

# **DISPOSITION 4: NETWORK SECURITY MECHANISMS**

---

Mathias Ravn Tversted

January 16, 2020

# TABLE OF CONTENTS

Authenticated Key Exchange

Needham-Schröder Attack

SSL/TLS

Diffie Hellman and IPSEC

Difference between SSL and IPSEC

# **AUTHENTICATED KEY EXCHANGE**

---

## DEFINITION OF AKE

An *authenticated key exchange protocol* is a protocol for two parties  $A, B$ . Each party starts the protocol with the intention of establishing a key with some other party. At the end, each party outputs either "accept" or "reject" as well as a key. The protocol is said to be secure if the three conditions hold.

**Agreement:** Assume that  $A$  intends to talk to  $B$  and vice versa. Assume also that both parties accept  $A$  and outputs key  $K_A$  while  $B$  outputs key  $K_B$ , then  $K_A = K_B$ .

# SECRECY AND AUTHENTICATION

**Secrecy and Authentication:** Assume  $A$  intends to talk to  $B$ , and accepts. Then it must be the case that  $B$  participated in the protocol and if  $B$  also accepts, he did indeed intend to talk to  $A$ . Furthermore, the adversary does not know the key  $K$  that  $A$  outputs. A symmetric condition holds for  $B$ .

**Freshness:** if  $A(B)$  follows the protocol and accepts, it is guaranteed that the key  $K$  that is output is a fresh key, i.e., it has been randomly chosen for this instance of the protocol, independently of anything else.

These properties give us some desirable properties, such as the preventing the following attacks:

- Adversary cannot stop last message in the protocol for party *A* but not party *B*
- Freshness means adversary cannot trick you into using old key
- Worst thing adversary can do is attempt to block communication



# NEEDHAM-SCHRÖDER ATTACK

---

# THE N-S PROTOCOL

1. A chooses nonce  $n_A$  and sends  $E_{pk_B}(ID_A, n_A)$  to B.
2. B decrypts, checks  $ID_A$ , chooses nonce  $n_B$  and sends  $E_{pk_A}(n_A, n_B)$  to A.
3. A decrypts and checks that the correct value of  $n_A$  appears in the result, and sends  $E_{pk_B}(n_B)$  to B.
4. B decrypts and checks that the correct value of  $n_B$  appears in the result.
5. If the checks are executed are OK, each party computes the session key as some fixed function of  $n_A, n_B$ .

If some third-party  $E$  has a certified public key, they can mix two instances of the protocol and fool  $B$ .

1.  $A$  starts a session with  $E$ .  $A$  sends  $E_{pk_E}(ID_A, n_A)$  to  $E$
2.  $E$  decrypts this and starts session with  $B$ , pretending to be  $A$ .  $E$  sends  $E_{pk_B}(ID_A, n_A)$  to  $B$
3.  $B$  decrypts and finds the right ID and sends  $E_{pk_A}(n_A, n_B)$  to  $E$
4.  $E$  is not able to decrypt, but forwards message  $E_{pk_A}(n_A, n_B)$  to  $A$
5.  $A$  will decrypt, and find an expected result, and will return  $E_{pk_E}(n_B)$  to  $E$
6.  $E$  can decrypt it, find  $n_B$  and can send  $E_{pk_B}(n_B)$  to  $B$
7. When  $B$  decrypts it, he can will accept because it's the right value of  $n_B$ .

# SSL/TLS

---

## SSL protocol for authenticated key exchange

- Uses digital signatures in addition to encryption, as opposed to Needham-Schroeder
- Basic idea: Party must sign a nonce chosen by the other
- Replaced by Transaction Layer Security (TLS), but still referred to as SSL due to minor differences
- SSL is secure against the Needham-Schroeder attack
- SSL sits between application and TCP/IP Transport Layers

# SSL PROTOCOLS

SSL is comprised of the following protocols

- **Record Protocol:** Responsible for "raw" transmission of data. *cipher spec* determines what kinds of encryption algorithms to use
- **Handshake protocol:** The part of SSL that does the authenticated key exchange
- **Change cipher spec protocol:** Changes the cipher spec
- **Alert protocol:** Error messages

# ONE-SIDED SSL HANDSHAKE

- Client picks random  $n_C$
- Server picks random  $n_S$
- Server sends  $n_S, Cert_S$
- Client picks random  $pms$ , sends  $E_{pk_S}(pms)$
- Server computes  $K = f(pms, n_C, n_S)$  and sends  $MAC_K(view_S)$
- Client sends  $MAC_K(view_C)$

# HANDSHAKE PROTOCOL

The actual key-exchange protocol works as follows. This is but one of the variations of the handshake protocol

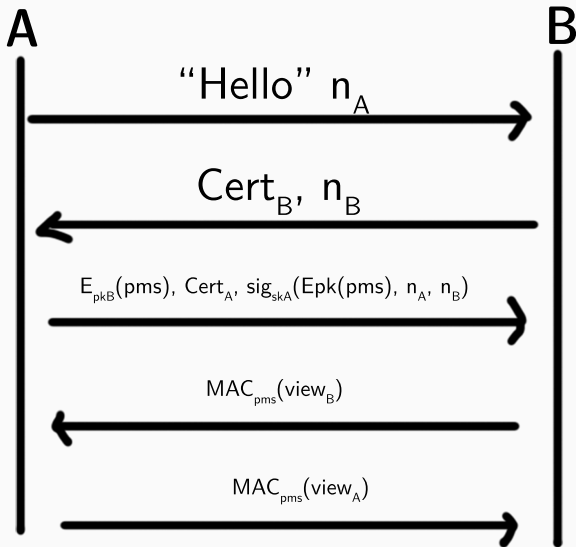
- C sends a hello message containing nonce  $n_C$
- S sends a nonce  $n_S$  and its certificate  $Cert_S$  (containing public key  $pk_S$  of S)
- C verifies  $Cert_S$  and chooses a so called pre master secret  $pms$  at random. C sends  $E_{pk_S}(pms)$  to S, also C sends its certificate  $Cert_C$  to S, plus its signature  $sig_C$  on the concat of  $n_C$ ,  $n_S$  and  $E_{pk_S}(pms)$
- S verifies  $Cert_C$  and  $sig_C$ . If OK; it decrypts  $pms$ .
- S sends to C a "finished" message containing essentially a MAC all messages he has sent and received sent in this instance of the protocol, with  $pms$  the secret key



