**Practical session: the ProVerif and deepsec tools**

# 1 ProVerif

## The Needham Schroeder Lowe public key protocol (continued)

Recall the Needham Schroeder Lowe protocol:

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

It fixes the attack of the original Needham Schroeder protocol by adding the identity of B in the second message.

1. Starting from the truncated file of the NS protocol (provided on Monday) verify that the protocol satisfies *real-or-random secrecy* of nonce $N_b$.

   As an alternative to the choice operator *real-or-random secrecy* can be checked using the `weaksecret` query:

   ```
   free nb: bitstring [private].
   weaksecret nb.
   ```

   You probably find an attack! Explain what went wrong and fix the model.

2. Does the NSL protocol verify *strong secrecy* as well?

   As an alternative to the choice operator *real-or-random secrecy* can be checked using the `weaksecret` query:

   ```
   free nb: bitstring [private].
   noninterf nb.
   ```

# 2 deepsec

## The PAP protocol

Consider the following simplified version of Abadi and Fournet's private authentication protocol:

$$\begin{array}{llll} A & \to B: & \{N_a, \mathsf{pk}(A)\}_{\mathsf{pk}(B)} & \\ A & \to B: & \{N_a, N_b, \mathsf{pk}(B)\}_{\mathsf{pk}(A)} & \text{if } B \text{ accepts requests from } A \\ & & \{N_b\}_{\mathsf{pk}(B)} & \text{otherwise} \end{array}$$

In this protocol $B$ may accept authentication requests from some party, but not from others. The goal is to hide from the adversary whether requests from an (uncompromised) agent $A$ are accepted or not. (In case requests from $A$ are not accepted a decoy message $\{N_b\}_{\mathsf{pk}(B)}$ is returned.)

1. Model this protocol, the anonymity property and check that it holds. You may start from the `pap-truncated.txt` file.
2. What happens when the decoy message is removed?
3. What happens when the decoy message is encrypted with $A$'s public key instead of $B$'s (i.e. send $\{N_b\}_{\mathsf{pk}(A)}$ instead of $\{N_b\}_{\mathsf{pk}(B)}$).

## The Helios e-voting protocol

Consider the following simplified version of the Helios protocol.

- Voters submit through an authenticated channel their ballot $(id, \{v\}^r_{pkE})$ to a bulletin board where $pkE$ is the election public key, and $r$ are the random coins;
- Encrypted votes are sent through a decryption mix and the votes are published.

1. Starting from `helios-truncated.txt` file, model this simplified version of Helios for 3 voters: 2 honest and 1 dishonest voter.

   A few hints:

   - Authenticated channels are modelled by two sequential outputs, one on a secret (restricted with `new`) and one on a public channel:

     ```
     new sc;
     ...
     out(sc,t);
     out(c,t);
     ...
     ```

     (Note that the corresponding input on `sc` must be in the scope of `new sc`.
   - The mixnet (see `helios-truncated.txt`) can be modelled by sending the encrypted votes through a secret channel to the mixnet which outputs them in non-deterministic order (e.g. in parallel).

2. Vote privacy can be expressed by verifying equivalence of the scenarios where $A$ votes yes, $B$ votes no, and $A$ votes no, $B$ votes yes. Show that you can find the *vote copying* attack present in Helios.

3. The attack can be avoided by adding a zero knowledge proof witnessing that $A$ knows the randomness used for encryption (and including identity in the zero-knowledge proof). Give a simple rewrite system modelling this and update the protocol. Check that vote privacy is now guaranteed.