Technical University of Denmark

02239

Data Security

Protocol Security Lab

Authors:	*Student Nr.:*
Jeff Gyldenbrand	s202790
Marcus Pagh	s151714

December 20, 2021

Contents

1 Exercise 1: Kerberos PKInit

1.1 Question 1

We have created an illustration of the `Kerberos_PKINIT`-protocol of `PKINIT.AnB` which is shown in fig 3.4. As walthrough of the protocol is described below:

- First step is for C to contact the trusted server *ath* in order to get authenticated for the ticket granting server. In this message is included information about who he is and what he wants. This is signed by his private key, allowing *ath* to verify his identity. A timestamp, *T0* is also added to the encrypted part of the message to avoid replay-attacks[1]
- In step two, the client C is receiving a ticket-granting ticket from *ath*. C can decrypt just the last part of the message:

  $$\{tag, \{Ktemp\}inv(pk(ath))\}pk(C)$$

  using his own private key, insuring confidentiality. He is now able to read *tag* and the symmetric key *Ktemp* which is signed by *ath* insuring integrity. With *Ktemp* C is able to decrypt the another part of the received message,

  $$\{|g, KCG, T1, N1|\}Ktemp$$

  , thus learning the symmetric session-key *KCG*.
- Now C contacts the ticket-granting server, forwarding the ticket-granting ticket he just received as well as information about the service he would like access to and an encrypted message containing his identity, encrypted by the session-key *KCG* he learned from authenticating with *ath*.
- g responds with a service-ticket as well as another session-key, *KCS* which C learns by decrypting the message

  $$\{|s, KCS, T2, N2|\}KCG$$

  with his session-key *KCG* from *ath*.
- It is now finally time for C to contact the service-provider. He does this by sending his service-ticket as well as another message to authenticate himself encrypted with his newly-acquired session-key, *KCS*.
- Finally s is happy and responds with the payload (service) as requested, encrypted by the symmetric session-key *KCS*

When the protocol is done, C has learned about: *tag,* ***Ktemp****,* ***KCG****, T1, N1,* ***KCS****, T2, N2, Payload* (Keys in bold)

[1] An encrypted timestamp is included in every further step as well. This is omitted from the walk-through to prevent repeating the same explanation.

1.2 Question 2

The attack is an attack against *weak authentication*. It is not considered an attack as such, because the intruder, i does not really learn any secrets from the messages between C and *ath*. However, C can't verify that the message is meant for him. i is however able to - for some time at least - keep C occupied by simulating a lost or bad connection.

As observed in the *attack trace*[1.1], in step (1), i intercepts the message sent from C, denoted *x501*. In step (2) i creates hes own message, encrypts it with hes own public key. *ath* then respons to i with a service granting ticket in step (3). Finally i forwards this message to C in the last step.

```
ATTACK TRACE:
(x501,1) -> i: step1,x501,g,N1(1),{T0(1),N1(1),hash(x501,g,N1(1))}_inv(pk(x501))
i -> (ath,1): step1,i,g,N1(1),{x306,N1(1),hash(i,g,N1(1))}_inv(pk(i))
(ath,1) -> i: step2,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): step2,x501,x405,{|g,KCG(2),T1(2),N1(1)|}_Ktemp(2),{tag,{Ktemp(2)}_inv(pk(ath))}_(pk(x501))
```

Figure 1.1: Exercise 1.2: Attack Trace

The fix for this attack is for *ath* to include the reciever, C in the message, such that the message goes from:

$$\{tag, \{Ktemp\}inv(pk(ath))\}pk(C)$$

to

$$\{tag, \{Ktemp, C\}inv(pk(ath))\}pk(C)$$

This message is signed with the private key of *ath* and cannot be forged by i. This way C can be sure its meant for him.

