Protocol Security Lab — 02239 Data Security

Hound-out: September 6, 2023 — Hand-in: October 6, 2023

Group hand-in: form groups of 3 to 4 participants on DTU Learn (self-enrollment)

The lab exercises use the protocol analyzer OFMC 2023 that is found on DTU Learn. The distribution includes executables for Windows, Mac and Linux.[1] For the exercises please use OFMC with the following command line:

`ofmc --numSess 2` *filename*

You can test OFMC on the lecture example `nspk.AnB` in `examples/cj/6.7...`.

There is also a GUI for OFMC written by a former DTU student:

`https://github.com/ulfur88/OFMC-GUI/releases/tag/v1.2.1`

We have observed that this has trouble on some machines with the latest Java; should work best with JDK11.

Both of the following two exercises are real-world protocols (with minor modifications), both protocols had received a thorough security analysis, and both had attacks that the analysts missed. Have fun finding and fixing them!

Exercise 1: AMP Consider the example `AMP-given.AnB` on DTU Learn which is a simple Single-Sign-On protocol: A, a normal Internet user, can connect to a webserver B, and authenticate to B using a trusted third party s with whom A has a secret $pw(A, s)$. Assume this is a cryptographically strong symmetric key for beginning.

1. Describe the attack found by OFMC: what does the attacker do and what goal is broken here?
2. How could this attack be prevented? Try to find a fix by extending some messages with more information. You should *not* introduce more cryptographic operations.
3. Suppose now $pw(A, s)$ is *not* a strong cryptographic key, but a poorly chosen password. The specification contains the commented-out goal

   ```
   pw(A,s) guessable secret between A,s
   ```

   Explain the attack that OFMC finds on this goal and explain what is the problem.
4. Try to fix also this second problem, i.e., that even in presence of the password the protocol works correctly. Here you have to change a bit the use of cryptography.

[1] For compiling the sources yourself, you need the Glasgow Haskell Compiler.

5. Suppose one wants to implement the single-sign-on with a TLS channel from exercise 1 to secure the connection between A and B. Describe how that relates to the current formulation of the protocol, in particular if that has any advantages/disadvantages. You do not need to make any experiments in OFMC (the problem might be too complex when combining the two protocols).

Exercise 2: Selfie.AnB This protocol, which is also found on the DTU Learn page, has been designed as a key update protocol: two parties already share a secret key and they want to update it for a new shared key. The shared secret key they have before the protocol is `exp(exp(g,secretk(A)),secretk(B))` where `secretk(A)` and `secretk(B)` are long-term secret keys of `A` and `B`, respectively. (Note that `A` does not know the long-term secret of `B`, and `B` does not know the long-term-secret of `A`.)

Moreover, `kdf` is a key-derivation function, to build a new key from given arguments, and `mac` is a keyed hash function (Message Authentication Code). As a cryptographic blackbox it suffices to say that from the function result, one cannot obtain the arguments anymore, but everybody who knows the arguments can apply the function.

The values `g`, `exp(g,secretk(A))` and `exp(g,secretk(B))` are necessarily public. This follows from the knowledge specification when we consider for example the instantiation `A=i, B=b`: if the intruder is playing role `A`, then they must have the respective instantiated knowledge of `A`:

```
i,b,exp(g,secretk(b)),g,secretk(i),mac,kdf,outer;
```

thus, the intruder knows the value `exp(g,secretk(b))` of `b`. Since this holds for any agent playing role `B`, the intruder in fact knows the value `exp(g,secretk(B))` for any agent `B`. In contrast, `secretk(A)` is only known by the intruder for `A=i`.

1. Try to describe and explain the protocol: explain the shape of the initial key, what the new key is, what the purpose of the nonces and MACs are, and why the goals are meaningful.

2. OFMC returns an attack on `Selfie.AnB`. Describe the attack: what happens, what is each agent "thinking" what happens, why does this violate the goals? Essentially: what went wrong here?

3. Fix the protocol and verify the fixed version for two sessions with OFMC.

4. Suppose that after the agents have executed the protocol, the intruder would find out `secretk(A)` (for some honest agent `A`$\neq$`i`). Explain why this would break the secrecy goal of the protocol. Is it possible to modify the protocol so that secrecy would still hold as long as `secretk(A)` is only discovered *after* the execution of the protocol? Note that you cannot check this in AnB/OFMC because of the "after" restriction.