Technical University of Denmark

Protocol Security Lab

02239 - Data Security

Group 7

Authors

Student Number:	Name:
s202789	Adamantios Alexandros Kounis
s202518	Christos Grigoriou
s213685	Nikolaos Karavasilis
s202074	Joachim Morken

October 8, 2021

Exercise 1: Kerberos PKInit

1 The purpose of the protocol is to authenticate the agent C to sever s, in order to receive the payload from the latter.

Initially, agent C refers to ath by sending a message encrypted using agent's C private key. This results in the receiver of the message knowing where the message came from. Following, the message that the authenticator sends to C includes the following :

- A message that includes the symmetric key KTemp which is encrypted with the private key of the authenticator, along side a tag number, which is all encrypted with the public key of C.
- Another symmetrically encrypted message that includes the symmetric key that is meant for another encryption between C and g. The whole message is encrypted with Ktemp that C learns in the aforementioned bullet point.
- Lastly, one more message that C can not decrypt which is meant to be forwarded to g. ($\{|ath, C, g, KCG, T1|\}skag$)

The communication continues by C sending a message to g. The message includes the agent C wants to communicate with (s), C includes in the message the previous message which C not able to decrypt along side a message which verifies him, which is encrypted using the symmetric key that C had just found out from ath (KCG). Agent g then responds with another message that C can decrypt using the same symmetric key from auth (KCG) and another message which he can not decrypt which is meant to be forwarded to agent s ($\{|C, s, KCS, T2|\}skgs$). Following the same pattern, now C forwards to s the same message from g that he was not able to decrypt, along with a hashed nonce and who the message comes from. The whole message is also encrypted using this time, the key that C got from g from the previous communication (KCS). Finally, s responds to C with the requested payload and a tag number that is encrypted with the symmetric key that C used (KCS) and the same hashed nonce around the message C sent encrypted again with the KCS symmetric key.

Messages C can decrypt:

- $\{|g, KCG, T1, N1|\}Ktemp$). - line 22
- $\{tag$,{Ktemp}$inv(pk(ath))\}pk(C)$ - line 23
- $\{|s, KCS, T2, N2|\}KCG$ - line 31
- $(\{|hash(T2)|\}KCS)$ - line 36
- $\{|tag, Payload|\}KCS$ - line 36

Keys C learns: :

- Ktemp - line 23

- KCG - line 22
- KCS - line 31

The protocol prevents illegitimate access by exchanging messages between g, s and ath using symmetric keys skag and skgs as well as nonces T1 and T2 for verification amongst the agents.

2 The attack that occurs in PKINIT is known as the man in the middle attack. The attack happens in the first interaction that C is attempting to have with ath. The intruder is able to capture the message from C that is encrypted with his own private key, change the values in it, and pass it as of its own message then using his own private key. This way the intruder is able, once he receives a response from ath, to communicate back and forth with agent C pretending to be the autheticator, g and s all at the same time.

In order to fix this, apart from C encrypting part of the message with his own private key, which secures to whoever decrypts it to be certain that the message came from C, it is also necessary to encrypt the whole message with ath's public key. This results in the auheticator being the only one who has the ability to decrypt it. Formally, the solution is:

```
1 Protocol: PKINIT_short_Revised
2
3 C → auth:        {C,g,N1,{T0,N1,hash(C,g,N1)}inv(pk(C))}pk(ath)
```