Worth mentioning, even though this is not the answer for this question, is that we found another way to prevent this attack by simply encrypting the first message from C with the public key of *ath*:

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

1.3 Question 3:

It is possible for C and *ath* to do a `Diffie-Hellman` key-exchange instead of having *ath* generate the key *Ktemp*. For this exchange to work, it was necessary to add the public `Diffie-Hellman` group, *dh* (the name g is already in use) to the knowledge of both participants. To do the actual exchange, *exp(dh,X)*, where X is chosen by C, is added to the first message from C to *ath*. Since this exchange is generally prone to man-in-the-middle attacks, *exp(dh,X)* is added to the signed part of the message. To generate the actual key, *ath* has to similarly choose a Y and send back *exp(dh,Y)*. This message is likewise signed. Since *ath* now knows both $exp(dh, X)$ and Y, it can already encrypt *KCG* with $exp(exp(dh, X), Y)$ in its first message. When C receives this message, he will know both X

and $exp(dh, Y)$ and can thus construct a key $exp(dh, Y), X)$. Even though C does not know the Y chosen by *ath*, by utilizing the property $(dh^X modp)^Y modp = (dh^Y modp)^X modp$ (in OFMC this is equivalent to $exp(exp(dh, X), Y) = exp(exp(dh, Y), X))$, he is able to decipher the message and reading KCG while still insuring confidentiality. This is shown in fig 3.1

2 Exercise 2: Calling Home

2.1 Question 1

This protocol is trying to establish a secret / encrypted channel of communication between the two agents A and B insured by an trusted server, *home*.

In step(1):

$$A \rightarrow B : A, B, exp(g, X), mac(pw(A, home), m1, A, B, exp(g, X))$$

it is observed that A generates his part of the key-negotiation g^X from the publicly agreed upon g. A sends this value along with a MAC-tag of the message, including a nonce, *m1*, known to B and *home*, and the password known only to A and *home*. In step(2), B then forwards this message to *home* along with the MAC-tag of his message, including the password only known to himself and *home*. B also includes, in this MAC, the originally MAC received from A, this way *home* can ensure *authenticity* and *integrity* upon both A and B:

$$B \rightarrow home : A, B, exp(g, X), mac(pw(A, home), m1, A, B, exp(g, X)),$$
$$B, exp(g, X), exp(g, Y), mac(pw(B, home), m2, A, B, exp(g, X),$$
$$mac(pw(A, home), m1, A, B, exp(g, X)), B, exp(g, X), exp(g, Y))$$

In step(3) *home* response with a message back to B, where it is supposed to ensure to B that everything is OK:

$$home \rightarrow B : B, A, mac(pw(A, home), m3, B, exp(g, X), exp(g, Y)),$$
$$mac(pw(B, home), m4, B, A, mac(pw(A, home), m3, B, exp(g, X), exp(g, Y)))$$

However, the message it a bit flawed. This is explained in 2.2.

In step(4), B sends the value of his part of the key g^X to A. Now both agents can create the secret key by: g^{XY} and use this key to encrypt / decrypt their messages.

2.2 Question 2

The attack is an attack against weak authentication. From step(1) in the attack trace[2.1], it is observed that the intruder, i, is intercepting the message:

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

The message is from A, denoted *x802* in the attack trace, and meant for B, denoted *x801*. Along with this message is A's part of the key-negotiation, generated from his privately chosen, secret value, X: $exp(g^X)$, and a MAC-tag of the message. From the publicly available g, i forges the message by choosing $X = 1$ such that $g^x = g^1 = g$. Since $pw(A, home)$ is still kept secret from i, the MAC-tag is no longer valid. In step(2), i then forwards the new message to B:

$$i- > (B, 1) : A, B, g, x306$$

.

In step(3), B then forwards the message to *home*, with a message containing his part of the key: $exp(g, Y)$. This message is also intercepted by i, letting i learn $exp(g, Y)$ from B.

In step(4), i instantiates a new session with B, this time forwarding the intact message from A from step(1). When B again forwards this message to *home* in step(5), i intercepts it once more, this time forwarding it as is to *home* in step(6).

In step(7) *home* replies to the message believed to come from B, allowing i in step(8) to forward the reply back to B in the first session where i intercepted the message from B to *home* in step(3). Since the MAC-tag of $exp(g, X), exp(g, Y)$ is made with the key $pw(A, home)$ which B does not know, and neither $exp(g, X)$ nor $exp(g, Y)$ itself, B is only able to verify that the message originates from *home*, but not that the message actually contains the $exp(g, X)$ and $exp(g, Y)$ that B expects. This is where the crucial vulnerability lies, as the message from *home* actually originates from another session.

