MPRI 2.30, part I

Formal Proofs of Cryptographic Protocols

David Baelde

November 29, 2017

The exam consists of three independent exercises. You have three hours.
All documents are allowed, but electronic devices are forbidden.

1 First-order definitions of trace equivalence

We consider a simple process calculus that is a restriction of the one(s) seen in the lectures. We take only input and output prefixes and parallel composition. As usual, we identify processes up to associativity and commutativity of parallel composition.

The calculus relies on a signature Σ and an equational theory $=_{\mathsf{E}}$ over terms of $\mathcal{T}(\Sigma, \mathcal{X} \cup \mathcal{N})$ which is supposed to be convergent: there is a normal form operator $\downarrow$ such that for all terms u and v, $u =_{\mathsf{E}} v$ iff $u\downarrow = v\downarrow$.

Transition systems We define a second-order LTS (with recipes) $\rightarrow_2$ by the following two rules:

$$(P \mid \mathsf{in}(c,x).Q, \Phi) \xrightarrow{\mathsf{in}(c,R)}_2 (P \mid Q[x := R\Phi\downarrow], \Phi) \qquad \text{where } R \in \mathcal{T}(\Sigma, \mathsf{dom}(\Phi))$$

$$(P \mid \mathsf{out}(c,t).Q, \Phi) \xrightarrow{\mathsf{out}(c,w)}_2 (P \mid Q, \Phi + \{w \mapsto t\}) \qquad \text{where } w \notin \mathsf{dom}(\Phi)$$

We also define a first-order LTS which is an hybrid of the two styles seen in the lectures: input actions contain the message rather than the recipe, but output actions and frames make use of handles. It is given by the following two rules:

$$(P \mid \mathsf{in}(c,x).Q, \Phi) \xrightarrow{\mathsf{in}(c,t)}_1 (P \mid Q[x := t], \Phi) \qquad \text{if } t = R\Phi\downarrow \text{ for some } R \in \mathcal{T}(\Sigma, \mathsf{dom}(\Phi))$$

$$(P \mid \mathsf{out}(c,t).Q, \Phi) \xrightarrow{\mathsf{out}(c,w)}_1 (P \mid Q, \Phi + \{w \mapsto t\}) \qquad \text{where } w \notin \mathsf{dom}(\Phi)$$

For example, considering the configuration $A = (\mathsf{in}(c,x).\mathsf{out}(c,x), \emptyset)$ and a term $t \in \mathcal{T}(\Sigma, \emptyset)$ such that $t = t\downarrow$, there is only one transition in the first-order LTS from A to $(\mathsf{out}(c,t), \emptyset)$ (with label $\mathsf{in}(c,t)$) while there might be several transitions in the second-order LTS (with labels $\mathsf{in}(c,R)$ for all R such that $R\downarrow = t$).

Correspondences We say that a second-order transition $A \xrightarrow{\alpha}_2 A'$ and a first-order transition $A \xrightarrow{\beta}_1 A'$ correspond when (they have the same source and target configurations and):

- either $\alpha = \beta = \mathsf{out}(c,w)$ and the transitions are identical, selecting the same sub-process in A to perform the output;
- or $\alpha = \mathsf{in}(c,R)$ and $\beta = \mathsf{in}(c,t)$ where $t = R\Phi\downarrow$ and the transitions are otherwise identical, selecting the same sub-process in A to perform the input.

The notion is lifted to traces in a natural way:

- $A \xrightarrow{\epsilon}_2 A$ corresponds to $A \xrightarrow{\epsilon}_1 A$ (empty executions correspond);
- $A \xrightarrow{\alpha}_2 A'' \xrightarrow{\mathsf{ta}'}_2 A'$ and $A \xrightarrow{\beta}_1 A'' \xrightarrow{\mathsf{tb}'}_2 A'$ correspond when the first steps correspond and the rest of the executions correspond.

With these definitions, we have that any second-order execution corresponds to exactly one first-order execution. However, a first-order execution will generally correspond to several second-order executions.

