CryptoVerif – Practical Session 2

June 2023

1 The Signed DH Protocol

The goal of this session is to complete the *template-signedDH.ocv* file with a full modeling of the signed DH protocol, including secrecy and authentication properties that CryptoVerif should be able to prove automatically.

The model will be a "basic" version: we only model two fixed honest participants A and B. The adversary can instruct A and B to initiate an arbitrary polynomial number of sessions, and the adversary can control who A and B will try to talk to.

The template file already contains the main process declaration (the skeleton of the security game): it includes a process for A with fixed private key skA and a processB with fixed private key skB for B.

In addition, the template contains most of the preamble definitions (types, macro expansion for crypto functions, ...).

1. Define the contents of the `pkI` process. It shall expose an oracle that the adversary can use to register a pair of a hostname and a public key. The oracle shall fill a table with this information (also to be defined by you) which corresponds to the public key infrastructure: the other oracles can, given a hostname, retrieve the corresponding public key.

   In the template, the `pkI` process gets the two honest public keys as parameters. This is so that the oracle can insert the appropriate honest public key into the table for the honest hostnames A and B – whenever the adversary tries to register something for A or B, just use the honest public keys and ignore the key supplied by the adversary!

2. Define the processA corresponding to an initiator behavior. This means you shall define oracles for protocol message handling of the initiator side of the protocol. In the first oracle, the adversary shall choose the hostname of the peer (by giving it as a parameter to the oracle). The initiator shall always be A and use the private key `skA`. Some more hints:

   - Use sequential oracles, see the slides of the first part for a reminder.
   - When you need the signature public key of the peer, use `get` to retrieve it from the table that was filled by the oracle in the `pkI` process.
   - Define two message encoding functions, one for the data signed as part of the second message, and one for the third message. Observe that they take the same parameters, and output a term of type `message` – this is the type that `sign` and `verify` take as input. Also write an equation that states that the outputs of the two message encoding functions do not collide (there is some help at the end of the cheatsheet).

3. Define the processB similarly.

4. Add events into processA and processB for the authentication queries. Add the authentication queries to the preamble. Try to prove it automatically with CryptoVerif. Refer to the protocol diagram on the slides if you do not know where to add the events, and to the cheatsheet for how to write correspondence queries.

5. Within processA and processB, store within variables `keyA` and `keyB`, respectively, the final key if it is a key for an honest session between A and B. Add the corresponding secrecy queries to the preamble checking that all values stored within keyA and keyB are secret. Try to automatically prove it with CryptoVerif.

6. Extract from the CryptoVerif output the final probability bound for the secrecy of keyA. Try to roughly give meaning to each part of the sum, as well as their corresponding factor, w.r.t. to an "intuitive" security argument of the signed DH protocol.

7. Do a similar thing for the query checking that B is authenticated to A. Do you think the bound is optimal? If it is not, can you find a way to get a better bound?