Technical University of Denmark

02239

Data Security

Laboratory 1: Protocol Security

Author:
Román Cárdenas

Student Number:
s182004

October 10, 2018

Contents

List of Figures

List of Codes

1 H.530 Protocol

1.1 Original H.530 protocol description

The H.530 protocol tries to address the problem of the Diffie-Hellman communication protocol authentication. As described in [1, p. 19], without the authentication of half-keys, this protocol could lead to a *man-in-the-middle* attack. In this exercise, the authentication is based on a third entity s, which is trusted by Alice (A) and Bob (B). Both A and B share a secret key with the trusted source s, so this last entity is in charge of signing A and B authenticity in order to ensure both parts that the other part is trustworthy.

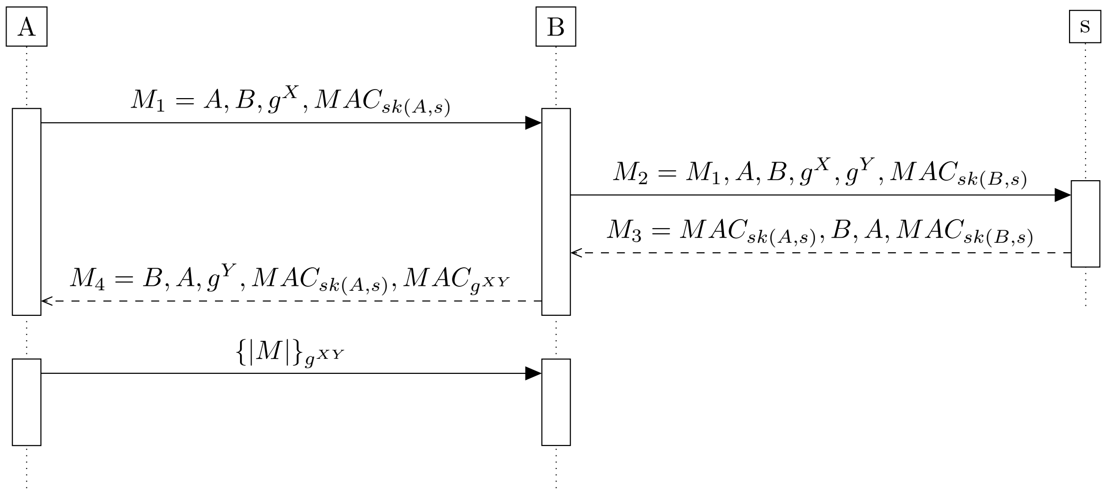

A
B
s
$M_1 = A, B, g^X, MAC_{sk(A,s)}$
$M_2 = M_1, A, B, g^X, g^Y, MAC_{sk(B,s)}$
$M_3 = MAC_{sk(A,s)}, B, A, MAC_{sk(B,s)}$
$M_4 = B, A, g^Y, MAC_{sk(A,s)}, MAC_{g^{XY}}$
$\{|M|\}_{g^{XY}}$

Figure 1: H.530 sequence diagram

As shown in *Figure 1*, the protocol would be as follows:

– First of all, Alice sends a message to Bob with her half-key. The message is signed so that the trustworthy entity can verify the author of that message as Alice.

– Bob asks the trustworthy entity about the authenticity of the message. He also includes his half-key and signs the message so that the trustworthy entity can verify the author of this message as Bob.

– If both signatures are correct, the trustworthy entity will respond Bob by sending a message in which it states that A and B are trustworthy. The message will be signed twice in order for both Alice and Bob to be able to verify the third party's verdict.

– Finally, when Bob verifies that Alice is trustworthy, he sends her a message with his half-key signed twice, one by the third party entity and the other by Bob himself, using the shared key g^{XY}.

– If Alice can verify the authenticity of the message, she will be able to send a message to Bob secured by their recently created shared key g^{XY}.

