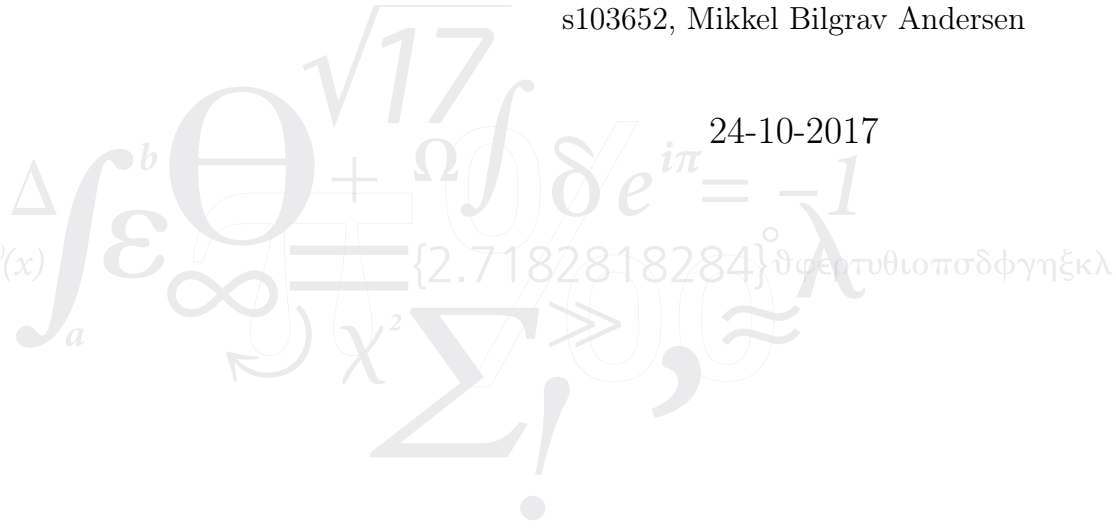
Protocol Security Lab

02239 Data Security

Raghav Patne *s161227*

Collaboration with
s103652, Mikkel Bilgrav Andersen

24-10-2017



1 Exercise 1

1.1 Question 1

The following is the description of the protocol with respect to every action being carried out. The messages that can be Decrypted by C, and the keys that C gets to know, are mentioned as and when they appear, in **Highlight**.

1.
```
C -> ath: C,g,N1,{T0,N1,hash(C,g,N1)}inv(pk(C))
```

   - The agent C sends the ath server its identity, a Nonce value. Along with this, it also sends the timestamp, the nonce, and an irreversible hash value, all three of which are encrypted with C's private key, which essentially means that its digitally signed by C.

2.
```
ath -> C: C,
   ({|ath,C,g,KCG,T1|}skag),
           ({|g,KCG,T1,N1|}Ktemp),
           { tag,{Ktemp}inv(pk(ath))}pk(C)
```

   - The ath reverts back to C with some information. The ath server uses symmetric key encryption using Ktemp to encrypt the information {|g,KCG,T1,N1|}. This **Can be Decrypted by C. C learns about the symmetric key KCG for communication with g.**
   - Although {|ath,C,g,KCG,T1|} is encrypted with a shared key between ath and g and hence cannot be decrypted by C. But, { tag,Ktempinv(pk(ath))} is encrypted using asymmetric public key cryptography, or shortly by using the public key of C and hence **Can be Decrypted by C. C learns the key Ktemp from the Certificate provided by ath.{Ktemp}inv(pk(ath)) is the certificate provided by ath.**.

3.
```
C -> g: s,N2,
 ({|ath,C,g,KCG,T1|}skag),
 ({|C,T1|}KCG)
```

   - Here C gives g the information received from ath for communication between C and g. The information {|C,T1|} is shared using the symmetric encryption using Key KCG.
   - {|ath,C,g,KCG,T1|} is encrypted with the pre-shared key between g and ath, and thus g can decrypt it.

4.
```
g -> C: C,
 ({|C,s,KCS,T2|}skgs),
 {|s,KCS,T2,N2|}KCG
```

- C learns {|s,KCS,T2,N2|}, which was encrypted symmetrically using the key KCG, and thus **it can be decrypted by C, and C learns a key KCS, which is also a symmetric key for communication with s.**
- notice that the timestamp T2 and the Nonce N2 is also given.

