# Home Assignment 1b: A Shared Key Protocol

**To hand in before or on November 2, 2010.**
**The penalty for delays is 2 points per day.**

October

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 18 | (19) | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |

November

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | (2) | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |

Electronic versions (PDF only) can be sent by email to ⟨schmitz@lsv.ens-cachan.fr⟩, paper versions should be handed in on the 2nd or put in my mailbox at LSV, ENS Cachan.

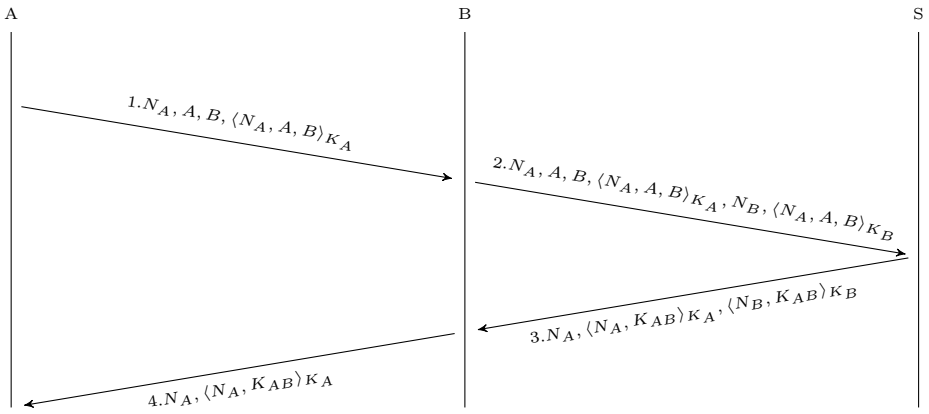
## 1 A Shared Key Protocol

The protocol relies on

- the generation of *nounces* $N_C$: random numbers that should only be used in a single session, and
- on key encryption: we denote the encryption of message $M$ using $C$'s secret key $K_C$ by $\langle M \rangle_{K_C}$.

We assume perfect cryptography and nounce generation, which means that an attacker cannot decrypt a message without the corresponding key, nor guess a nounce.

A(lice) and B(ob) try to make establish a secure shared key $K_{AB}$ with the help of an authentication S(erver). They follow the following exchange:




1. Alice first generates a fresh nounce $N_A$ and initiates the exchange with B, with an encrypted tuple $\langle N_A, A, B\rangle_{K_A}$ that only her and the authentication server can decrypt.
2. Bob generates a fresh nounce $N_B$, adds it with an encrypted tuple $\langle N_A, A, B\rangle_{K_B}$ to the message, and forwards the whole thing to the server.
3. The server, knowing the keys of both Alice and Bob, can decrypt the tuples, thus checks that the nounces and participants correspond with the clear parts of the message, generates a fresh shared key $K_{AB}$, and sends it encrypted in two tuples $\langle N_A, K_{AB}\rangle_{K_A}$ and $\langle N_B, K_{AB}\rangle_{K_B}$ to Bob.
4. Bob opens his tuple, checks that the nounce $N_B$ is the desired one, saves the shared key for future exchanges, and forwards the remaining part of the message to Alice.
5. Alice can open the remaining tuple $\langle N_A, K_{AB}\rangle_{K_A}$ and get the shared key.

In order to account for the insecure channel, we have to add an intruder $I$ to the model, who has his own nounce $N_I$ and secret key $K_I$ (also held by the server), and can read and send any message it fancies, but can only decrypt messages of form $\langle \cdots \rangle_{K_I}$ and cannot guess the nounces and keys of Alice and Bob.

## 2 Exercises

The purpose of the exercises is to find a flaw in the protocol using `spin`. The installation of `spin` and its learning are part of your work; see `http://spinroot.com/` for everything you need. The answer for the assignment consists in both a paper part for the models of exercises 1 and 2 and the specification and the attack of exercise 4, and an electronic part with the full `promela` source for the implementations and the specification.

The flaw appears when several sessions of the protocol are executed simultaneously; since for each session one should generate new nounces and shared keys, we need to bound the number of sessions in order to keep a finite system. In fact two interleaved sessions are enough to find the flaw.

**Exercise 1** (Model for Alice and Bob)**.** Although the presentation of the protocol differentiates between Alice and Bob, the roles can be exchanged in a multi-session scenario: a principal might initiate one session as Alice, but receive a message of type 1 and start another session as Bob with both sessions active from there on.

1. Propose a model for a single session $s$ of principal $C$. It takes as parameters a name $C$, a secret key $K_C$, and a nounce $N_C^s$ (that should be unique in the full system). It can either initiate a session as Alice by sending a message of type 1, or as Bob by receiving a message of type 1. Upon completion of a session, respectively after reception and after sending a message of type 4, it goes to a final state with information on the shared key $K_{AB}$ and the participants $A$ and $B$—the set of atomic propositions should allow to access this information.

2. Implement your model in `spin`.

**Exercise 2** (Model for the Server)**.** Model the server and implement your model in `spin`.

**Exercise 3** (Model for the Intruder)**.** We want the intruder to be able to read any message over the public channel, decide to drop it, replicate it, open it and save some part of it (including the contents of an encrypted tuple if the intruder has found the corresponding secret key earlier) using some finite amount of memory, and forge new messages using its saved information or public information.

The intruder has its own name $I$, nounce $N_I$ (we could allow a bounded number of nounces, but actually one is enough) and secret key $K_I$, and knows the names $A$ and $B$.

Implement an intruder model in `spin`.

**Exercise 4** (Specification and Flaw)**.** The overall system consists two sessions for the server, two instances $A$ and $B$ for Alice and Bob with two sessions each (thus four instances of a principal in total), and one instance of the intruder.

1. Specify the following property in LTL, using an appropriate set of atomic propositions matching your model for Exercise 1: if Alice is in its final state and believes it has a shared key with Bob, then the same holds for the other and the intruder does not have this shared key.

2. Construct a Büchi automaton for this LTL formula using the method seen during the lectures.

3. Implement this specification in `spin`.

4. Use `spin` to find a counter-example to this property and draw a timeline execution for it (as done in this subject for the normally expected execution).

5. Does the attack you found look like an actual attack on the protocol? If not, refine your models and/or specification...