3 In order to replace the symmetric key encryption with the Diffie-Hellman key exchange, we need to add a few properties in order to replace it. First we need to add prime p to the numbers listed as well as to the knowledge of both agents who are to use the encryption (C ath). Following that, we will add the exp(p,X) to the values of the first message which after the fix is now secure from any intruders. Next instead, of encrypting the message in line 23 with Ktemp we use exp(exp(p,X),Y). Agent C will now be able to decrypt the message because in addition to the exp(p,X) that he sent to the authenticator he has also knowledge of exp(p,Y) which the authenticator is sending to him in line 22. The goals are adapted accordingly. The secret key between C and ath is changed from Ktemp to exp(exp(p,X),Y) and C authenticated ath on exp(exp(p,X),Y) instead of Ktemp.

```
1 Protocol: Kerberos PKINIT
2
3 # Just the first two steps of the Kerberos PKINIT
4 # (sufficient for finding the attack)
5
6 Types: Agent C,ath,g,s;
7        Number p,N1,N2,T0,T1,T2,Payload,tag;
8        Function pk,hash,sk;
9        Symmetric_key KCG,KCS,Ktemp,skag,skgs
```

```
10
11 Knowledge: C: C,ath,g,s,pk(ath),pk(C),inv(pk(C)),hash,tag,pk,p;
12             ath: C,ath,g,pk(C),pk(ath),inv(pk(ath)),hash,skag,tag,p
13
14 where C!=ath
15
16 Actions:
17
18 C -> ath: {C,g,N1,exp(p,X),{T0,N1,hash(C,g,N1)}inv(pk(C))}pk(ath)
19
20 ath -> C: C,
21           ({| ath,C,g,KCG,T1 |}skag),
22           (exp(p,Y)),
23           ({| g,KCG,T1,N1 |}exp(exp(p,X),Y))
24
25 Goals:
26 KCG secret between C,ath
27 exp(exp(p,X),Y) secret between C,ath
28 C authenticates ath on exp(exp(p,X),Y)
```

Exercise 2: Calling Home

1 The purpose of the protocol is to authenticate A and B using server home before agents A and B are able to establish a shared symmetric key between them using Diffie-Hellman approach. Diffie-Hellman is a non-authenticated key-agreement protocol and our protocol tries to establish a symmetric key using Diffie-Helman but also to ensure the authentication that is missing from Diffie-Helman. The agent home is a server and provides a secure way for A and B to authenticate each other.

The strands of the protocol are depicted in figure 1. For the simplicity of the diagram we have substituted the transmitted messages with *Steps*.

The protocol starts with agent A sending to agent B a message containing *exp(g,X)*, signing it with a MAC and with her secret key *pw(A, home)* (Step1). Agent B receives this message and A's half-key, and could continue with the original Diffie-Helman protocol. However, that would not ensure authentication of A, as B does not know this agent. In other words, he can not be sure of who the creator of *exp(g,X)* is.

B then proceeds to send the MAC tag received from A, alongside with his half-key, to agent home to check if home can verify the authenticity of A. Agent B signs this message with MAC using his secret key *pw(B, home)* (Step2). Agent home is able to verify that this message is indeed sent from agent B as he possesses the necessary knowledge to produce the corresponding MAC tag.

Then home authenticates the message from A using his secret key with A. Agent home then sends back the half-keys to agent B signing them with his secret key with B (Step3). Agent B then authenticates this message with his secret key with home and finally sends back to A the message containing m3 by signing it using the exchanged symmetric key *exp(exp(g,X),Y)* (Step4).

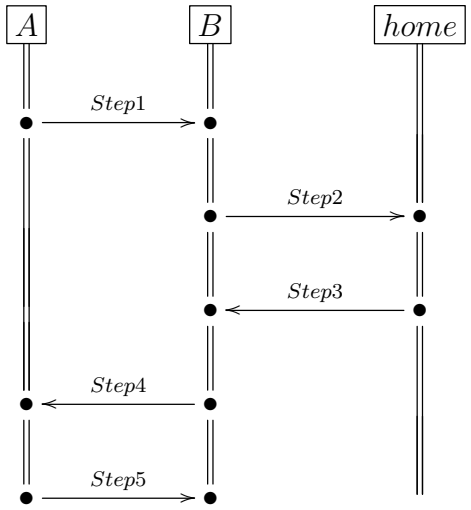

Figure 1: MSC of the Call_Home protocol

Lastly, agent A sends to agent B the secret M using the newly obtained symmetric key between A and B (Step5).

2 The attack analysis using OFMC produces the diagram depicted in figure 2. For the simplicity of the diagram we have substituted the transmitted messages with *Steps'* using an apostrophe to differentiate from the *Steps* of the MSC - figure 1. Using both figures we describe below each *Step'* of the attack:

1. **Step1':** *x802,x801,exp(g,X(1)),mac(pw(x802,home),m1,x802,x801,exp(g,X(1)))*
   In the first step of the attack, the intruder intercepts the message from A(x802) to B(x801).

2. **Step2':** *x802,x801,g,x306*
   The intruder then initiates a communication with B. He is doing *Step1* of the protocol and since it is computationally hard to compute the value of X, he chooses X to be equal to 1. We can clearly see from *Step2'* that instead of sending a MAC value he sends an arbitrary value(x306), which constitutes an indication that it is completely irrelevant for the attack what the intruder chooses here. The intruder is, also, unable to create MAC values, because the shared passwords do not belong to his knowledge.

