Data security: Lab 1

Autumn 22, 02239
October 11, 2022

s174391, Sölvi Pálsson
s174434, Mads Esben Hansen
s180857, Ditte Aarøe Jepsen
s221969, Rémi Lejeune

Danmarks
Tekniske Universitet

1.1

The *Selfie* protocol is defined to ensure secure update of an already shared key between two parties, here represented by Alice and Bob.
The initial key is given by `exp(exp(g, secretk(A)), secretk(B))`, i.e., a (most likely large) number. As a mean to update this key both Alice and Bob generate one nonce each, N1 and N2 respectively.
The initial key is then updated using a key-derivation-function with the old key, N1 and N2 as parameters, `kdf(exp(exp(g, secretk(A)), secretk(B)), N1, N2)`. The purpose of the nonces are quite simply to have random variables with which the secret key can be updated. The purpose of using a MAC encryption is to "hide" the new secret shared between Alice and Bob while still being able to confirm that the other part received the message correctly, and also to verify each other's identity. The overarching goal is to update the secret key between Alice and Bob while *keeping* it secret during communication channels and preventing that the update happens with an unintended party. Therefore the stated goal that Bob must authenticate the nonce given by Alice and visa versa is strictly necessary and the goal that the new key must be secret between Alice and Bob is obviously meaningful.

1.2

There is a "weak_auth" attack detected, this immediately gives an indication that the issue is to do with failed identification. Looking at the attack trace, we see that Alice thinks she initializes talks with Bob, however, she is really talking with the intruder, `I`. The intruder does not know what to answer but can initialize communication with Alice and get her to tell him what he is supposed to answer. In this way, Alice thinks she has updated the secret key shared between her and Bob, but Bob has no knowledge of this. Specifically, this violates "`B authenticates A on N1`", since Bob has actually never authenticated the nonce given by Alice - the intruder did this!

1.3

In order to fix the authentication attack, the actions stated in the protocol must be amended to ensure authentication from both Alice and Bob in their communication. From the original protocol, we have

```
ACTIONS

 1. A->B:  N1

 2. B->A:  N2, mac(kdf(exp(exp(g,secretk(A)),secretk(B)),N1,N2), N1,N2)

 3. A->B:  A, B, mac(kdf(exp(exp(g, secretk(A)), secretk(B)), N1, N2),
    wa, N1, N2, mac(kdf(exp(exp(g, secretk(A)), secretk(B)), N1, N2),
    N1, N2))
```

Action 2 is amended in order for Alice to properly authenticate Bob.
Bob authenticates his identity in return of nonce `N2` to Alice by stating his identity in the MAC function. Formally, this requires update in action 2 `N2, mac(kdf(exp(exp(g,secretk(A))` `B)`. The final update is thus:

```
ACTIONS
```

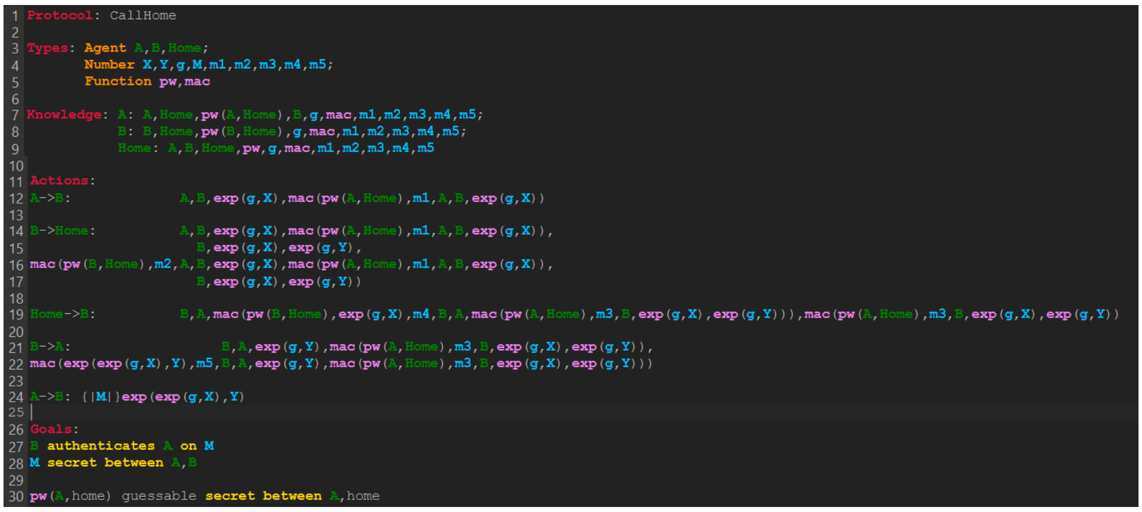
1. `A->B:  N1`
2. `B->A:  N2, mac(kdf(exp(exp(g, secretk(A)), secretk(B)), N1, N2), N1, N2, B)`
3. `A->B:  A, B, mac(kdf(exp(exp(g, secretk(A)), secretk(B)), N1, N2), wa, N1, N2, mac(kdf(exp(exp(g, secretk(A)), secretk(B)), N1, N2), N1, N2))`

