Computationally Sound Mechanized Proofs of Correspondence Assertions

Bruno Blanchet
Inria, Paris
Bruno.Blanchet@inria.fr

Year 2022-23
Our goal: implement an automatic, *computationally sound* prover for security protocols.

We have already discussed *secrecy properties*.

In this talk, we show how to prove 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.*
Differences with secrecy

For the proof of correspondences:

- The language for games is 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:

\[
\text{in}(c_0, ()); \text{new } rk_A : \text{keyseed}; \\
\text{let } pk_A = \text{pkgen}(rk_A) \text{ in let } sk_A = \text{skgen}(rk_A) \text{ in out}(c_1, pk_A); \\
!^{i_A \leq n}\text{in}(c_2[i_A], (xN : \text{nonce}, xB : \text{host})); \text{event } e_A(pk_A, xB, xN); \\
\text{new } r : \text{seed}; \text{out}(c_3[i_A], \text{sign} (\text{concat}(pk_A, xB, xN), sk_A, r)) \\
| !^{i_B \leq n}\text{in}(c_4[i_B], xpk_A : \text{pkey}); \text{new } N : \text{nonce}; \\
\text{out}(c_5[i_B], (N, B)); \text{in}(c_6[i_B], s : \text{signature}); \\
\text{if } \text{verify}(\text{concat}(xpk_A, B, N), xpk_A, s) \text{ then } \\
\text{if } xpk_A = pk_A \text{ then event } e_B(xpk_A, B, N) 
\]
Arrays

All variables defined under replications are implicitly arrays. This allows us to store all values that occur during the executions of the game. This replaces lists used by cryptographers and is key to cryptographic proofs.

\[
\begin{align*}
\text{in}(c_0, ()); & \quad \textbf{new} \ rk_A : \text{keyseed}; \\
\text{let} \ pk_A = \text{pkgen}(rk_A) \ \textbf{in} & \ \textbf{let} \ sk_A = \text{skgen}(rk_A) \ \textbf{in} \ \textbf{out}(c_1, pk_A); \\
!i_A^{\leq n} \ \text{in}(c_2[i_A], (xN[i_A] : \text{nonce}, xB[i_A] : \text{host})); \\
\textbf{event} \ e_A(pk_A, xB[i_A], xN[i_A]); & \quad \textbf{new} \ r[i_A] : \text{seed}; \\
\textbf{out}(c_3[i_A], \text{sign}(\text{concat}(pk_A, xB[i_A], xN[i_A]), sk_A, r[i_A])); \\
| \ !i_B^{\leq n} \ \text{in}(c_4[i_B], xpk_A[i_B] : \text{pkey}); & \quad \textbf{new} \ N[i_B] : \text{nonce}; \\
\textbf{out}(c_5[i_B], (N[i_B], B)); & \ \textbf{in}(c_6[i_B], s[i_B] : \text{signature}); \\
\textbf{if} \ \text{verify}(\text{concat}(xpk_A[i_B], B, N[i_B]), xpk_A[i_B], s[i_B]) \ \textbf{then} \\
\textbf{if} \ xpk_A[i_B] = pk_A \ \textbf{then} \ \textbf{event} \ e_B(xpk_A[i_B], B, N[i_B])
\end{align*}
\]
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:

\[
\text{in}(c0, ()); \textbf{new } rk_A : \text{keyseed}; \textbf{let } pk_A = pkgen'(rk_A) \textbf{ in } \text{out}(c1, pk_A); \]
\[
!^{i_A \leq n}\text{in}(c2[i_A], (xN : nonce, xB : host));
\]
\[
\textbf{event } e_A(pk_A, xB, xN); \textbf{let } m = \text{concat}(pk_A, xB, xN) \textbf{ in }
\]
\[
\text{new } r : \text{seed}; \text{out}(c3[i_A], \text{sign}'(m, skgen'(rk_A), r))
\]
\[
| \text{!}^{i_B \leq n}\text{in}(c4[i_B], xpk_A : pkey); \textbf{new } N : \text{nonce}; \text{out}(c5[i_B], (N, B));
\]
\[
\text{in}(c6[i_B], s : \text{signature}); \textbf{find } u \leq n \textbf{ such that }
\]
\[
\text{defined}(m[u], xB[u], xN[u]) \land (xpk_A = pk_A) \land (B = xB[u])
\]
\[
\land (N = xN[u]) \land \text{verify}'(\text{concat}(xpk_A, B, N), xpk_A, s) \textbf{ then }
\]
\[
\textbf{event } e_B(xpk_A, B, N)
\]
A **non-injective correspondence** is a formula of the form $\psi \Rightarrow \phi$ where

\[
\phi ::= \begin{array}{l}
M \\
\text{event}(e(M_1, \ldots, M_m)) \\
\phi_1 \land \phi_2 \\
\phi_1 \lor \phi_2
\end{array}
\]

and $\psi$ is a formula that contains only events and conjunctions.

**Example**

\[
\text{event}(e_B(x, y, z)) \Rightarrow \text{event}(e_A(x, y, z))
\]

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).
Let $\rho$ 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 $\rho$

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

**Definition**

$\mathcal{E} \vdash \psi \Rightarrow \phi$ if and only if for all $\rho$ defined on $\text{var}(\psi)$ such that $\rho, \mathcal{E} \vdash \psi$, there exists an extension $\rho'$ of $\rho$ to $\text{var}(\phi)$ 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.
An **injective correspondence** also allows injective events
\( \text{inj-event}(e(M_1, \ldots, M_m)) \).

Each execution of the injective events in \( \psi \) corresponds to **distinct**
injective events in \( \phi \).

**Example**

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

means that each execution of \( e_B(x, y, z) \) corresponds to a distinct
execution of \( e_A(x, y, z) \).
Intuition for the proof: non-injective correspondences (1)

Prove the correspondence $\text{event}(e_B(x, y, z)) \Rightarrow \text{event}(e_A(x, y, z))$ in the game

$\ldots !^{i_A \leq n} \ldots \text{event } e_A(pk_A, xB, xN); \text{let } m = \ldots \text{ in } \ldots$

$| !^{i_B \leq n} \ldots \text{find } u \leq n \text{ such that } \text{defined}(m[u], xB[u], xN[u]) \land$

$(xpk_A = pk_A) \land (B = xB[u]) \land (N = xN[u]) \land$

$\text{verify}'(\text{concat}(xpk_A, B, N), xpk_A, s) \text{ then } \text{event } e_B(xpk_A, B, N)$
Intuition for the proof: non-injective correspondences (1)

Prove the correspondence $\text{event}(e_B(x, y, z)) \Rightarrow \text{event}(e_A(x, y, z))$
in the game

\[
\ldots i_A \leq n \ldots \text{event } e_A(pk_A, xB, xN); \text{let } m = \ldots \text{ in } \ldots
\]

\[
| \ldots i_B \leq n \ldots \text{find } u \leq n \text{ such that } \text{defined}(m[u], xB[u], xN[u]) \land
\]

\[
(xpk_A = pk_A) \land (B = xB[u]) \land (N = xN[u]) \land
\]

\[
\text{verify}'(\text{concat}(xp_k_A, B, N), xp_k_A, s) \text{ then event } e_B(xp_k_A, B, N)
\]

If $\text{event}(e_B(x, y, z))$ has been executed, the program point

$\text{event } e_B(xp_k_A, B, N)$

has been reached for some $i_B = i'_B$, and

$e_B(x, y, z) = e_B(xp_k_A[i'_B], B, N[i'_B])$.