1.2 Original H.530 protocol analysis with OFMC and protocol redefinition

As expected, OFMC [2] detects an attack, which trace can be found in *Code 1.* This trace describes how a *man-in-the-middle* attack could be done with this protocol:

– Alice shares her half-key, g^{X_1} , with Bob. However, it is intercepted by an intruder.

– The intruder opens the **first** session with Bob, pretending that he/she is Alice and making up the half-key (X_2 is set to 1, so $g^{X_2} = g$) and Alice's signature.

– Bob asks the trustworthy entity to verify the malicious request, attaching to the message his half-key g^{Y_1}, but it is also intercepted by the intruder.

– The intruder now reproduces the original message from Alice and sends it to Bob by opening the **second** session.

– Bob asks again the trustworthy entity to verify this duplicated request, attaching to his message a new half-key, g^{Y_2}. The intruder forwards the message to the trustworthy entity and intercepts its response.

– The trustworthy entity confirms that both Alice and Bob can be trusted. However, it does not sign any half-key to be used: Bob does not know that the original key, g^{X_1}, is the one that should be trusted.

Figure 2 shows a sequence diagram that illustrates the attack.
In *"h530-fix.AnB"*, this issue is solved by making the trustworthy entity to sign not only Alice and Bob's identity but also the half-key g^X A have decided to use for this session. With this small change, the security threat disappears: OFMC does not detect any attack.
A sequence diagram with the changes made to the protocol is shown in *Figure 3.*