5. `C -> s: ({|C,s,KCS,T2|}skgs),`
   `{|C,hash(T2)|}KCS`

- C provides s with the information {|C,s,KCS,T2|} encrypted symmetrically using the key skgs, and it cannot be read by c. It contains the information for s, for communication with c.
- C also provides symmetrically encrypted information {|C,hash(T2)|}KCS. With the hash function being known to s, it can reproduce T2, the timestamp.

6. `s -> C: ({|hash(T2)|}KCS), {|tag,Payload|}KCS`

- s reverts back to C, using the symmetric encryption key KCS, with the timestamp T2, and also the main message {|tag,Payload|}.**This of course can be decrypted by C.**

To prevent illegitimate access, the protocol uses, the digital signatures. As ath knows the public key of C, it can at very first step, confirm and be sure that it is talking to C. Also, the Ktemp is provided by ath in a digital certificate and thus, C can be assured of its integrity. This ktemp is used to further decrypt another part of the message. Thus, all other keys are henceforth safe. The hash function also prevent illegitimate access, as it is largely irreversible, and hence an eavesdropper cannot figure out the actual information, by knowing only the hash function value.

1.2 Question 2

The attack trace is as shown below.

```
ATTACK TRACE:
(x501,1) -> i: x501,g,N1(1),{T0(1),N1(1),hash(x501,g,N1(1))}_inv(pk(x501))
i -> (ath,1): i,g,N1(1),{x306,N1(1),hash(i,g,N1(1))}_inv(pk(i))
(ath,1) -> i: i,{|ath,i,g,KCG(2),T1(2)|}_skag,{|g,KCG(2),T1(2),N1(1)|}_Ktemp(2),{tag,{Ktemp(2)}_inv(pk(ath))}_(pk(i))
i -> (x501,1): x501,x405,{|g,KCG(2),T1(2),N1(1)|}_Ktemp(2),{tag,{Ktemp(2)}_inv(pk(ath))}_(pk(x501))
```

Figure 1: Attack trace for PKINIT-short.AnB

We can see from the attack trace, that the intruder acts as a classic example of a man-in-the-middle. Although this is attack in our case is not a exploitation of the secrecy, but more of a weak authentication case. The intruder, interrupts the message from C, and forwards its own message, by putting its own values in the fields. The ath responds to this request of i, who then further acts as ath for the client server.

This can be corrected simply by putting the information about the client in the certificate issued by the ath server. This insures full authentication that the ath server is

talking to the correct client. The client can be sure, because it is a certificate from a trusted server ath, whose public key it knows.

Thus, the correct protocol specification would be as follows:-

```
Protocol: Kerberos_PKINIT_setup
# Just the first two steps of the Kerberos PKINIT
# (sufficient for finding the attack)

Types: Agent C,ath,g,s;
       Number N1,N2,T0,T1,T2,Payload,tag;
       Function pk,hash,sk;
       Symmetric_key KCG,KCS,Ktemp,skag,skgs

Knowledge: C: C,ath,g,s,pk(ath),pk(C),inv(pk(C)),hash,tag,pk;
   ath: C,ath,g,pk(C),pk(ath),inv(pk(ath)),hash,skag,tag

where C!=ath

Actions:


C -> ath: C,g,N1,{T0,N1,hash(C,g,N1)}inv(pk(C))

ath -> C: C,
({|ath,C,g,KCG,T1|}skag),
        ({|g,KCG,T1,N1|}Ktemp),
        { tag,{Ktemp,C}inv(pk(ath))}pk(C)


Goals:
C authenticates ath on Ktemp
Ktemp secret between C,ath
KCG secret between C,ath
```

1.3 Question 3

The changes that need to be made to implement Diffie-Hellman key-exchange, is that we have to make the agents aware of the number a, and the random number generators X, and Y used for the computations exp(a,X) and exp(a,Y). The function exp, which is the exponential modulus function need not be mentioned explicitly as it is understood by OFMC. Further the actions are changed to allow for the Diffie-Hellman exchange, and the goals are changed to authenticate the value of the function exp(exp(a,X),Y) and exp(exp(a,Y),X). Thus, the specification is as follows:-

```
Protocol: Kerberos_PKINIT_setup
```

```
# Just the first two steps of the Kerberos PKINIT
# (sufficient for finding the attack)

Types: Agent C,ath,g,s;
       Number X,Y,a,N1,N2,T0,T1,T2,Payload,tag,a;
       Function pk,hash,sk;
       Symmetric_key KCG,KCS,Ktemp,skag,skgs

Knowledge: C: C,ath,g,s,pk(ath),pk(C),inv(pk(C)),hash,tag,pk,a;
   ath: C,ath,g,pk(C),pk(ath),inv(pk(ath)),hash,skag,tag,a

where C!=ath

Actions:

C -> ath : {C ,ath , exp (a , X )} inv( pk ( C ))

ath -> C : {C ,ath , exp (a , Y )} inv( pk ( ath ))

C -> ath: {|C,ath,g,N1,{T0,N1,hash(C,g,N1)}inv(pk(C))|}exp(exp(a,X),Y)

ath -> C: C,
({|ath,C,g,KCG,T1|}skag),
        ({|g,KCG,T1,N1|}exp(exp(a,X),Y)),
        { tag,{exp(exp(a,X),Y)}inv(pk(ath))}pk(C)

Goals:
C authenticates ath on exp ( exp (a , X ) , Y )
ath authenticates C on exp ( exp (a , X ) , Y )
exp ( exp (a , X ) , Y ) secret between C , ath
KCG secret between C,ath
```