So $m[u[i'_B]]$, $xB[u[i'_B]]$, and $xN[u[i'_B]]$ are defined,

$xp_k_A[i'_B] = pk_A$, $B = xB[u[i'_B]]$, and $N[i'_B] = xN[u[i'_B]]$.

Since $m[u[i'_B]]$ is defined, the definition of $m[i_A]$ has been executed
for $i_A = u[i'_B]$, so $\text{event } e_A(pk_A, xB[i_A], xN[i_A])$ has been executed.
If \( \text{event}(e_B(x, y, z)) \) has been executed, the program point 
\textbf{event} \( e_B(xpk_A, B, N) \) has been reached for some \( i_B = i'_B \), and 
\( e_B(x, y, z) = e_B(xpk_A[i'_B], B, N[i'_B]) \).

So \( m[u[i'_B]], x_B[u[i'_B]], \) and \( xN[u[i'_B]] \) are defined, 
\( xpk_A[i'_B] = pk_A, \) \( B = xB[u[i'_B]] \), and \( N[i'_B] = xN[u[i'_B]] \).

Since \( m[u[i'_B]] \) is defined, the definition of \( m[i_A] \) has been executed 
for \( i_A = u[i'_B] \), so \textbf{event} \( e_A(pk_A, xB[i_A], xN[i_A]) \) has been executed.

We have

- \( x = xpk_A[i'_B] = pk_A \)
- \( y = B = xB[u[i'_B]] = xB[i_A] \)
- \( z = N[i'_B] = xN[u[i'_B]] = xN[i_A] \)

so \( e_A(pk_A, xB[i_A], xN[i_A]) = e_A(x, y, z) \) has been executed.
Intuition for the proof: injective correspondences (1)

Prove the correspondence

\[ \text{inj-event}(e_B(x, y, z)) \Rightarrow \text{inj-event}(e_A(x, y, z)) \] in the game

... \( \text{!}^i_A \leq n \) ... \text{event} e_A(pk_A, xB, xN); let \( m = \ldots \) in ...  
\[ \text{!}^i_B \leq n \) ... \text{find} u \leq n \text{ suchthat} \begin{aligned} \text{defined}(m[u], xB[u], xN[u]) \land \\
(xpk_A = pk_A) \land (B = xB[u]) \land (N = xN[u]) \land \\
\text{verify}'(\text{concat}(xpk_A, B, N), xpk_A, s) \text{ then event} e_B(xpk_A, B, N) \]
Intuition for the proof: injective correspondences (2)

Prove the correspondence

\[ \text{inj-event}(e_B(i, x, y, z)) \Rightarrow \text{inj-event}(e_A(i', x, y, z)) \] in the game

\[ \ldots \forall i_A \leq n \ldots \text{event} \ e_A(i_A, pk_A, xB, xN); \text{let} \ m = \ldots \text{in} \ldots \]

\[ \mid \forall i_B \leq n \ldots \text{find} \ u \leq n \text{ such that } \text{defined}(m[u], xB[u], xN[u]) \land \]

\[ (xpk_A = pk_A) \land (B = xB[u]) \land (N = xN[u]) \land \]

\[ \text{verify}'(\text{concat}(xpk_A, B, N), xpk_A, s) \text{ then event } e_B(i_B, xpk_A, B, N) \]

In order to record in which session each event is executed, we add replication indices to events.
Intuition for the proof: injective correspondences (3)

Prove the correspondence

\[ \text{inj-event}(e_B(i, x, y, z)) \Rightarrow \text{inj-event}(e_A(i', x, y, z)) \]

in the game

\[ \ldots !i_A \leq n \ldots \text{event } e_A(i_A, pk_A, xB, xN); \text{let } m = \ldots \text{ in } \ldots \]

\[ | !i_B \leq n \ldots \text{find } u \leq n \text{ such that defined}(m[u], xB[u], xN[u]) \land (xpk_A = pk_A) \land (B = xB[u]) \land (N = xN[u]) \land \text{verify}'(concat(xpk_A, B, N), xpk_A, s) \text{ then event } e_B(i_B, xpk_A, B, N) \]

