In this exercise session, we will model some protocols and verify certain properties about them using ProVerif.

1 Key exchange

1.1 A simple key exchange protocol

We start from a simple key exchange protocol, mentioned in the previous lectures. The initial version of the protocol (let us denote it $P$) is given below.

Ensure that the following properties hold for your model of $P$:

1. The process of an honest user Alice runs successfully up to the end.
2. The process of an honest user Bob runs successfully up to the end.
3. The attacker is unable to read the secret message $M$.
4. Honest Alice and Bob will have a common agreement on the shared key $K_{AB}$.
5. The agreement on the key $K_{AB}$ is injective (i.e. a single session of Alice cannot be replayed in several session with Bob).
Will secrecy of $M$ be still maintained if the secret key of Alice/Bob leaks after the key exchange? Verify whether the following statements hold (Hint: use phases to ensure that the secret key leaks after the key exchange).

6. The attacker is unable to read $M$ if the secret key of Alice leaks after key exchange.

7. The attacker is unable to read $M$ if the secret key of Bob leaks after key exchange.

1.2 An enhanced key exchange protocol

We can use Diffie-Hellmann to enhance the previous protocol in such a way that the secrecy of $M$ is kept even if the secret key of Bob leaks afterwards. Recall that Diffie-Hellman key exchange works as follows:

- Alice generates an exponent $a$, and Bob generates an exponent $b$.
- Alice and Bob compute and exchange $g^a$ and $g^b$ respectively.
- Alice computes $K_{AB} = (g^b)^a$, and Bob computes $K_{AB} = (g^a)^b$. We have $(g^b)^a = (g^a)^b = g^{ab}$.

We can enhance our previous protocol treating $N_A$ and $N_B$ as $g^a$ and $g^b$, where $a$ and $b$ remain in the heads of Alice and Bob and do not leak even if their secret keys leak. The initial version of the protocol (let us denote it $P'$) is given below.

Ensure that the following properties hold for your model of $P'$:

8. The attacker is unable to read $M$ if the secret key of Alice leaks after key exchange.

9. The attacker is unable to read $M$ if the secret key of Bob leaks after key exchange.

10. The attacker is able to read $M$ if the secret key of Alice leaks before key exchange.
2 A simple voting protocol

Consider a simple voting protocol where Alice, Bob, and Chris vote whether there should be chicken or fish for lunch today. Assume that there is a trusted server to whom the parties are connected via private channels, which collects the votes of Alice, Bob and Chris, and chooses the winner. The election result is public. The attacker can in addition have access to a particular side channel information.

First of all, model the protocol that just outputs to the public network the final result, without any particular side channels. Try out the following queries, each of which represents a different way of defining voter privacy.

1. Check whether the attacker can read the vote of Alice. Temporarily add a side-channel that leaks hash of the vote of Alice. Does such a query seem reasonable?

2. Check whether the attacker is able to distinguish whether Alice has voted for fish or chicken (remove the side-channel that was established in the previous query).

3. In general, we cannot hide information that leaks through the final output of a voting protocol. Hence, a standard privacy definition for voting protocols in symbolic model says that the attacker should not be able to detect if two honest voters have swapped their votes. Model this check in ProVerif.

- Verify that the protocol is private without the side channel.
- Verify that the protocol is no longer private if we add the side channel that outputs hashes of votes.