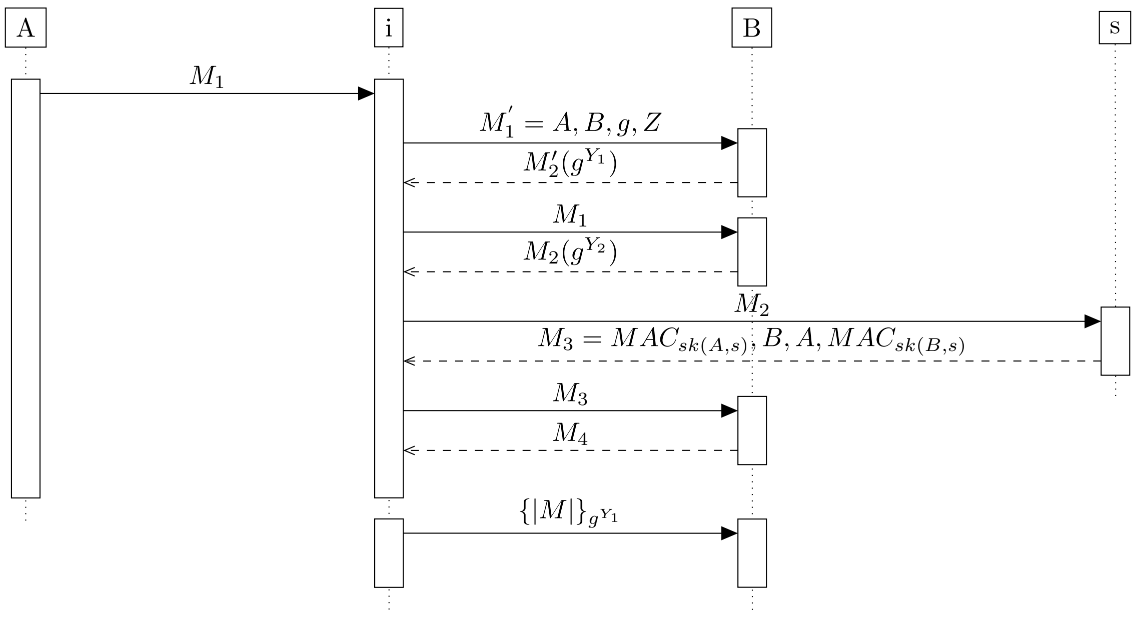


Figure 2: H.530 attack sequence diagram

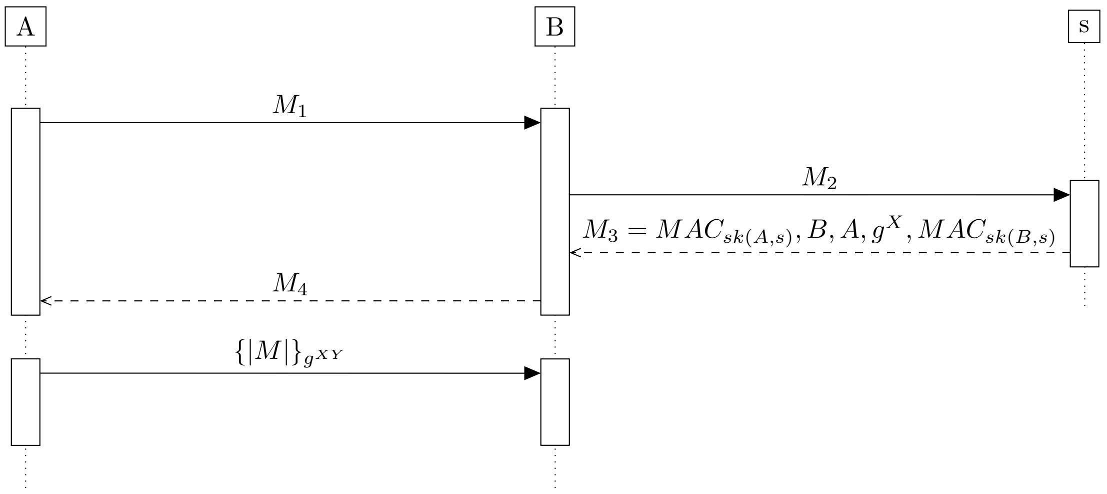


Figure 3: H.530 sequence diagram (fixed)

1.3 Fixed H.530 protocol with untrustworthy third party entity

By doing that, we are saying that the trustworthy entity cannot be trusted anymore: we cannot determine that an identity verification from S is safe. As expected, once we have changed the entity and examined with OFMC, the simulator detects a potential attack, which steps are shown in *Figure 4*.

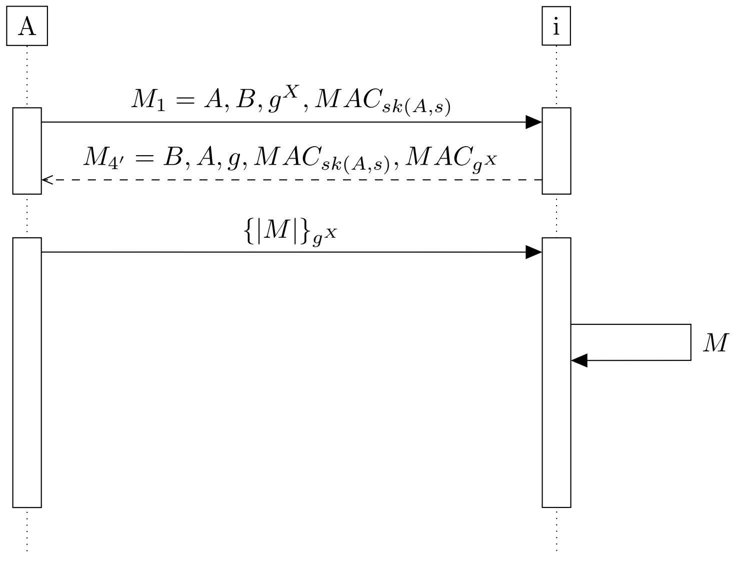

A
i
$M_1 = A, B, g^X, MAC_{sk(A,s)}$
$M_{4'} = B, A, g, MAC_{sk(A,s)}, MAC_{g^X}$
$\{|M|\}_{g^X}$
M

Figure 4: H.530 attack sequence diagram (untrustworthy third party entity)