If \( \text{event}(e_B(i, x, y, z)) \) has been executed, the program point

\( \text{event } e_B(i_B, xpk_A, B, N) \) has been reached for some \( i_B = i_B' \), and

\( e_B(i, x, y, z) = e_B(i_B', xpk_A[i_B'], B, N[i_B']). \)

So \( m[u[i_B']], xB[u[i_B']], \text{ and } xN[u[i_B']] \) are defined,

\( xpk_A[i_B'] = pk_A, B = xB[u[i_B']], \text{ and } N[i_B'] = xN[u[i_B']]. \)

Since \( m[u[i_B']] \) is defined, the definition of \( m[i_A] \) has been executed for \( i_A = u[i_B'] \), so \( \text{event } e_A(i_A, pk_A, xB[i_A], xN[i_A]) \) has been executed.
As before, $e_A(i_A, pk_A, xB[i_A], xN[i_A]) = e_A(i', x, y, z)$ for some $i'$.

In order to show injectivity, we show that $e_B$ executed twice, for $i_B = i'_B$ and $i_B = i''_B$, with $i'_B \neq i''_B$.

$\Rightarrow e_A$ executed twice, for $i_A = u[i'_B]$ and $i_A = u[i''_B]$, with $u[i'_B] \neq u[i''_B]$.

By contraposition, we show $u[i'_B] = u[i''_B] \Rightarrow i'_B = i''_B$.

$u[i'_B] = u[i''_B] \Rightarrow xN[u[i'_B]] = xN[u[i''_B]]$

$\Rightarrow N[i'_B] = N[i''_B]$ since $xN[u[i'_B]] = N[i'_B]$ and $xN[u[i''_B]] = N[i''_B]$

$\Rightarrow i'_B = i''_B$ up to negligible probability by eliminating collisions.
Proof technique

In order to prove $\psi \Rightarrow \phi$, two main steps:

1. Collect the facts that hold when the events in $\psi$ are executed.
2. Reason on these facts using an equational prover in order to show that the events in $\phi$ have been executed (and show injectivity when needed).

We shall now detail these points.
Collecting true facts

For each program point $P$, we collect a set of true facts at that point $F_P$.

- We take into account assignments and tests above $P$.

**Example**

In $\textbf{if } M \textbf{ then } P$, $M \in F_P$.

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

**Example**

If $\textbf{defined}(x[M]) \in F_P$ and $M$ holds at all definitions of $x[\tilde{i}]$, then $M\{\tilde{M}/i\} \in F_P$.

- We take into account that code is always executed up to the next output before switching to another thread.
Collecting true facts: example (1)

\[
\begin{align*}
\text{in}(c0, ()); \textbf{new } &rk_A : \text{keyseed}; \textbf{let } pk_A = \text{pkgen}'(rk_A) \textbf{ in out}(c1, pk_A); \\
!^{i_A \leq n}_1 &\text{in}(c2[i_A], (xN : nonce, xB : host)); \\
\textbf{event } &e_A(pk_A, xB, xN); \textbf{let } m = \text{concat}(pk_A, xB, xN) \textbf{ in} \\
\textbf{new } &r : \text{seed}; \textbf{out}(c3[i_A], \text{sign}'(m, \text{skgen}'(rk_A), r)) \\
| &!^{i_B \leq n}_2 \text{in}(c4[i_B], xpk_A : pkey); \textbf{new } N : \text{nonce}; \textbf{out}(c5[i_B], (N, B)) \\
\textbf{in}(c6[i_B], s : \text{signature}); \textbf{find } u \leq n \textbf{ suchthat} \\
&\text{defined}(m[u], xB[u], xN[u]) \land (xpk_A = pk_A) \land (B = xB[u]) \\
&\land (N = xN[u]) \land \text{verify}'(\text{concat}(xpk_A, B, N), xpk_A, s) \textbf{ then} \\
\textbf{event } &e_B(xpk_A, B, N)
\end{align*}
\]