3. **Step3':** *x802,x801,g,x306,x801,g,exp(g,Y(2)),mac(pw(x801,home),m2,x802,x801,g, x306,x801,g,exp(g,Y(2)))*
   Since B got a message in the previous step, he is trying to execute the next part of the protocol(Step3). However, the intruder intercepts the message from B to home.

4. **Step4':** *Same message as Step1'.*
   Now the intruder executes again *Step1* of the protocol in order to create another

session. This time he sends the exact same message that A wanted to send to B in *Step1'*.

5. **Step5':** *x802,x801,exp(g,X(1)),mac(pw(x802,home),m1,x802,x801,exp(g,X(1))),x801, exp(g,X(1)),exp(g,Y(3)),mac(pw(x801,home),m2,x802,x801,exp(g,X(1)), mac(pw(x802,home),m1,x802,x801,exp(g,X(1))),x801,exp(g,X(1)),exp(g,Y(3)))*
(Session 2) B tries to execute Step2 of the protocol. However, the intruder intercepts the message from B to home.

6. **Step6':** *Same message as in Step5'.*
(Session 1) Now the intruder is doing the previous step again for the first session, but this time he is pretending to be B and forwards the previous message to home. Thanks to this he has authenticated himself with home.

7. **Step7':** *x801,x802,mac(pw(x802,home),m3,x801,exp(g,X(1)),exp(g,Y(3))), mac(pw(x801,home),m4,x801,x802,mac(pw(x802,home),m3,x801,exp(g,X(1)), exp(g,Y(3))))*
(Session 1) Home replies to the intruder, thinking he is B (Step3).

8. **Step8':** *Same message as in Step7'.*
(Session 1) The intruder forwards this message to B pretending to be home. Therefore, he executes *Step3* again.

9. **Step9':** *x801,x802,exp(g,Y(2)),mac(pw(x802,home),m3,x801,exp(g,X(1)),exp(g,Y(3))), mac(exp(g,Y(2)),m5,x801,x802,exp(g,Y(2)),mac(pw(x802,home),m3,x801,exp(g,X(1)), exp(g,Y(3))))*
(Session 1) B now thinks that the intruder is honest, since "home" has authenticated him. Also, B compares the MAC tag with the hash produced by the hash function, unable to identify that the exp(g,Y), inside the hash values, corresponds to the second session. As result, he sends to the intruder the symmetric key generated by using exp(g,1) and exp(g,Y), with Y being the one from session 1 (Y(2)). This is *Step4* of the protocol.

10. **Step10':** */x709/exp(g,Y(2))* (Session 1) The intruder executes the last step of the protocol and replies to B. Now he has succeeded to create a symmetric key with B by pretending to be A.

The attack is a violation against *weak authentication*. What the intruder has done here is to authenticate himself with home, having home thinking he authenticated agent A, and to to create a symmetric key with by B pretending to be A. In particular, the intruder used the "man in the middle" approach and intercepted some messages in order to succeed. He created a Diffie-Hellman key with B, without knowing the initial value of X. The first session with B (Step2') was created with the aim that he will eventually use this session to create a symmetric key between him with B and the second session was created as a means to use A's *pw(A,home)* in order to authenticate himself in the first session, having home thinking he authenticated A.