Notions of trace inclusion We define three notions of trace inclusion. The first one is standard, based on the second-order LTS. The next two are based in part on the first-order LTS.

$$A \sqsubseteq^2 B \text{ iff } \begin{cases} \text{for all } \mathsf{tr} \text{ and } A' \text{ such that } A \xrightarrow{\mathsf{tr}}_2 A', \\ \text{there exist } B' \text{ such that } B \xrightarrow{\mathsf{tr}}_2 B' \text{ and } \Phi(A') \sim \Phi(B'). \end{cases}$$

$$A \sqsubseteq^\exists B \text{ iff } \begin{cases} \text{for all } \mathsf{tr} \text{ and } A' \text{ such that } A \xrightarrow{\mathsf{tr1}}_1 A', \\ \text{there exists a corresponding execution } A \xrightarrow{\mathsf{tr2}}_2 A', \\ \text{there exist } B' \text{ such that } B \xrightarrow{\mathsf{tr2}}_2 B' \text{ and } \Phi(A') \sim \Phi(B'). \end{cases}$$

$$A \sqsubseteq^\forall B \text{ iff } \begin{cases} \text{for all } \mathsf{tr} \text{ and } A' \text{ such that } A \xrightarrow{\mathsf{tr1}}_1 A', \\ \text{for all corresponding execution } A \xrightarrow{\mathsf{tr2}}_2 A', \\ \text{there exist } B' \text{ such that } B \xrightarrow{\mathsf{tr2}}_2 B' \text{ and } \Phi(A') \sim \Phi(B'). \end{cases}$$

Question 1 Show that $A \sqsubseteq^\forall B$ implies $A \sqsubseteq^2 B$.

Question 2 Show that $A \sqsubseteq^2 B$ implies $A \sqsubseteq^\exists B$.

Question 3 Show that $A \sqsubseteq^\exists B$ implies $A \sqsubseteq^\forall B$.

2 Resolution with selection

We consider a Proverif model containing only the following primitives:

```
fun ok:bitstring.
fun senc(bitstring,bitstring):bitstring.
reduc forall x:bitstring, k:bitstring;
  sdec(senc(x,k),k) = x.
```

We assume a public channel c and private bitstrings n and k, and take the following process as the system under study:

```
in(c,x:bitstring);
let y:bitstring = sdec(x,k) in
out(c,senc(n,k))
```

Question 1 Give the clauses generated by Proverif for the primitives and for the system. Since we only use a public channel c, you can (and must) ignore the $\mathsf{mess}(\cdot,\cdot)$ predicate and the associated attacker clauses: your generated clauses should only mention the $\mathsf{att}(\cdot)$ predicate. Please give names (or numbers) to your clauses.

Question 2 Saturate the previous set of clauses by resolution with selection, assuming that the selection function selects, when possible, exactly one hypothesis of the form $\mathsf{att}(t)$ where t is not a variable, and selects no hypothesis otherwise. You should not use any optimization involving subsumption, but you can drop clauses that have one of their hypotheses as conclusion. Please give names to the generated clauses, indicate from which clauses they have been obtained, and underline selected hypotheses.

Question 3 Considering the set of solved clauses of the saturated set previously computed, what would Proverif conclude regarding the secrecy of n, k, and $\mathsf{senc}(n, k)$?

Question 4 Describe what happens if the selection function never select any hypothesis: describe the shape of solved clauses, the result of saturation, and discuss its usefulness with respect to Proverif's procedure for semi-deciding secrecy. In particular, illustrate what would happen on the above example: what would the saturation do, what would be the result of the three secrecy queries?

Question 5 Describe what happens if the selection function selects all hypotheses in all clauses: describe the shape of solved clauses, the set of solved clauses of a saturated set of clauses, and discuss its usefulness with respect to Proverif's procedure for secrecy. In particular, illustrate what would happen on the above example: what would the saturation do, what would be the result of the three secrecy queries?