2 Exercise 2

2.1 Question 1

1. Based on the initial knowledge it is clear that there is a asymmetric encryption in place. Each of the agents are aware of the public keys of every other agents. The agent s acts as a honest(trustworthy) server, which is also why it mentioned as a constant!

2. The protocol uses asymmetric and symmetric encryption for communication between the clients as in :-

```
A->B: {A,B,Request,K}pk(B)
B->A: {|A,s,B,ReqID,Request|}K
```

3. The protocol, makes use of digital signatures to ensure the authentication of messages between the agents and the server, as in :-

```
A->s:  { {A,s,B,ReqID,Request}inv(pk(A)) }pk(s)
```

4. The protocol uses session-ID's to avoid replays of old messages(Read ReqID).

5. The protocol also uses certificates to assure the clients of authentication of messages from the server, as in :-

```
s->A:  { {A,s}inv(pk(s)) }pk(A)
```

2.2 Question 2

The attack trace for the AMP protocol specification is as shown:-

```
ATTACK TRACE:
(x802,1) -> i: {x802,i,Request(1),K(1)}_(pk(i))
i -> (x801,1): {x802,x801,x306,K(1)}_(pk(x801))
(x801,1) -> i: {|x802,s,x801,ReqID(2),x306|}_K(1)
i -> (x802,1): {|x802,s,i,x506,Request(1)|}_K(1)
(x802,1) -> i: {{x802,s,i,x506,Request(1)}_inv(pk(x802))}_(pk(s))
i -> (s,1): {{x802,s,i,x506,Request(1)}_inv(pk(x802))}_(pk(s))
(s,1) -> i: {{x802,s}_inv(pk(s))}_(pk(x802))
i -> (x802,1): {{x802,s}_inv(pk(s))}_(pk(x802))
(x802,1) -> i: {|{x802,s}_inv(pk(s))|}_K(1)
i -> (x801,1): {|{x802,s}_inv(pk(s))|}_K(1)
(x801,1) -> i: {|x306,Data(6)|}_K(1)
```