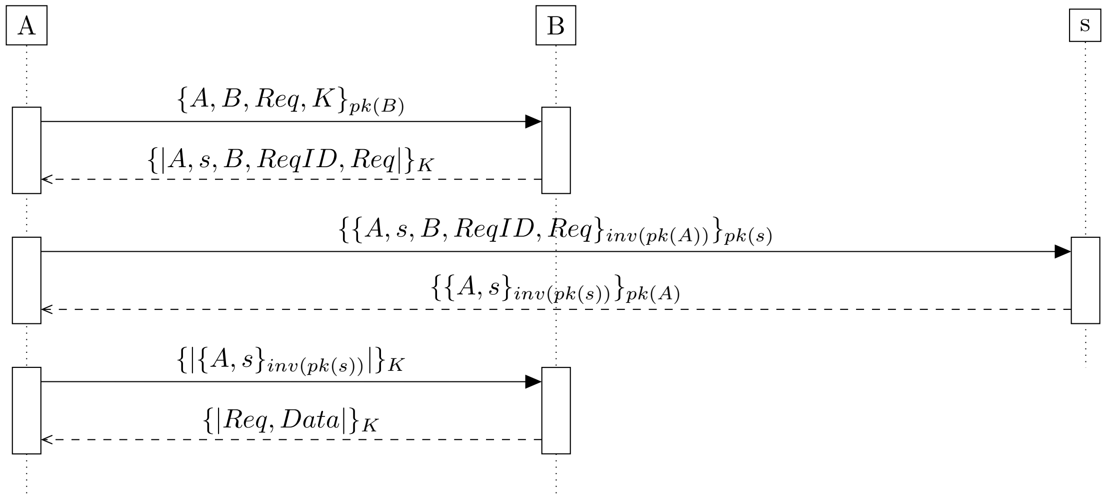
- Alice shares her half-key, g^X, with Bob. However, it is intercepted by an intruder.
- The intruder pretends to be Bob and the third party entity at the same time, sending to Alice a weak half-key $g^{Y=1} = g$. It reuses the signature provided by Alice in the first message and signs the message with their new insecure shared key $g^{XY} = g^X$, which could be simply extracted from Alice's original message. From the point of view of Alice, the message is legit, as it appears to be signed by the trustworthy entity and by Bob with their recently created shared key.
- Alice sends an encrypted message to the intruder, believing that the receiver is Bob. The intruder is able to decrypt the message and reads its content.

This attack trace shows that, if the trustworthy server can be substituted by a *man-in-the-middle* attack, it would be possible for an attacker to open a private, secure session with Alice, pretending that it is Bob the one at the other side.

2 AMP Protocol

2.1 AMP protocol description

The AMP protocol establishes a shared key between Alice and Bob, allowing them to communicate in a secure way.
Figure 5 shows the AMP protocol sequence diagram.



A
B
s
$\{A, B, Req, K\}_{pk(B)}$
$\{|A, s, B, ReqID, Req|\}_K$
$\{\{A, s, B, ReqID, Req\}_{inv(pk(A))}\}_{pk(s)}$
$\{\{A, s\}_{inv(pk(s))}\}_{pk(A)}$
$\{|\{A, s\}_{inv(pk(s))}|\}_K$
$\{|Req, Data|\}_K$

Figure 5: AMP sequence diagram

– Initially, Alice knows Bob's public key. However, Bob does not know Alice's.

– Alice sends a request to Bob. In this request, Alice attaches a request number and her desired shared key. The message is encrypted using Bob's public key, so only Bob is able to decrypt it.

– Bob responds to Alice with a message containing a request identification number for Alice's request. He also includes the trustworthy third party entity in charge of verifying Alice's identity. This third party is also trusted by Alice, so she will also be able to verify Bob's identity.
  The message is encrypted using the key shared by Alice and Bob, so only Alice is able to read it.

– Alice sends a message which contains Alice's and Bob's identity to the third party entity. The message also includes the request number (created at the very beginning by Alice) and the request identification number (created by

Bob).
This message is firstly signed by Alice using her secret key, which ensures that Alice and only Alice was the author of the message. Finally, the message is encrypted using the trustworthy entity's public key, ensuring that only this entity is able to read its content.