3 Blind tokens

We consider a protocol where users (U) may obtain some tokens from an authority (A) and use them to access a service (S). We do not detail how the users may get their tokens: they might have to pay, prove that they belong to a group, etc. but simply model the protocol as follows:

$$\begin{array}{lll}
1. & U \to A: & \mathsf{blind}(n, r) \\
2. & A \to U: & \mathsf{sign}(\mathsf{blind}(n, r), k) \\
3. & U \to S: & \langle n, \mathsf{sign}(n, k)\rangle \\
4. & S \to U: & \mathsf{ok}
\end{array}$$

Here k is a private signing key that only the authority has. The associated public signing key $\mathsf{spk}(k)$ is known by all agents. The nonces n and r are generated by the user at the beginning of each session. At step (2) the user checks that the message he receives is a signature of the message he sent at step (1), otherwise he aborts the interaction. At step (3) the service should check that the second component of the tuple is a signature of the first component.

In order to model this protocol in the applied pi-calculus, we consider a signature Σ consisting of the binary function symbols $\langle \cdot, \cdot \rangle$, sign, check, blind and $\mathsf{unblind}$, the unary function symbols spk, fst and snd, and the constant symbol ok. We take the equational theory generated by the following equations:

$$\mathsf{snd}(\langle x, y\rangle) = y \qquad \mathsf{fst}(\langle x, y\rangle) = x$$

$$\mathsf{unblind}(\mathsf{sign}(\mathsf{blind}(m, r), k), r) = \mathsf{sign}(m, k) \qquad \mathsf{check}(\mathsf{sign}(m, k), \mathsf{spk}(k)) = m$$

In Proverif's terminology, all symbols would be constructors and all messages would be of type `bitstring`.

Question 1 Give applied pi-calculus processes A, S and U that respectively model the authority, service and user's roles. The user's role should consist of its interaction with the authority followed by its interaction with the service. Moreover:

- The processes should only describe a single interaction, which should not require any replication construct.

- The processes should have no free variable and only k as a free name. Moreover, k should only occur as part of $\mathsf{spk}(k)$ in S and U.
- Inputs and outputs performed by the authority (resp. the service) should be on channel c_A (resp. c_S). The user should use channel c_U^A when interacting with A and c_U^S when interacting with S. Thus, a message sent by U to A will be outputted on c_U^A and (if the environment/attacker decides so) forwarded to be inputted on c_A.

Question 2 We consider a scenario describing a single session of the protocol:

$$\mathsf{new}\ k.\mathsf{out}(c_A, \mathsf{spk}(k)).\big(U \mid A \mid S\big)$$

The intent of our protocol is that the service may only be used with a token that has been previously issued by the authority[1]: in any execution trace of our scenario, S should only be able to receive a message $\langle x, \mathsf{sign}(x,k)\rangle$ (for some x) if $\mathsf{sign}(\mathsf{blind}(x,y),k)$ (for some y) has previously been output by U. Using Proverif's notation for correspondences, we expect the following property of our traces[2]:

$$\mathsf{query}\ x : \mathsf{bitstring}, y : \mathsf{bitstring};\ ES_3(\langle x, \mathsf{sign}(x,k)\rangle) ==> EA_2(\mathsf{sign}(\mathsf{blind}(x,y),k))$$

where the event ES_3 would be emitted just after the input of S at step (3) and EA_2 would be just before the output of A at (2). Show that this security property is not satisfied by our protocol: exhibit a trace that does not satisfy the correspondence. The trace should be given using a second-order semantics (with recipes) and the resulting frame should be given explicitly. Invisible actions (τ) can be omitted.