Running the Selfie.ANB protocol now satisfies the stated goals and secures the communication between Alice and Bob for two sessions with OFMC.

1.4

If the intruder were to know `secretk(A)` he would be able to figure out the original key. He would do this by asking Bob for his public key, `exp(g, secretk(B))`, and then simply compute `exp(exp(g, secretk(A)), secretk(B))`. Since the new key was generated by taking the key-derivation-function with the old key and two publicly stated nonces, the intruder would be able to compute this himself. To fix this problem, Alice and Bob should broadcast the nonces asymmetrically encrypted. This could be done by, e.g., encrypting the nonce meant for Bob with the public key of Bob, `exp(g,secretk(B))`, and the nonce meant for Alice with the public key of Alice, `exp(g,secretk(B))`. This would still allow Alice and Bob to update their shared key without publicly stating the nonces used to do so. Since the intruder only discovered the old shared key after the update and does not know how the key was updated he cannot feasibly break their new key - he can, however, decipher old messages.

2.0



```
Protocol: CallHome

Types: Agent A,B,Home;
       Number X,Y,g,M,m1,m2,m3,m4,m5;
       Function pw,mac

Knowledge: A: A,Home,pw(A,Home),B,g,mac,m1,m2,m3,m4,m5;
           B: B,Home,pw(B,Home),g,mac,m1,m2,m3,m4,m5;
           Home: A,B,Home,pw,g,mac,m1,m2,m3,m4,m5

Actions:
A->B:             A,B,exp(g,X),mac(pw(A,Home),m1,A,B,exp(g,X))

B->Home:          A,B,exp(g,X),mac(pw(A,Home),m1,A,B,exp(g,X)),
                    B,exp(g,X),exp(g,Y),
mac(pw(B,Home),m2,A,B,exp(g,X),mac(pw(A,Home),m1,A,B,exp(g,X)),
                    B,exp(g,X),exp(g,Y))

Home->B:          B,A,mac(pw(B,Home),exp(g,X),m4,B,A,mac(pw(A,Home),m3,B,exp(g,X),exp(g,Y))),mac(pw(A,Home),m3,B,exp(g,X),exp(g,Y))

B->A:                  B,A,exp(g,Y),mac(pw(A,Home),m3,B,exp(g,X),exp(g,Y)),
mac(exp(exp(g,X),Y),m5,B,A,exp(g,Y),mac(pw(A,Home),m3,B,exp(g,X),exp(g,Y)))

A->B: {|M|}exp(exp(g,X),Y)

Goals:
B authenticates A on M
M secret between A,B

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

Figure 1 – CallHome code

2.1

The *Call-Home* protocol has three types of agents, A, B and home. The protocol is used to securely send a message M from A to B. A and B both share a secret password with a trusted third party, home, that is used to generate a token that A can use to authenticate B before sending the symmetrically encrypted message, M. In the end, B should be able to authenticate A on M and M should be a secret between only A and B. In Figure 2, the five actions of the protocol can be visualized.

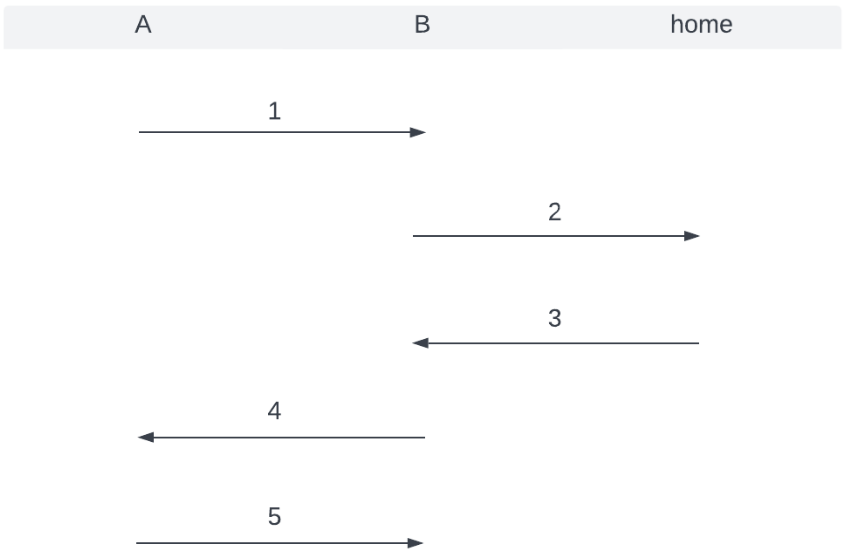


Figure 2 – CallHome Actions - The 3 agents of the protocol and the actions

1. A initializes the protocol by sending a request(1) to B. Request(1) has a hashed value that contains the secret password between A and home (pw(A,home)) as well as an exponential exp(g,X).

2. B continues the protocol by sending a request(2) to home (*Call-Home*), a trusted third party (TTP). Request(2) has the same values as in request(1) followed by values containing a hashed value of B's password with home (pw(B,home)) as well as an exponential exp(g,Y).

