_B]$, $i'_B \neq i''_B$ yield a contradiction.
($N[i'_B] = N[i''_B]$ implies $i'_B = i''_B$ except in cases of negligible probability, by eliminating collisions on N.)

Authentication and key exchange

Consider a protocol between A and B:

$$Q_0 = \mathit{Init}; (!^{i_A \leq n} Q_A \mid !^{i_B \leq n} Q_B \mid Q_S)$$

We assume that

- A is the initiator, B is the responder;
- the protocol has an odd number of rounds r;
- the messages of the protocol are stored in $x_1, \ldots, x_r$ by A, $y_1, \ldots, y_r$ by B;
- when A accepts with Y, it sends $x_r, \mathit{accept}_A(Y)$;
  when B accepts with X, it sends $\mathit{accept}_B(X)$.

Mutual authentication: definition

Q_0 is a secure mutual authentication protocol with session ids sid and sid' if

1. If the adversary just relays messages between $Q_A[i]$ and $Q_B[i']$, then $Q_A[i]$ accepts with B and $Q_B[i']$ accepts with A.
2. The following are true with overwhelming probability:
   - there exists an injective function that maps each index i of a session $Q_A[i]$ that accepts with B to the index i' of a session $Q_B[i']$ with expected partner A with the same session id: $sid'(x_1[i], \ldots, x_{r-1}[i]) = sid'(y_1[i'], \ldots, y_{r-1}[i'])$;
   - there exists an injective function that maps each index i' of a session $Q_B[i']$ that accepts with A to the index i of a session $Q_A[i]$ that accepts with B with the same session id: $sid(x_1[i], \ldots, x_r[i]) = sid(y_1[i'], \ldots, y_r[i'])$.

Mutual authentication: proof

Let Q_0' be obtained from Q_0 by adding

- **event** $part_A(Y, sid'(x_1, \ldots, x_{r-1}))$;
  event $full_A(Y, sid(x_1, \ldots, x_r))$
  just before A sends x_r, $accept_A(Y)$;
- **event** $part_B(X, sid'(y_1, \ldots, y_{r-1}))$ just before B sends y_{r-1}.
- **event** $full_B(X, sid(y_1, \ldots, y_r))$ just before B sends $accept_B(X)$;

If Q_0 satisfies Condition 1 of the previous definition
and Q_0' satisfies the correspondences

$$inj\text{-}event(part_A(B, x)) \Rightarrow inj\text{-}event(part_B(A, x))$$
$$inj\text{-}event(full_B(A, x)) \Rightarrow inj\text{-}event(full_A(B, x))$$

with public variables $V = \emptyset$,
then Q_0 is a secure mutual authentication protocol with session ids sid and sid'.

Authenticated key exchange: definition

Q_0 is a secure authenticated key exchange over T with session ids sid and sid' if

1. Q_0 is a secure mutual authentication protocol with session ids sid and sid';
2. if the adversary just relays messages between Q_A^i and $Q_B^{i'}$, then Q_A^i accepts with B, $Q_B^{i'}$ accepts with A, $k_A[i] = k_B[i']$, and this random variable is uniformly distributed in T;
3. the adversary cannot distinguish the keys k_A, k_B from a independent random numbers with several test queries.

   (In the "random" case, the test queries return the same result when they are asked to the same oracle or to a partner oracle with the same session id.)

Authenticated key exchange: proof (1)

Let Q'_0 be obtained from Q_0 by

- replacing $\overline{c_{Ar}[i_A]}\langle x_r, accept_A(Y)\rangle$ with

  event $part_A(Y, sid'(x_1, \ldots, x_{r-1}))$;
  event $full_A(Y, k_A, sid(x_1, \ldots, x_r))$;
  if $Y = B$ **then let** $k'_A = k_A$ **in** $\overline{c_{Ar}[i_A]}\langle x_r, accept_A(Y)\rangle$
  else $\overline{c_{Ar}[i_A]}\langle x_r, accept_A(Y)\rangle; c_{AK}[i_A](); \overline{c_{AK}[i_A]}\langle k_A\rangle$

- replacing $\overline{c_{Br+1}[i_B]}\langle accept_B(X)\rangle$ with

  event $full_B(X, k_B, sid(y_1, \ldots, y_r))$;
  if $X = A$ **then** $\overline{c_{Br+1}[i_B]}\langle accept_B(X)\rangle$
  else $\overline{c_{Br+1}[i_B]}\langle accept_B(X)\rangle; c_{BK}[i_B](); \overline{c_{BK}[i_B]}\langle k_B\rangle$

- and adding **event** $part_B(X, sid'(y_1, \ldots, y_{r-1}))$ just before Q_B sends y_{r-1}.

Authenticated key exchange: proof (2)

If the following conditions are satisfied:

1. Q_0 satisfies Condition 2 of the previous definition;
2. Q'_0 preserves the secrecy of k'_A;
3. Q'_0 satisfies the correspondences

$$\textit{inj-event}(\textit{part}_A(B, x)) \Rightarrow \textit{inj-event}(\textit{part}_B(A, x))$$
$$\textit{inj-event}(\textit{full}_B(A, k, x)) \Rightarrow \textit{inj-event}(\textit{full}_A(B, k, x))$$
$$\textit{event}(\textit{full}_B(A, k, x)) \wedge \textit{event}(\textit{full}_A(B, k', x)) \Rightarrow k = k'$$

with public variables $\{k'_A\}$

then Q_0 is a secure authenticated key exchange with session ids sid and sid'.

Intuition: The correspondences show that each key of B with A is an element of array k'_A, so the secrecy of k'_A is sufficient.

Experimental results

We have tested our prover on the following protocols:

- Woo-Lam shared-key original and corrected versions
- Woo-Lam public-key original and corrected versions
- Needham-Schroeder public-key original and corrected versions
- Denning-Sacco public-key original and corrected versions
- Needham-Schroeder shared-key original and corrected versions, with and without key confirmation
- Yahalom, with and without key confirmation
- Otway-Rees

We try to prove authentication and authenticated key exchange, as appropriate for each protocol.

The prover obviously fails proving false properties.

It succeeds proving true ones, except in one case:
the original version of the Needham-Schroeder shared-key protocol.
(It does not see that $N_B[i] \neq N_B[i'] - 1$ except in cases of negligible probability.)

Conclusion and future work

The prover succeeds proving most desired correspondences, but some points could still be improved:

- automatic proof strategy: the prover sometimes needs indications from the user.
- equational theories, e.g. handle the equations of XOR.
- handle more primitives, e.g. Diffie-Hellman key agreements.