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

\[
\mathcal{F}_P = \{ \text{defined}(m[u[i_B]]), \text{defined}(xB[u[i_B]]), \text{defined}(xN[u[i_B]]), \\
xpk_A[i_B] = pk_A, B = xB[u[i_B]], N[i_B] = xN[u[i_B]], \ldots \}
\]
Collecting true facts: example (2)

\texttt{in}(c0, ()); \texttt{new} \: \texttt{rk}_A : \texttt{keyseed}; \texttt{let} \: \texttt{pk}_A = \texttt{pkgen}^\prime(\texttt{rk}_A) \: \texttt{in} \: \texttt{out}(c1, \texttt{pk}_A);
\!^{i_A \leq n}\texttt{in}(c2[i_A], (xN : nonce, xB : host));
\texttt{event} \: e_A(\texttt{pk}_A, xB, xN); \texttt{let} \: m = \texttt{concat}(\texttt{pk}_A, xB, xN) \: \texttt{in}
\texttt{new} \: r : \texttt{seed}; \texttt{out}(c3[i_A], \texttt{sign}^\prime(m, \texttt{skgen}^\prime(\texttt{rk}_A), r))

| \ldots |

At program point \( P = \texttt{event} \: e_B(\texttt{xpk}_A, B, N), \)

\( \mathcal{F}_P = \{ \texttt{defined}(m[u[i_B]]), \texttt{defined}(xB[u[i_B]]), \texttt{defined}(xN[u[i_B]]), \)
\( xpk_A[i_B] = \texttt{pk}_A, B = xB[u[i_B]], N[i_B] = xN[u[i_B]], \)
\( \texttt{event}(e_A(\texttt{pk}_A, xB[u[i_B]], xN[u[i_B]])), \ldots \} \)

because \texttt{defined}(m[u[i_B]]) \in \mathcal{F}_P.
We use an algorithm inspired by the Knuth-Bendix completion algorithm to derive new equalities from known equalities.

The equational prover also eliminates collisions when they have negligible probability.

Details of this prover can be found in the paper: Blanchet, A Computationally Sound Mechanized Prover for Security Protocols, TDSC, 2008.

We say that $\mathcal{F}$ yields a contradiction when the prover starting from $\mathcal{F}$ derives $false$. 
Proof of non-injective correspondences (1)

Let \( \psi \Rightarrow \phi = F_1 \land \ldots \land F_m \Rightarrow \phi \) 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 \( \theta_j' \) renames the replication indices at \( P_j \) to fresh replication indices.

**Example**

\[
\text{event} \ e_B(xpk_A, B, N) \text{ has been executed, with } \\]
\[
event(e_B(x, y, z)) = event(e_B(xpk_A[i'_B], B, N[i'_B])), \quad \theta' = \{i'_B/i_B\}.
\]
Proof of non-injective correspondences (2)

Then the facts \( F_j = \theta'_j F_{P_j} \cup \{ \theta'_j M'_{j1} = M_{j1}, \ldots, \theta'_j M'_{jm_j} = M_{jm_j} \} \) hold.

Example

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

For each such \( P_1, \ldots, P_m \), we show that

\[ F = F_1 \cup \ldots \cup F_m \text{ implies } \theta \phi \]

for some \( \theta \) equal to the identity on \( \text{var}(\psi) \), by the equational prover.

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

Example

\[ x = xpk_A[i'_B], y = B, z = N[i'_B], xpk_A[i'_B] = pk_A, B = xB[u[i'_B]], N[i'_B] = xN[u[i'_B]], \text{ event}(e_A(pk_A, xB[u[i'_B]], xN[u[i'_B]])) \]

imply \( \text{event}(e_A(x, y, z)) \).