– The third party responds to Alice with a message in which Alice is said to be trustful. The message is signed using the entity's private key, ensuring that it was the one that created the message. It is also encrypted using Alice's public key, so only Alice is able to read it.

– Alice re-sends to Bob the message that the trustworthy entity has signed, encrypted this time with their new shared key. Now, Bob can check that the trustworthy entity verifies Alice's identity, so he can start sending data to Alice using their brand new shared key.

2.2 AMP protocol analysis with OFMC

Inspecting the protocol by using OFMC, a possible attack is found.
A sequence diagram of the detected attack is shown in *Figure 6.*

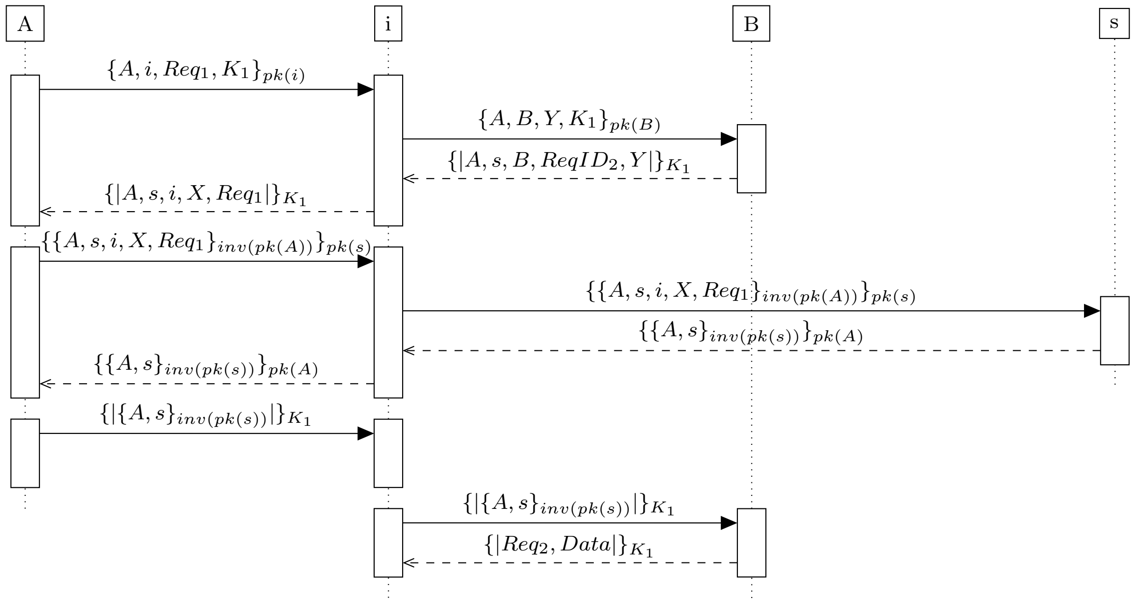

A
i
B
s
$\{A, i, Req_1, K_1\}_{pk(i)}$
$\{A, B, Y, K_1\}_{pk(B)}$
$\{|A, s, B, ReqID_2, Y|\}_{K_1}$
$\{|A, s, i, X, Req_1|\}_{K_1}$
$\{\{A, s, i, X, Req_1\}_{inv(pk(A))}\}_{pk(s)}$
$\{\{A, s, i, X, Req_1\}_{inv(pk(A))}\}_{pk(s)}$
$\{\{A, s\}_{inv(pk(s))}\}_{pk(A)}$
$\{\{A, s\}_{inv(pk(s))}\}_{pk(A)}$
$\{|\{A, s\}_{inv(pk(s))}|\}_{K_1}$
$\{|\{A, s\}_{inv(pk(s))}|\}_{K_1}$
$\{|Req_2, Data|\}_{K_1}$