Question 3 We avoid the attack identified in the previous question by using symmetric encryption in the first two exchanges. For simplicity we assume that the authority shares a secret key k' with all users — in other words, there is a single user. The protocol is changed as shown below:

$$\begin{array}{lll}
1. & U \to A : & \mathsf{senc}(\mathsf{blind}(n,r),k') \\
2. & A \to U : & \mathsf{senc}(\mathsf{sign}(\mathsf{blind}(n,r),k),k') \\
3. & U \to S : & \langle n, \mathsf{sign}(n,k)\rangle \\
4. & S \to U : & \mathsf{ok}
\end{array}$$

We assume that the processes from the first question have been updated to reflect this change of the protocol, in a theory enriched with sdec, senc and the equation $\mathsf{sdec}(\mathsf{senc}(x,y),y) = x$. The scenario under study is simply updated to $\mathsf{new}\ k.\mathsf{new}\ k'.\mathsf{out}(c_A, \mathsf{spk}(k)).\big(U \mid A \mid S\big)$. The attack identified before should not be possible anymore. However, we will see that problems re-appear when we consider additional algebraic properties that our primitives may satisfy. Specifically, some RSA-based blind signature scheme would satisfy the following two equations, involving the binary symbol $\times$:

$$\begin{array}{rcl}
\mathsf{sign}(x_1,y) \times \mathsf{sign}(x_2,y) & = & \mathsf{sign}(x_1 \times x_2, y) \\
\mathsf{blind}(x_1,y_1) \times \mathsf{blind}(x_2,y_2) & = & \mathsf{blind}(x_1 \times x_2, y_1 \times y_2)
\end{array}$$

Show that the previous correspondence property is still broken with this variant of the protocol.

[1]We would also like that the service can only be used once with a given token. That would be enforced in a straightforward manner by keeping a table of spent tokens. We do not care to model this, and can analyze other security properties independently.

[2]This Proverif query would not be possible with k bound in the scenario under consideration, but we would declare k as a private name.

Question 4 We turn to another fix, forgetting about encryption but using hashing instead:

$$\begin{array}{lll} 1. & U \to A: & \mathsf{blind}(\mathsf{h}(n), r) \\ 2. & A \to U: & \mathsf{sign}(\mathsf{blind}(\mathsf{h}(n), r), k) \\ 3. & U \to S: & \langle n, \mathsf{sign}(\mathsf{h}(n), k)\rangle \\ 4. & S \to U: & \mathsf{ok} \end{array}$$

We admit that the protocol now satisfies the desired property, modified to take the hash into account: S may only receive $\langle x, \mathsf{sign}(\mathsf{h}(x), k)\rangle$ if A has previously outputted $\mathsf{sign}(\mathsf{blind}(\mathsf{h}(x), y), k)$.

We now wonder whether this version of the protocol could satisfy a form of unlinkability, even though all exchanges are made in clear. In fact, this is an opportunity to verify whether the protocol ensures unlinkability of the user not only against an outside attacker, but also against the authority and service operators. To put it differently, we are investigating whether unlinkability is maintained when these agents are compromised.

In order to define the unlinkability property, we split the process U into the part interacting with A and the part interacting with S. The first part, U_A, is a closed process. The second part is a process $U_S(n, t)$ parameterized by the nonce and the signed token obtained in the first part. In the end we intuitively[3] have $U = U_A; U_S(n, \mathsf{unblind}(x, r))$ if n is the nonce introduced in U_A and x is the last input variable of U_A. We finally define $U'_S := \mathsf{new}\ n.\ U_S(n, \mathsf{sign}(\mathsf{h}(n), k))$.

We consider four security properties defined as trace equivalences, given below. In each case there is no setup phase: instead k should be considered as a public constant of Σ (representing the fact that the authority has been compromised).

$$\begin{aligned} !U &\approx !U_A && (1) \\ !U &\approx !U_A \mid !U'_S && (2) \\ !U \mid !U'_S &\approx !U_A \mid !U'_S && (3) \\ !U \mid !U'_S \mid !A \mid !S &\approx !\ U_A \mid !U'_S \mid !A \mid !S && (4) \end{aligned}$$

(4.a) For each of these equivalences, indicate if it does not hold by describing a distinguishing trace. If you believe that it holds, indicate whether it suitably models unlinkability.

(4.b) What would change in the previous answer if the user did not check that the message he receives at step (2) is indeed a signature of what he sent just before?

[3]This is only an intuition since the applied pi-calculus does not have a sequential composition operator $P; Q$ on processes, and the use of sequential composition made here behaves funnily with respect to binders.