In step(9) B replies back to A with a message containing his part of the original key-negotiation, $exp(g, Y)$. Again i intercepts. At this point i knows both X (As he has chosen it to be 1 himself) and $exp(g, Y)$ allowing him in step(10) to encrypt a message, M, with the key $exp(exp(g, Y), X)$ and sending it to B. As B believes only he and A knows the key and he is able to decrypt it, he is able to authenticate i as A on the message M, violating the goal `B authenticates A on M`.

```
ATTACK TRACE:
(x802,1) -> i: step1,x802,x801,exp(g,X(1)),mac(pw(x802,home),m1,x802,x801,exp(g,X(1)))
i -> (x801,1): step1,x802,x801,g,x306
(x801,1) -> i: step2,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)))
i -> (x801,2): step1,x802,x801,exp(g,X(1)),mac(pw(x802,home),m1,x802,x801,exp(g,X(1)))
(x801,2) -> i: step2,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)))
i -> (home,1): step2,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)))
(home,1) -> i: step3,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))))
i -> (x801,1): step3,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))))
(x801,1) -> i: step4,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))))
i -> (x801,1): step5,{|x709|}_(exp(g,Y(2)))
```

Figure 2.1: Exercise 1.2: Attack Trace

2.3 Question 3

In order to fix the protocol we need to change the message sent by *home* to B:

$$home- > B : B, A, mac(pw(A, home), m3, B,$$

$$exp(g, X), exp(g, Y)), mac(pw(B, home), m4, B, A,$$

$$mac(pw(A, home), m3, B, exp(g, X), exp(g, Y)))$$

into:

$$home- > B : \mathbf{exp(g,X)}, \mathbf{exp(g,Y)}, B, A, mac(pw(A, home), m3, B,$$

$$exp(g, X), exp(g, Y)), mac(pw(B, home), m4, B, A,$$

$$\mathbf{exp(g,X)}, \mathbf{exp(g,Y)}, mac(pw(A, home), m3, B, exp(g, X), exp(g, Y)))$$

This will allow B to not only verify that the message originates from *home* (Since only *home* (and B) can create a MAC-tag with $pw(B, home)$ as key), but also that the message is indeed meant for B to establish a confidential channel between B and A. This works since B is now able to verify that he and *home* agrees on the chosen values, removing the vulnerability of accepting a simple replay from another session. See the complete AnB code in fig 3.2

2.4 Question 4

We set a new goal where the password *pw(A,home)* is a guessable secret:

$$pw(A, home) \ guessable \ secret \ between \ A, home$$

From the OFMC-tutorial[2], it is stated that,

[2]http://www2.imm.dtu.dk/ samo/OFMC-tutorial.pdf pg. 81-83

> "[...] whenever a message is produced by en honest agent that contains the password, this triggers a new secrecy goal, namely to produce the same message with the password replaced by *guessPW* [...] This acts as a witness that the intruder could produce the same message by guessing."

By inspecting the attack trace[2.2] we observe in step(1) that the honest agent A, denoted *x20*, is sending a message containing his password, *pw(x20,home)*. This message is intercepted by the intruder, i, whom is performing the guessing attack by recursively checking m_i in *pw*(m_1, m_2). As seen in the last two step, i is able to produce the same message with *guessPW*

```
ATTACK TRACE:
(x20,1) -> i: step1,x20,x35,exp(g,X(1)),mac(pw(x20,home),m1,x20,x35,exp(g,X(1)))
i -> (i,17): x20,x35,exp(g,X(1)),mac(guessPW,m1,x20,x35,exp(g,X(1)))
i -> (i,17): x20,x35,exp(g,X(1)),mac(guessPW,m1,x20,x35,exp(g,X(1)))
```

Figure 2.2: Exercise 2.4: Attack Trace

2.5 Question 5

As seen in the complete code[3.3] we have changed the former trusted server, *home* to an agent with a normal role, *Home*. This means that this agent can be instantiated by the intruder, thus no longer is considered an trusted server.

In step(1), A, denoted *x20* in the attack trace[2.3], sends his message, just as usual. The intruder, i, intercepts this message, which contains A's part of the key, *exp*(g^X). Now i simply generates his part of the key, *exp*(g^Y) where $Y = 1$, thus $g^Y = g^1 = g$ and obtains the secret key from g^{XY}. Because i has knowledge on *pw(A,Home)* he is able to decrypt the MAC in the message from A, and immediately creates a response message to A, as seen in step(3). From A's perspective, there now exists an secure channel from A to B where in reality A is talking to the intruder.

```
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 2.3: Exercise 2.5: Attack Trace