Figure 6: AMP attack sequence diagram

- Alice sends a request to the intruder. In the request, it can be found the shared key Alice wants to use.
- The intruder sends a request to Bob, pretending that he/she is Alice. The shared key proposed by the intruder for the secure channel is the same Alice sent previously to the intruder.
- Bob sends to the intruder the trustworthy entity's address.
- The intruder sends the third party entity's address to Alice.
- Alice tries to establish a session with the third party, so it can verify Alice's identity. However, the session is intercepted by the intruder, who just forwards the message to the trustworthy entity.
- The third party entity recognises Alice's signature, so it signs Alice authenticity and sends the signature back to the intruder. The intruder re-sends this message to Alice.
- Alice sends to the intruder the signature of the trustworthy entity, this time encrypted by their shared key. As the intruder is able to decrypt the message, he/she has access to the trustworthy identity's signature.
  The intruder re-sends this signature to Bob using their shared key.
- As the trustworthy entity only signed that Alice can be trusted, and Bob believes that the intruder is Alice, he sends to the intruder confidential data.

This attack shows that not only can the intruder get confidential information from Bob, but also impersonate the identity of Alice.

2.3 AMP protocol fix

This security threat can be easily fixed if the trustworthy third party entity signs not only that Alice can be trusted, but also the request number and the request ID under which Alice wants to establish a secure channel with Bob.
In this way, the third party entity will sign that Alice can be trusted by Bob if and only if the request and request identification number coincide with certain values, previously negotiated between both Alice and Bob.

Code 1 shows the *AnB* source code for the fixed AMP protocol.
After analysing this new protocol, OFDM did not find any threat for a maximum of two sessions.

```
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,Request}inv(pk(s)) }pk(A)

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

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

Code 1: AnB code for fixed AMP protocol

2.4 Fixed AMP protocol with untrustworthy third party entity

Again, if we consider that the third party entity is no longer trustworthy, as in *Subsection 1.3*, OFMC detects a security threat, which sequence diagram is depicted in *Figure 7*.

In this case, the intruder is the third party entity itself.

- The intruder starts a session with Bob and claims to be Alice wanting to establish a secure channel using the AMP protocol.
- Bob responds to "*Alice*" that she needs first to demonstrate her trustworthiness via the third party entity, which, in this case, is the intruder.
- As expected, being the intruder the one that has to sign Alice's authenticity, it is very easy to him/her to cheat Bob, who thinks that it is Alice the one at the other side of the session.

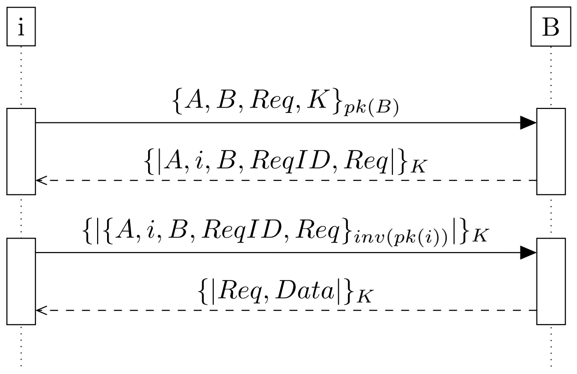

i
B
$\{A, B, Req, K\}_{pk(B)}$
$\{|A, i, B, ReqID, Req|\}_K$
$\{|\{A, i, B, ReqID, Req\}_{inv(pk(i))}|\}_K$
$\{|Req, Data|\}_K$

Figure 7: AMP attack sequence diagram (untrustworthy third party entity)

After analysing the problem, and providing that Bob initially does not know anything about Alice, I could not find any way of avoiding this security threat.
As previously said, Bob does not have Alice's public key, possibly because they have never established contact. For this reason, if the third party entity is not trustworthy, there is not any way of verifying Alice's identity.

References

[1] S. Mödersheim, “Introduction to Protocol Security,” *Campus Net*, 2018.

[2] D. Basin, S. Mödersheim, and L. Viganò, “OFMC: A symbolic model checker for security protocols,” *International Journal of Information Security*, 2005.

[3] N. Søborg, “LaTeX AnB Listings Syntax.” `https://github.com/NicolaiSoeborg/LaTeX-AnB-Listings-Syntax`, 2018.