3. Home uses the information from request(2) to validate the passwords of the two agents, A and B, before generating new hashed values, tokens, containing the secret passwords between A and home, and B and home that are then sent back to B as request(3)

4. From request(3), B can confirm that it is indeed being sent from home as it can derive the secret password (pw(B,home)) and confirm it. It then constructs a new hashed value containing the token for A, that it got from home and the value exp(exp(g,X),Y) before sending request(4) back to A.

5. A can verify that B did indeed call home by deriving and confirming the secret password (pw(A,home)) from the token in request(4). It then uses the value exp(exp(g,X),Y) to symmetrically encrypt the message M before it sends it as request(5) to B.

2.2

Running the protocol in OFMC yields a violation of the authentication goal in a successful attack through two sessions.

Session 1: From the initiation step (1) the intruder receives information on two things when A initiates communication with B;

- the session dependent nonce `exp(g, X)` to be used in a Diffie-Hellman key exchange with B.

- A MAC-encrypted token generated partly from the personal password.

Through an impersonation of A the intruder forwards a *similar* request to B where `exp(g,X)` is replaced with `g` and the original MAC token is replaced with a random value `c`. B uses the modified tokens received in a server request to `home` in which the intruder receives information on B's contribution for the Diffie-Hellman key exchange, `exp(g,`Y_1`)`.

Session 2: The intruder impersonates A once again and initiates a new parallel session with B, where the original request (1) from A is sent to B. In the succeeding server request (`B -> i(home)`) B choose another secret nonce for exp(g,Y_2) dependent for a later Diffie-Hellman key exchange in current session. The intruder forwards the request to `home` authenticating as B through

```
    i(B) -> home:

1        A, B, exp(g,X), mac(pw(A,home),m1,A,B,exp(g,X)),

2        B, exp(g,X), exp(g,Y2),

3        mac(pw(B,home),m2,A,B,g,c,B,g,exp(g,Y1),

4            mac(pw(A,home),m1,A,B,exp(g,X)),

5            B, exp(g,X),exp(g,Y2))
```

Session 1: In above server request both A and B are successfully authenticated and thus the server responds

```
    home -> i(B):

1        B, A, mac(pw(A,home),m3,B,exp(g,X), exp(g,Y2)),

2        mac(pw(B,home),m4,A,B, mac(pw(A,home),m3,B,exp(g,X),exp(g,Y2))
```

Note in line (2) that B have no knowledge of which session the response is linked to from the arguments in the outer mac encryption, but accepts the response because A is authenticated with the server.
The intruder now has all information required for a Diffie-Hellman key exchange obtained through two different sessions and succeeded in authenticating both parties with *home*. Forwarding the server response to B yields authentication request to A from B

```
    B -> i(A):

1        B, A, exp(g,Y1), mac(pw(A,home),m3,B,exp(g,X), exp(g,Y2)),

2        mac(exp(exp(g,X),Y1), m5, A, B, exp(g,Y1),

3            mac(pw(A,home),m3,B,exp(g,X), exp(g,Y2)))
```

To which the intruder is now able to authenticate as A with the symmetrical Diffie-Hellman key exchange, while B also being authenticated through a TTP.