```
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,dh,X,Y;
       Function pk,hash,sk;
       Symmetric_key KCG,KCS,skag,skgs

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

where C!=ath

Actions:


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

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


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

```

Figure 3.1: A modified version of the (corrected) example file `PKINIT-short.AnB` where the key *Ktemp* is not generated by *ath* but is obtained from a `Diffie-Hellman` key-exchange.

3 Appendix

3.1 Code

```
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:      exp(g,X),exp(g,Y),B,mac(pw(A,home),m3,B,exp(g,X),exp(g,Y)),
mac(pw(B,home),m4,B,A,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 3.2: Question 2.3: CallHome. Attack is fixed

```
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:      exp(g,X),exp(g,Y),B,mac(pw(A,Home),m3,B,exp(g,X),exp(g,Y)),
mac(pw(B,Home),m4,B,A,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 3.3: Question 2.5: CallHome. Home is now a normal role

3.2 Illustrations

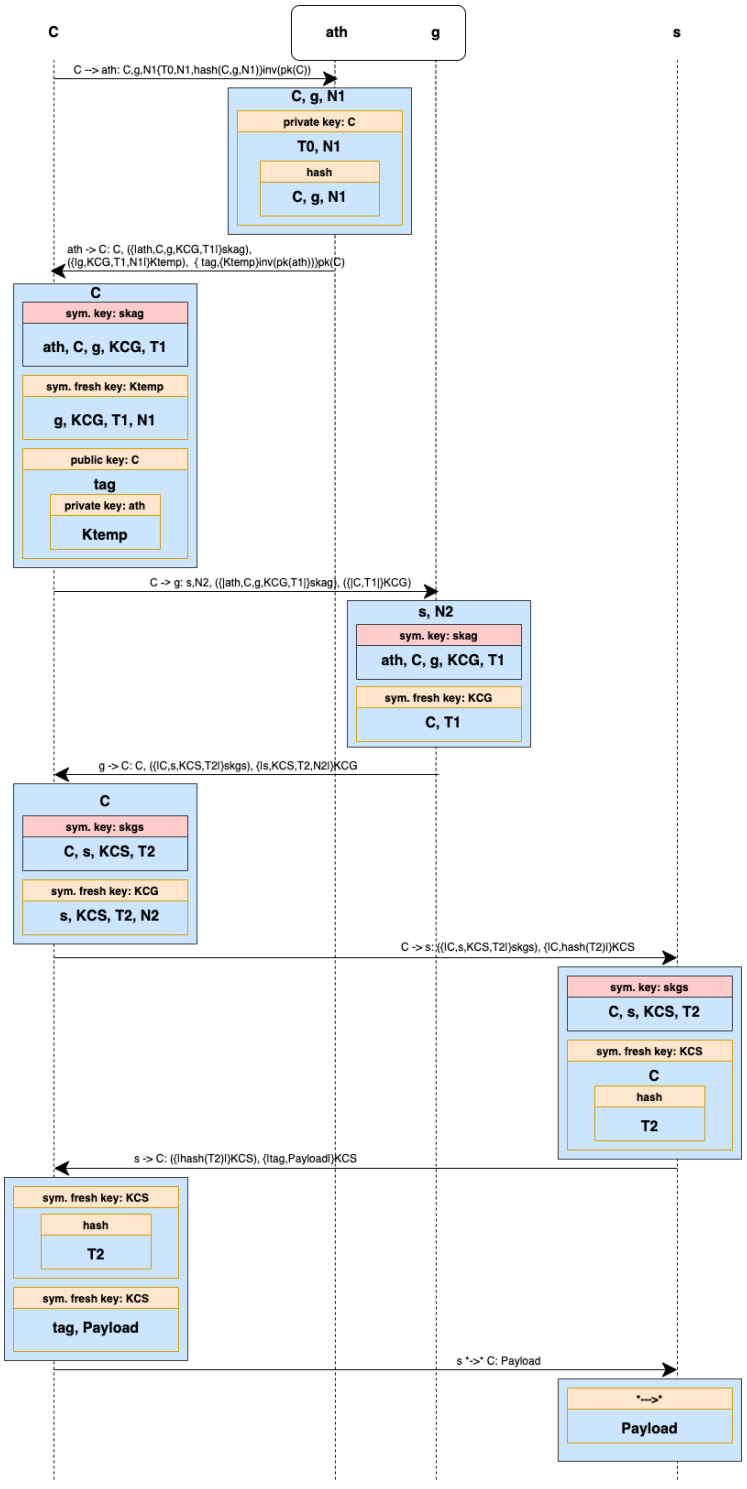

Figure 3.4: Exercise 1: Illustration of PKINIT.AnB. Blue boxes are the complete message, which whitin contains the individual parts of the message. Boxes with a red field are messages C is not able to read or decrypt

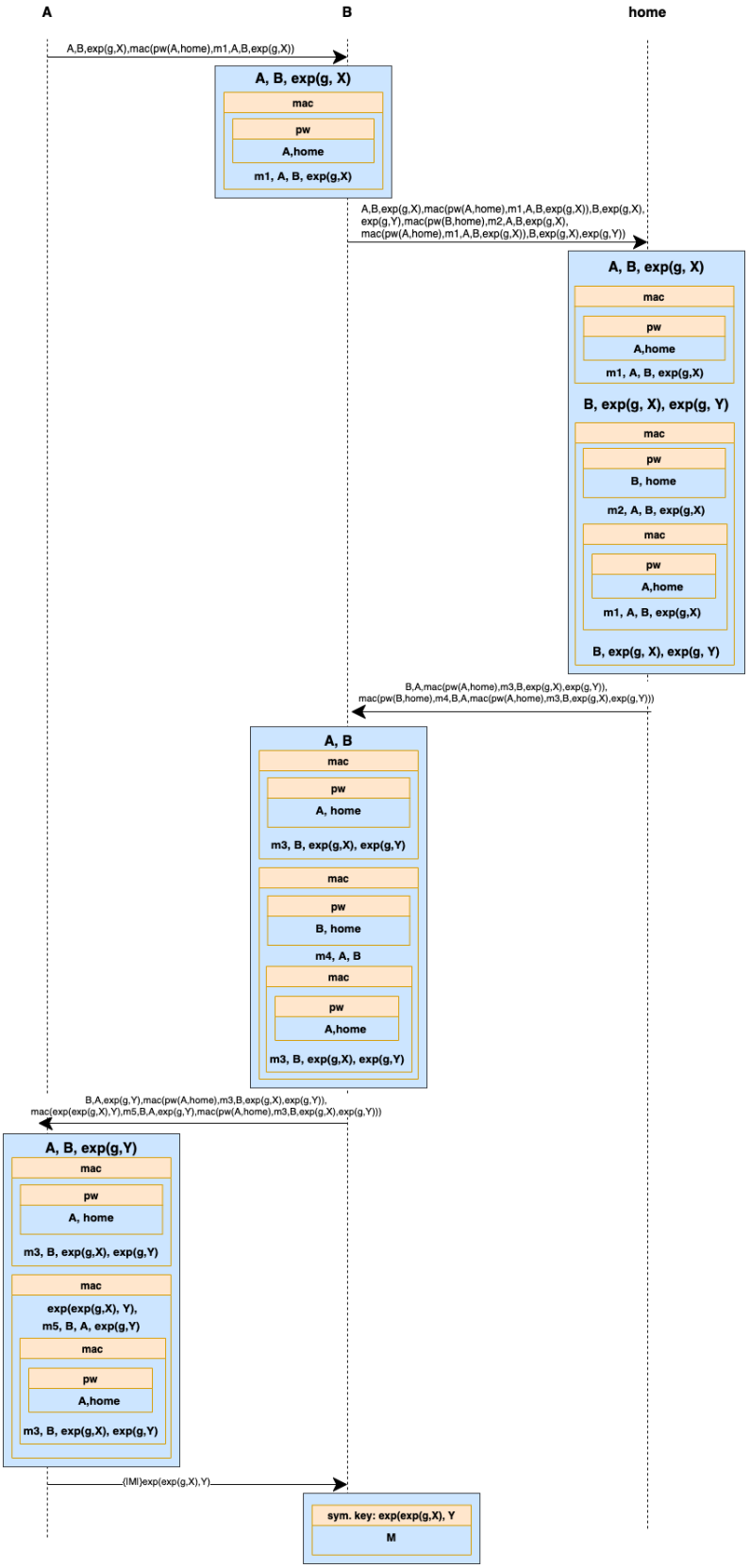

Figure 3.5: Exercise 2: Illustration of call-home.AnB. Blue boxes are the complete message, which whitin contains the individual parts of the message.