# HANDSHAKE PROTOCOL

- S sends to C a "finished" message containing essentially a MAC all messages he has sent and received sent in this instance of the protocol, with  $pms$  the secret key.
- C verifies the MAC, and if OK returns its own finished message to S, also with a MAC on all messages sent and received up to this step (note that this now includes the finished message from S, so one cannot just repeat the previous message). S verifies the MAC when it is received.
- At this point both parties derive from  $n_S$ ,  $n_C$  and  $pms$  a set of keys for secret-key authentication and encryption of the following data exchange. Typically this is done using a hash function.

It is important to point out that there are MAC's on all messages sent. This means that the attacker cannot modify messages sent. This doesn't prove that the interleaving attack doesn't work.



## WHY IS IT CORRECT

Consider honest client  $C$  who wants to talk to  $S$ .  $C$  sends  $pms$  encrypted under  $pk_S$ . Only  $S$  will be able to decrypt this. Protocol never asks  $S$  for  $pms$  When  $C$  receives a message with a MAC that verifies using  $pms$ , he knows this MAC comes from  $S$ . The MAC can be verified w.r.t. all messages  $C$  has sent and received it must be the case that  $S$  has seen this same of messages on his side. An adversary that sits between  $C$  and  $S$  can have done nothing except forward all messages unchanged.

Conversely, let us see things from the point of view of an honest  $S$  who wants to talk to  $C$ . He sees a signature from  $C$  on a ciphertext  $c$  and the nonces, and since he chose one of the nonces himself, he knows the signature and hence  $c$  was indeed produced now by  $C$ , i.e., it is not a

# **DIFFIE HELLMAN AND IPSEC**

---

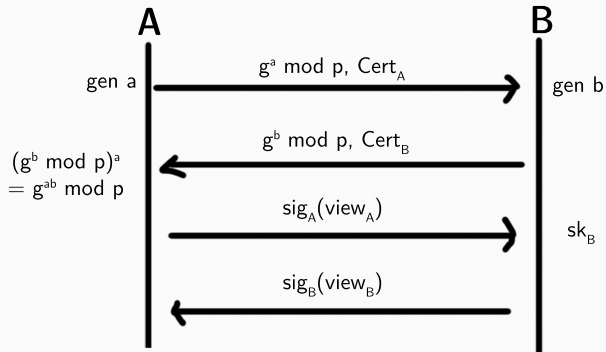
- Sits on Transport Layer
- Sets up secure connection, known as *Security Association*
- *Internet Key Exchange* (IKE)
- IKE is independent of IPSEC
- Mostly uses the *Diffie-Hellman Key Exchange*

## (AUTHENTICATED) DIFFIE-HELLMANN KEY EXCHANGE

Choose  $g \in [0, p - 1]$ . Then execute the following protocol

- C chooses a random number  $a$  and sends  $g^a \bmod p$  and certificate to S.
- S chooses  $b$  and sends  $g^b \bmod p$  and certificate to C
- C computes  $(g^b \bmod p)^a \bmod p$  based on what he got from S and sends own random choice
- S computes  $(g^a \bmod p)^b \bmod p$  These values are equal to  $g^{ab} \bmod p$ . This is now the common key
- C, S sign all messages they have seen so far and send them to each other
- If both verify, then everything is OK

Vulnerable to preprocessing attack *Index Calculus*





# **DIFFERENCE BETWEEN SSL AND IPSEC**

---

# DIFFERENCE BETWEEN SSL AND IPSEC

**Forward Security definition:** "Forward security means that the security of the session you do now cannot be compromised even if the adversary steals a long-term secret key some time in the future"

- SSL protects applications, even from other applications or users on the machine
- Applications must know how to use SSL
- IPSEC protects *all* applications, even if they are not aware of it.
- Data is always never secure once it leaves the encrypted tunnels
- SSL allows one-way authentication, so that the client doesn't need a certificate
- But client can use a password

## PASSWORD AUTHENTICATED KEY EXCHANGE

Password authenticated key exchange can be used to authenticate a user, so it's not just secure, but also the client is authenticated. The following is based on Diffie-Hellmann and is a simplified version of Bellare et al.

- C chooses random number  $a$  and sends  $A = E_{pw}(g^a \bmod p)$
- S chooses  $b$  at random and sends  $B = E_{pw}(g^b \bmod p)$
- C decrypts  $B$  with password  $pw$  and computes  $(g^b \bmod p)^a \bmod p$  based on result of decryption and his own random choice
- S computes  $(g^a \bmod p)^b \bmod p$ . These are equal and can be used as common key

# APPLICATIONS, DOCUMENT BASED SECURE FORMATS VERSUS SECURE TUNNELS

Indsæt stuff her. Sikkert ikke tid til det