Figure 2: Attack trace for AMP.AnB

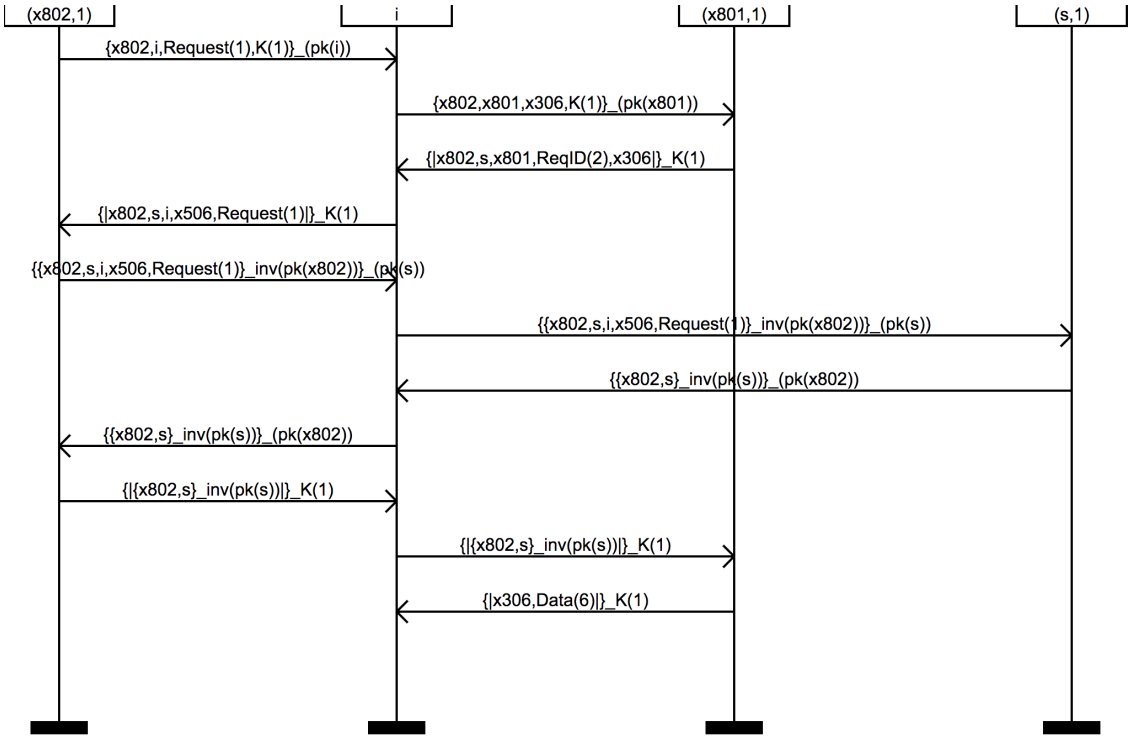


Figure 3: Message sequence chart for Attack trace for AMP.AnB

The message sequence chart helps to better visualize the attack. So, basically what is happening here is that the intruder takes up false identity for every agent in the Protocol. If we observe both the attack trace and the message sequence carefully, we can tally that the intruder takes up the identity of B for the agent A, and that of A for the agent B in the first session. Then it goes on to act as s for the agent A, and for s it acts as B. After this, the intruder i again becomes B for A and A for B. Despite this, all the agents of the protocol think that the messages are being delivered from the right parties, but in reality they are all going through i. Thus, there isn't any authentication, even then the agents believe that the messages originate from the correct places.

2.3 Question 3

Thus, from the previous attack-trace, it is clear that if anyway, we can make s reveal the identity of what it thinks B is, and in revealing the identity also insure that it cannot be altered by the intruder, we can make all agents confirm the overall state of the protocol. For this we will have to use the protective cover that the certificate provides us, in the following actions of the protocol:-

```
s->A:  { {A,s}inv(pk(s)) }pk(A)

A->B: {|{A,s}inv(pk(s))|}K
```

Thus, we include the information B,ReqID in the certificate, and thus get the following specification :-

```
Protocol: AMP

Types: Agent A,s,B;
       Number Request,ReqID,Data;
       Symmetric_key K

Knowledge:
       A: A,s,B,pk(s),pk(A),inv(pk(A)),pk(B);
       s: A,s,pk(s),inv(pk(s)),pk(A);
       B: s,B,pk(s),pk(B),inv(pk(B))
       where B!=s
Actions:

A->B: {A,B,Request,K}pk(B)
B->A: {|A,s,B,ReqID,Request|}K

A->s:  { {A,s,B,ReqID,Request}inv(pk(A)) }pk(s)
s->A:  { {A,s,B,ReqID}inv(pk(s)) }pk(A)  #change

A->B: {|{A,s,B,ReqID}inv(pk(s))|}K       #change
B->A: {|Request,Data|}K

Goals:

B authenticates A on Request
A authenticates B on Data
Data secret between B,A
```

2.4 Question 4

After making s as S on the corrected protocol and then checking the specification, the following attack trace was observed :-

```
ATTACK TRACE:
i -> (x401,1): {x402,x401,x207,x208}_(pk(x401))
(x401,1) -> i: {|x402,i,x401,ReqID(1),x207|}_x208
i -> (x401,1): {|{x402,i,x401,ReqID(1)}_inv(pk(i))|}_x208
(x401,1) -> i: {|x207,Data(2)|}_x208
```

Figure 4: Attack trace for modified specification

I think the reason why the attack was traced was because since the removal of an honest(trustworthy) entity s, and making it also a normal agent, there is no way to ensure the integrity of the certificate that S is now issuing, and hence there is no way to trust any information. Thus the messages cannot be fully authenticated when there is no trusted or honest agent.

I don't think it is possible the fix this version with the same initial knowledge and goals, as an intruder can successfully assume any of the three roles now. Although one might be tempted to think that using the Diffie-Hellman key Exchange, might be the solution,as it is one of the most secure ways for key-exchanges, but it still requires a small change in the initial knowledge and goals of the protocol.