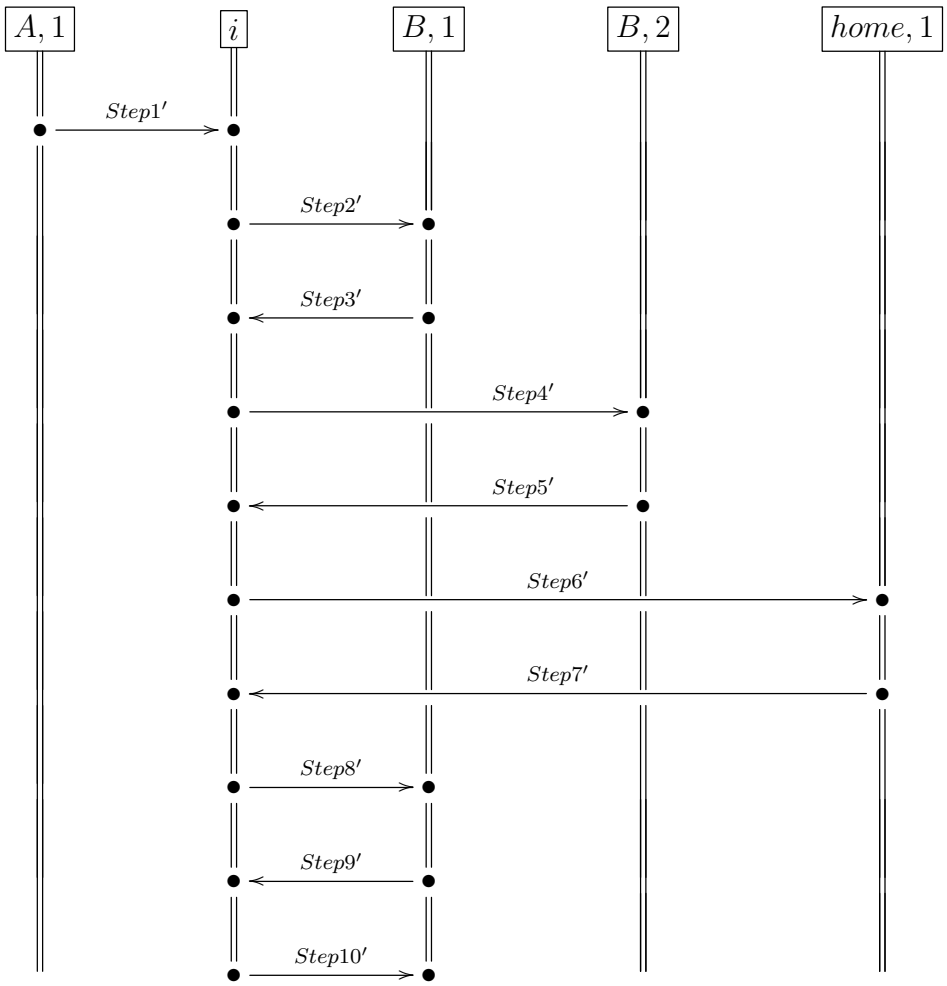

Figure 2: Attack diagram of the Call_Home protocol

3 The problem is indicated by the successful execution of *Step8'*. The intruder is able to use a message that contains X,Y values of another session without B being able to spot it. Specifically, B compares the MAC tag by hashing the message with *pw(B,home)* and does not find any difference. The solution to this problem is to make the intruder unable to use the message from *Step7'* to *Step8'*, since B is going to reject it. This is achieved in *Step3* of the protocol by adding inside the second mac function the value **exp(g,Y)**. Consequently, home is going to set exp(g,Y) equal to the value he acquired from *Step2* and since the intruder cannot create or decypher hashes, B would be able to spot the difference. If we chose to put the exp(g,Y) outside of the mac and not inside, then the intruder would still be successful, since in *Step8'* he can change the value exp(g,Y(3)) to exp(g,Y(2)) that already belong to his knowledge from *Step3'*.

The new *home -> B* is provided below and the red colour indicates the add-on solution.

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

4 The problem is that *pw(A, home)* is now a weak and guessable password. The problem arises when agent A sends the message containing *A,B,exp(g,X),mac(pw(A,home),m1,A,B, exp(g,X))* to agent B. The intruder intercepts the message and can crack the secret key *pw(A, home)*. The intruder now has free rein to act as agent A as he can now create a valid MAC tag that home will authenticate as agent A.

5 This induces an attack against the secrecy goals, namely *M secret between A, B*. If we let the server (Home) be instantiated by the intruder the protocol now has a simple attack. The intuition is that the server now can authenticate agent B without agent A knowing. This will result in the fact that the intruder now can communicate with agent A, while agent A thinks he is still communicating with agent B since the server and trusted third party *Home* authenticated B. The reason for this is that agent B is able to eavesdrop the message A is initially sending to B. In this step he learns the *exp(g, X)* and is thus only missing agent B's half-key. The intruder can now create his own prime number g, and since the server is now dishonest, it can authenticate the intruder as agent B without agent A knowing. The final result is that agent A receives back the prime number g from the authenticated agent B (which is in reality the intruder). Thus we have a secrecy attack where the intruder is able to read the messages A is sending to B.