```
i(A) -> B:      {|M|}exp(exp(g,X),Y1)
```

2.3

As stated above, we get a *weak_ auth* error because the intruder is able to initialize multiple sessions and thereby figure out what to answer in parallel sessions. We are given the hint that we should only change the protocol in the `home->B` message, so let us dissect that:

```
    home->B:
```

```
1        B,A,
2        mac(pw(A,home),m3,B,exp(g,X),exp(g,Y)),
3        mac(pw(B,home),m4,B,A,
4            mac(pw(A,home),m3,B,exp(g,X),exp(g,Y)))
```

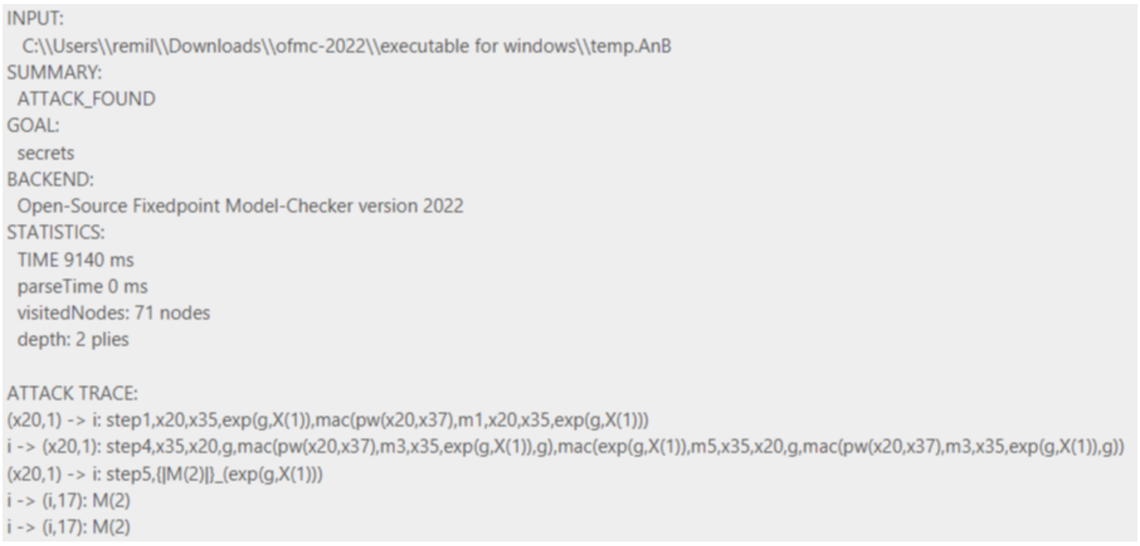
We know that we have to change something in this particular action, we also know that the issue is that information is being re-used in another session. Line (1) simply contains the two parts of the wanted conversation, `A` and `B`. Line (2) contains information that only `home` and `A` would be able to generate - this can later be used by `A` to authenticate that `B` did in fact speak with `home`. Finally, line (3) and (4) contains information that `B` can verify, notice that `B` knows everything in line (3) and line (4) is the same as line (2) (now just inside the `mac` function). The issue is, that this part (that `B` can verify) is not dependent on the specific session, so the intruder can reuse this message in other sessions!
The fix to the protocol is now obvious: Make line (3) dependent on the session. We can do this by including e.g. `exp(g,Y1)`, this is something that changes from session to session that `B` is still able to verify. So the protocol becomes:

```
     home->B:
1        B,A,
2        mac(pw(A,home),m3,B,exp(g,X),exp(g,Y)),
3        mac(pw(B,home),m4,B,A,exp(g,Y),
4            mac(pw(A,home),m3,B,exp(g,X),exp(g,Y)))
```

Running the new protocol on two sessions using *ofmc* yields no attacks.

2.4

Now that the password is guessable we see that *OFMC* finds an attack, namely that the secret between `A` and `B`, `M`, is not secret. Specifically, we see that the intruder is trying to find the `pw(A,Home)` to get M, as attack shown in Figure 3. They receive the request from Alice: `"A, B, exp(g,X), mac(pw(A,Home), m1, A, B, exp(g,X))"`.
Since they know `m1`, `A`, `B`, and `exp(g,X)`, they can guess the password using a brute force method with a dictionary, for example. This method will generate multiple strings, and for each string: We compare `mac(pw(A,Home), m1, A, B, exp(g,X))` with `mac(string, m1, A, B, exp(g,X))`.
If they are equal this means `string` is equal to `pw(A, home)` unless there is a hash collision. From that, the intruder can impersonate home and get `M`. They do this by sending the request from line 21 in Figure 1 to Alice, this is possible because they know `pw(A, home)`.
To fix this attack, we can either have a strong `pw(A, home)`, or add a variable in `mac(pw(A,Home), m1, A, B, exp(g,X))` that they do not know.



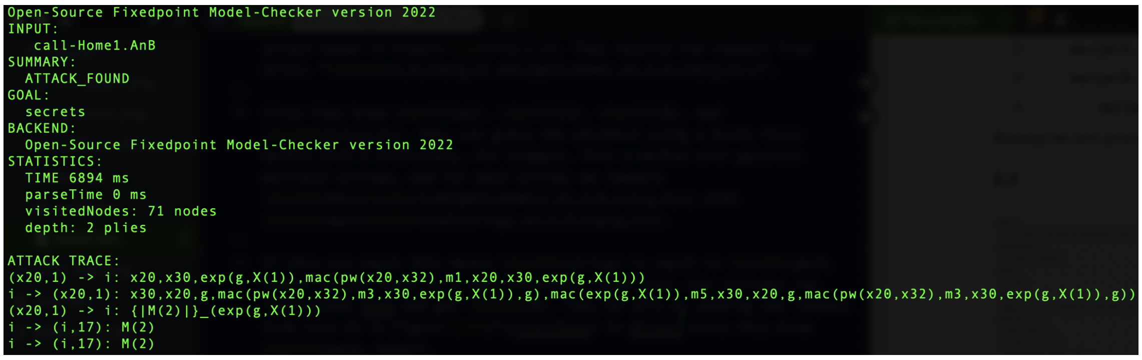
```
INPUT:
   C:\\Users\\remil\\Downloads\\ofmc-2022\\executable for windows\\temp.AnB
SUMMARY:
  ATTACK_FOUND
GOAL:
  secrets
BACKEND:
  Open-Source Fixedpoint Model-Checker version 2022
STATISTICS:
  TIME 9140 ms
  parseTime 0 ms
  visitedNodes: 71 nodes
  depth: 2 plies

ATTACK TRACE:
(x20,1) -> i: step1,x20,x35,exp(g,X(1)),mac(pw(x20,x37),m1,x20,x35,exp(g,X(1)))
i -> (x20,1): step4,x35,x20,g,mac(pw(x20,x37),m3,x35,exp(g,X(1)),g),mac(exp(g,X(1)),m5,x35,x20,g,mac(pw(x20,x37),m3,x35,exp(g,X(1)),g))
(x20,1) -> i: step5,{|M(2)|}_(exp(g,X(1)))
i -> (i,17): M(2)
i -> (i,17): M(2)
```

Figure 3 – Attack found

2.5

Now we replace `home` with `Home`, i.e., we allow the possibility that the "trusted" third party is an intruder. Figure 4 shows the output form *OFMC*. We immediately see that an attack was found, namely that the secrets between Alice and Bob were violated. Looking at the attack trace we see that Alice thinks she contacts Bob, however, she really contacts the intruder. The intruder now represents both Bob and Home, and therefore knows everything it needs to respond to Alice (the second to last action in the protocol). In the end, Alice thinks she has a secure connection with Bob, but in reality, the connection is not secure and it is not even with Bob.



```
Open-Source Fixedpoint Model-Checker version 2022
INPUT:
   call-Home1.AnB
SUMMARY:
  ATTACK_FOUND
GOAL:
  secrets
BACKEND:
  Open-Source Fixedpoint Model-Checker version 2022
STATISTICS:
  TIME 6894 ms
  parseTime 0 ms
  visitedNodes: 71 nodes
  depth: 2 plies

ATTACK TRACE:
(x20,1) -> i: x20,x30,exp(g,X(1)),mac(pw(x20,x32),m1,x20,x30,exp(g,X(1)))
i -> (x20,1): x30,x20,g,mac(pw(x20,x32),m3,x30,exp(g,X(1)),g),mac(exp(g,X(1)),m5,x30,x20,g,mac(pw(x20,x32),m3,x30,exp(g,X(1)),g))
(x20,1) -> i: {|M(2)|}_(exp(g,X(1)))
i -> (i,17): M(2)
i -> (i,17): M(2)
```

Figure 4 – Output from *OFMC* when password is not guessable and `Home` cannot necessarily be trusted.

Notice that the issue stated above is the attack found by *OMFC*. It is sufficient to tell us that the protocol is not secure, but it does not guarantee that this attack is the only possible attack.