Practical session: the ProVerif tool

Getting started. The first step is to install the ProVerif tool. ProVerif is freely available for download at:

`http://proverif.inria.fr`

Download the source package and follow the installation instructions in the `README` file. ProVerif also comes with an extensive documentation that can be obtained by downloading the documentation package.

The Needham Schroeder public key protocol.

$$\begin{aligned} A &\to B: \ \{A, N_a\}_{\mathsf{pk}(B)} \\ B &\to A: \ \{N_a, N_b\}_{\mathsf{pk}(A)} \\ A &\to B: \ \{N_b\}_{\mathsf{pk}(B)} \end{aligned}$$

Description. Alice starts the protocol by sending her identity A together with a freshly generated random number N_a. This message is encrypted using an asymmetric encryption algorithm with Bs public key (denoted $\mathsf{pk}(B)$). We suppose that only agent Bob (whose identity is B) knows the secret key corresponding to $\mathsf{pk}(B)$. Next Bob receives the message $\{A, N_a\}_{\mathsf{pk}(B)}$ sent by Alice. Using his private key, Bob decrypts the message. He sends the received nonce N_a together with a freshly generated nonce N_b encrypted with As public key ($\mathsf{pk}(A)$) to Alice. Finally Alice receives the message $\{N_a, N_b\}_{\mathsf{pk}(A)}$. She decrypts the message and checks that the nonce N_a corresponds to the nonce previously generated and sent to Bob. She sends the nonce N_b to Bob encrypted with Bobs public key. Upon reception of this message Bob decrypts it and checks that the nonce corresponds to the one previously generated.

Security properties. The protocol is supposed to achieve mutual authentication through the secrecy of the nonces N_a and N_b that are exchanged during an execution of this protocol.

The file `nspk.pv` contains a partial modelling of the Needham Schroeder protocol. It is modelled in a *typed* variant of the process calculus seen during the talk.

1. Fill in the missing responder process. Once the process `Pr (sk:skey)` has been defined, the command `proverif nspk.pv` should display the process.

   As a *sanity check* verify that the process is executable. This can be done by declaring an event `event reach.`, annotating the process with the instruction `event reach` and adding the query `query event(reach).` If the event is reachable, executing `proverif -in pitype nspk.pv` should provide the output

   ```
   goal reachable: end(reach)
   RESULT not event(reach) is false.
   ```

To see a graph representing the "attack trace", you can create a repository `REP` and execute `proverif -html REP -in pitype nspk.pv`. Proverif results will be available in the `REP` directory. Have a look to `Trace graph`.

2. The main process is a *naïve* encoding of the Needham Schroeder public key protocol allowing only one session to be executed between honest parties a and b. Check that in this simplified model the nonce na remains secret by adding the query `query attacker(new na).` and run ProVerif. Think about why the nonce nb is not secret in the given process.

3. Change the main process to include multiple sessions between 2 honest agents (a and b) and a dishonest one, namely c.

4. Verify secrecy of the nonce N_a generated by the initiator: when the initiator role executed by an honest agent ends a session apparently with another honest agent, then the nonce that has been generated by the initiator during this session remains unknown by the attacker. To encode this, you can rely on symmetric encryption and a private constant `secret`. Both have to be declared in the preamble:

```
free secret:bitstring [private].

fun senc(bitstring,bitstring):bitstring.
fun sdec(bitstring,bitstring):bitstring.
equation forall x:bitstring, y:bitstring; sdec(senc(x,y),y) = x.
```

   Then, you can emit (when needed) this private constant `secret` encrypted with N_a at the end of the initiator role.

5. Show that the same property is violated for the responder for the nonce N_b. Look at ProVerif's output (e.g. the graphical output) and explain the attack found.

Correcting the Needham Schroeder public key protocol. Lowe proposed a simple fix to the Needham-Schroeder protocol: add the responder's identity to the second message, i.e. replace message $\{N_a, N_b\}_{\mathsf{pk}(A)}$ by $\{N_a, N_b, B\}_{\mathsf{pk}(A)}$. This protocol is known as the Needham Schroeder Lowe public key protocol.

1. Update the model to reflect this fix. Verify that the modified protocol is executable.

2. Verify that the secrecy of N_a and N_b is satisfied.